CS27020 Career Planner Assignment

Samuel Jackson
slj11@aber.ac.uk

February 19, 2013

1 Introduction

In this assignment, we have been tasked to design and create a single user database system to track a series of applications forms being made to a variety of different roles at different companies. For the basis of this assignment we were given an example application form with all the relevant data fields included.

Section two of this document deals with the design and normalization for the proposed database, while section three outlines the PostgreSQL database implementation of the design.

2 Design

This section details the design process that has been used to create the database system. Firstly the attributes required for each application are identified and outlined in un-normalized form. The functional dependencies for each attribute are then identified before proceeding through the stages of first, second and third normalization.

2.1 Un-normalized Form and Functional Dependencies

Using the application form document provided in the brief, the following structure for the database in its un-normalized form is as follows. I have also listed each of the repeating groups of attributes contained in the given un-normalized form.

application(company_name, company_info, role, deadline, req_strength, req_qualification, strength, evidence, satisfaction, qualification, date, grade, personal_aim, relevance, CV, covering_letter, date_sent, response, interview_date, outcome, reflection)

(req_strength) Repeats for each required strength.
(req_qualification) Repeats for each required qualification.
(strength, evidence, satisfaction) Repeats for each personal strength.
(qualification, date, grade) Repeats for each qualification.
(personal_aim, relevance) Repeats for each personal aim.

Note that in the above outline I have shortened some of the attribute names to more concise abbreviations (e.g. Required Strength to ReqStrength).

2.1.1 Assumptions

In the above un-normalized form I have made several assumptions based on the document given by the brief to work from and of my own intuition. They are as follows:

The database is designed to handle the application forms of one user. In the brief there is no mention of the system having to handle any other users and therefore I have made the assumption that this database will only hold applications from one user at any one time.

Another assumption I have made is that, for each of the repeating attributes, the user may want to have more than simply three different entries as shown in the provided application form. I have therefore not considered any limit to these attributes and assume that each application can store an unspecified amount for each of these attributes.

I have also assumed that job applicant will only ever apply to the same role at the same company once. For example, my design cannot be used to apply to the same role at the same company but at another point in time. You could solve this problem by using the deadline for each application to further uniquely identify each application at a specific point in time. You could also accomplish the same result by adding an unique ID for each application.

One further assumption I have made is that there are never more than two roles with the same name in a company at the same time. Again, if this were not the case, we would probably have to look at including more attributes in the primary key, or adding a unique ID field to each record.

Another assumption I have made is that much of the documentation used for each application can be different. For example, the applicant may decide to send a different CV, covering letter or set of strengths/qualifications with each application depending on the role applied for.

2.1.2 Functional Dependencies

Below are the functional dependencies for the un-normalized structure, including the repeating units. The majority of the attributes are dependant on the company name and role. This suggests that company name and role are a composite key and are an excellent candidate for the primary key. This is because all of the data applied for can be uniquely identified by the name of the company and the role applied for within that company.

Note that we cannot simply use just a the company name or just the role to uniquely identify a data because a company may have many roles and the same role may be offered by many companies. Therefore the primary key must be a composite key consisting of both attributes.

Alternative keys could include a large composite key, such as including the deadline to specify which point in time the role was offered, solving the problem with the same company offering the same role in the future, but as I have stated in my assumptions, I believe this system is only designed to by used by one user at one point in time without having to consider the same role being offered twice. Therefore be using a composite key consisting of the company name and the role as my primary key for this design.

(company_name, role) → {company_info, deadline, req_strength, req_qualifications, strength, qualification, personal_aim, CV, covering_letter, date_sent, response, interview_date, outcome, reflection}

company_name → {company_info} strength → {evidence, satisfaction}
qualification → {date, grade}
personal_aim → {relevance}

Working through the functional dependencies in turn, we can see that the company name and role advertised is enough to determine when the application deadline is, the various requirements for the role (Qualifications, Strengths etc.) as well as the documentation that is required for it, such as a CV and covering letter.

Information about the company is obviously dependant on the name of the company and therefore does not require information of the role to determine it.

The attributes evidence and satisfaction can be determined by the just the name of the personal strength as the evidence and satisfaction given by the strength is obviously dependant on what the strength actually is.

Similarly, the attributes of date and grade are dependant on what the qualification actually is.

Finally the relevance of a personal aim can again only be determined once you know what the aim actually is and therefore the relevance of an aim relies of the individual aim alone.

2.2 First Normal Form

Once we identified the primary key to be used and the functional dependencies of the system, we can start normalising the structure. In first normal form we remove any repeating units from the main relation and put them in their own separate relation. Bringing the un-normalized structure given in the previous section to first normal form gives the following relations:

application(company_name, role, company_info, deadline, CV, covering_letter, date_sent, response, interview_date, outcome, reflection)

req_strengths(strength, company_name*, role*)

req_qualifications(qualification, company_name*, role*)

strengths(strength, company_name*, role*, evidence, satisfaction)

qualifications(qualification, company_name*, role*, date, grade)

aims(personal_aim, company_name*, role*, relevance)

In the above scheme I have removed all repeating units found in the un-normalized form along with any attributes that are functionally dependent on them from the application relation and made them into separate relation. This reduces the repetition and redundancy of data across the system. Note that I have also moved the required strengths and qualifications from the application relation as these are also repeating attributes, despite not having any other attributes being functionally dependant on them.

2.3 Second Normal Form

In second normal form we must ensure that every attribute in a relation is fully functionally dependant on the primary key. This means that any attribute that is only partially dependant on the primary key for that relation must be split into a smaller relation that removes the partial dependency. By rearranging the relations specified in the preceding section we get the following structure:

application(company_name*, role, deadline, CV, covering_letter, date_sent, response, interview_date, outcome, reflection)

company(name, information)

req_strengths(strength, company_name*, role*)

req_qualifications(qualification, company_name*, role*)

strengths(strength, company_name*, role*, evidence, satisfaction)

qualifications(qualification, company_name*, role*, date, grade)

aims(personal_aim, company_name*, role*, relevance)

Company name and company information can be moved from the application relation at this stage because it is only partially functionally dependant on the primary key (specifically the company name). This also correlates with the functional dependencies outlined in section 2.1.2. Therefore we can move this to its own relation.

2.4 Third Normal Form

Third normal form requires further modification of the design to remove any transitive dependencies within the design. This is done by splitting off any non key determinants from the relation. In our design this means splitting down many of the repeating units into smaller relations to remove non key determinants. This leaves us with the following final structure:

application(company_name*, role, deadline, CV, covering_letter, date_sent, response, interview_date, outcome, reflection)

company(name, information)

strength_link(strength*, company_name*, role*)

qualification_link(qualification*, company_name*, role*)

strengths(strength, evidence, satisfaction)

qualifications(qualification, date, grade)

req_strengths(strength, company_name*, role*)

req_qualifications(qualification, company_name*, role*)

aims_link(personal_aim*, company_name*, role*)

aims(personal_aim, relevance)

At this stage, I have introduced three new relations which act as linking tables. These are required to remove any transitive dependencies because some attributes are determined by an attribute that is itself determined by the primary key on the primary key (company name and role). An example transitive dependency is as follows:

(company_name, role) → {qualification, date, grade}
qualification → {date, grade}

(company_name, role) → qualification → {date, grade}

To solve this, I have introduced a relation called qualification_link. This only contains the name of the relation and the attributes that make up the primary key. The qualifications table now only contains the name of the qualification (which acts the the key for the relation scheme) along with the date and the grade for the qualification because the date and grade are intuitively only dependent on the qualification name and not on the company name or role. I have modified the strengths and aims in a similar way for the same reasoning.

Note that now the req_qualifications and re_strengths both cannot have transitive dependencies as they both only contain just the primary key and one other attribute. This the same for all three of the link tables. All other relations contain attributes that are only determined by the full key of the relation and not by any sub key or transitive dependency. At this point we cannot re-apply any of the normalization stages to this final structure.

I believe that the scheme presented here is the most optimal structure for the database and that there are no real further optimisations. I was considering that trying to merge strengths and qualifications with required strengths and qualifications, but because required strengths/qualifications do not have any dependant attributes while strengths and qualifications do, there is no way to merge them without having to add Nulls into many fields in the database. I feel that a cleaner structure is achieved by keeping these attributes separate.

3 Implementation

3.1 Database Creation

In this section I provide a full typescript copy of the SQL statements that were executed on the database in order to created the tables and relations required.

Listing 1: PostgreSQL statements used to create the database

```
/*
        CS27020 Career Planner Assignment
        PostgreSQL for creating the CS27020 assigment database
        Author: Samuel Jackson (slj11@aber.ac.uk)
        Date: 10/2/13
*/

-- Create database using the following terminal command:
-- psql -h db.dcs.aber.ac.uk -U slj11 -d cs27020_12_13 < db.sql

--Query to create the Company table
CREATE TABLE company (
        name    varchar(50) PRIMARY KEY,
        information  varchar(2000)
);

--Create a type for the outcome of the result
CREATE TYPE result as ENUM('ACCEPTED', 'REJECTED', 'N/A');

--Query to create the Application table
CREATE TABLE application (
        company_name  varchar(50) REFERENCES company(name),
        role    varchar(50),
        deadline   date,
        CV      varchar(2000),
        covering_letter  varchar(2000),
        date_sent   date,
        response   varchar,
        interview_date  date,
        outcome    result NOT NULL DEFAULT 'N/A',
        feedback   varchar,
        reflection   varchar,
        PRIMARY KEY(company_name, role)
);

--Query to create the Strengths table
CREATE TABLE strengths (
        strength   varchar(50) PRIMARY KEY,
        evidence  varchar(500),
        satisfaction varchar(500)
);

--Linking table for Strengths
CREATE TABLE strength_link (
        strength   varchar(50) REFERENCES strengths(strength),
        company_name varchar(50),
        role   varchar(50),
        FOREIGN KEY (company_name, role) REFERENCES application(company_name, role),
        PRIMARY KEY(strength, company_name, role)
);

--Query to create Qualifications table
CREATE TABLE qualifications (
        qualification   varchar(50) PRIMARY KEY,
        qualification_date date,
        grade    varchar(3) NOT NULL
);

--Linking table for Qualifications
```

```
CREATE TABLE qualification_link (
        qualification  varchar(50) REFERENCES qualifications(qualification),
        company_name varchar(50),
        role   varchar(50),
        FOREIGN KEY (company_name, role) REFERENCES application(company_name, role),
        PRIMARY KEY(qualification, company_name, role)
);

--Query to create the aims table
CREATE TABLE aims (
        company_name  varchar(50),
        role   varchar(50),
        personal_aim varchar(500),
        relevance  varchar(500),
        FOREIGN KEY (company_name, role) REFERENCES application(company_name, role),
        PRIMARY KEY (company_name, role, personal_aim)
);

--Query to create the table of required strengths
CREATE TABLE req_strengths (
        strength   varchar(50),
        company_name varchar(50),
        role   varchar(50),
        FOREIGN KEY (company_name, role) REFERENCES application(company_name, role),
        PRIMARY KEY(strength, company_name, role)
);

--Query to create the table of required qualifications
CREATE TABLE req_qualifications (
        qualification  varchar(50),
        company_name varchar(50),
        role   varchar(50),
        FOREIGN KEY (company_name, role) REFERENCES application(company_name, role),
        PRIMARY KEY(qualification, company_name, role)
);
```

3.1.1 Justification of Data Types

In my database structure, I have used a variety of different types to create the fields used in my database. Here I provide justification of my choices of data types for each attribute used. Where there are two or more attributes representing the same thing in different tables (e.g. part of a foreign key) I will not justify there data type as they are already specified elsewhere.

- **varchar** - I have used the varchar data type for fields in my design where I can potentially limit the number of characters required to a reasonable number. Examples of this include company name, strength, required strength, qualification, required qualification. This is because it can be assumed that each of these attributes can be described in (or abbreviated to) less than 50 characters. There are a couple of exceptions to the 50 character limit discussed below:
  - Attributes such as the personal_aim and relevance of aims and the evidence and satisfaction of strengths have been set to have a character limit of 500. This is because even though it is not certain how large the data in these fields are likely to be, I feel 500 characters is a reasonable amount of space.
  - The grade of a qualification has a character limit of 3. This is so that the field can store both letter grades such as A++, B or C- as well as being able to store degree level qualification grades such as 1:1, 2:1, 2:2 etc.
  - Attributes relating to large data fields that are supposedly going to contain URL links rather than actual data (such as CV and company information), have been set to be a varchar with the length of 2000 characters. This is because web browsers such as IE do not support URL lengths much above this limit.[1]

[1] http://support.microsoft.com/kb/208427

- **date** - The date data type is used wherever an attribute is storing data related to a specific date. Date was chosen over other data types such as timestamp, because only knowledge of the date is required and I am assuming that there is no information regarding specific times that need to be stored. Example fields that use this in the database are qualification_date, interview_date and deadline.

3.2 Testing the Database

In order to thoroughly test the database, I created a selection of queries to input and retrieve data from the system.