CS27020 Career Planner Assignment

Samuel Jackson
slj11@aber.ac.uk

February 27, 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 req_strength).

2.1.1 Assumptions

In the above un-normalized form I have made several assumptions about the attributes of the database based on the example 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 from just 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. If we were to store the applications of two or more users, we would have no way of distinguishing one uses data from another.

One major assumption I have made about the data is that the name of each company is unique. I feel that this is justified because generally speaking, most companies do have distinct names from one another. However, it is possible that two companies may exist with the same name and same role offered (particularly if, for example its the same company but for a position in another country). However, for the purposes of this document, I have assumed all company names to be unique.

Another assumption I have made is that, for each of the repeating attributes and their functional dependencies (such as qualifications, date and grade), 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 a 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, unless the previous application was removed first. 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.

Leading on from this, I have also assumed that the user will not make more than one application to the same company with the same role at the same time. I feel that these two assumptions can be justified by the fact that Jo's job hunt is unlikely to last so long that she begins to apply to the exact same position in the same company again and again.

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. This means that these attributes are not strictly repeating attributes as they may be different each time.

In relation to both the qualifications and required qualifications fields, I have assumed that each qualification name will also specify the type of qualification level. E.g. A application may have the qualifications "GCSE Biology" and "A Level Biology" associated with it. Otherwise, there would be no way to tell the difference between these two qualifications.

Finally, I have assumed that the outcome of field of the document as well as providing feedback with also clearly state the whether the application was successful or not. In my database design, this field will be left as pending while the application is still in review.

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 most of the data for a specific application 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.

I feel that this is a good choice of primary key because it makes good logical sense and because all attributes are either wholly, partially or transitively dependant on this key. It is also fairly concise and therefore reduces the amount of data redundancy (as opposed to using a larger subset of attributes for unique identification).

(company_name, role) → {company_info, deadline, req_strength, req_qualifications, strength, qualification, personal_aim, CV, covering_letter, date_sent, response, interview_date, outcome, feedback, 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 as each of these attributes are dependant upon both the company and a specific role.

Information about the company is obviously only dependant on the name of the company and therefore does not require knowledge 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.

Note that each of these additional functional dependencies are also dependant on the company name and role (excluding the company info, which is only partially dependant), further supporting the argument for using company name and role as a composite primary key.

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, feedback, 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 a 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 & Third 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, feedback, 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(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.

Similarly, we must move both the strengths and qualifications to their own relations as the evidence and satisfaction only depend on the strength and the date and grade only depend on the qualification. None of these attributes depend entirely on the company name, role and strength/qualification and therefore are only partially dependant on their relation's primary key in first normal form. I have also created a linking table for each of them to organise what strengths and qualifications are related to a particular application.

Both required strengths and required qualifications need not be adjusted at this stage because neither required strength nor required qualification is partially dependant on the composite key of company and role.

Likewise, the aims relation does not need to be changed as the relevance of a personal aims depends on the company name, the role and the personal aim itself because the relevance of an aim may change between roles, and companies (e.g. learning python might not be a good aim for a job that specifies it needs a Java programmer, but it is an excellent aim for a Python developer).

Since we have removed all partial dependencies from the relational structure the database design is now in second normal form. It is also the case that the steps taken in this section have removed any transitive dependencies that may have existed in first normal form, and therefore this structure is now automatically in third normal form and normalization is complete.

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 create the tables and relations outlined in the previous section.

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 < create.sql


−−set the date style to be the obviously more sensible DMY format
SET datestyle = "ISO, DMY";

−−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(100) REFERENCES company(name),
        role    varchar(100),
        deadline   date,
        CV       varchar(2000),
        covering_letter  varchar(2000),
        date_sent    date,
        response   varchar(2000),
        interview_date  date,
        outcome    result NOT NULL DEFAULT 'N/A',
        feedback   varchar(2000),
        reflection   varchar(2000),
        PRIMARY KEY(company_name, role)
);

−−Query to create the Strengths table
CREATE TABLE strengths (
        strength   varchar(100) PRIMARY KEY,
        evidence   varchar(500),
        satisfaction varchar(500)
);

−−Linking table for Strengths
CREATE TABLE strength_link (
        strength   varchar(100) REFERENCES strengths(strength),
        company_name varchar(100),
        role   varchar(100),
        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(100) PRIMARY KEY,
        qualification_date date,
```

```
        grade      varchar(3) NOT NULL
);


−−Linking table for Qualifications
CREATE TABLE qualification_link (
        qualification   varchar(100) REFERENCES qualifications(qualification),
        company_name varchar(100),
        role    varchar(100),
        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(100),
        role    varchar(100),
        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(100),
        company_name varchar(100),
        role    varchar(100),
        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(100),
        company_name varchar(100),
        role    varchar(100),
        FOREIGN KEY (company_name, role) REFERENCES application(company_name, role),
        PRIMARY KEY(qualification, company_name, role)
);
```

Below I provide two screenshots showing the the execution of the preceding script on the university database and a listing of the created tables:

Figure 1: Executing the database creation script on the database

Figure 2: The resulting tables created after the scripts execution

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 100 characters. The arbitrary choice of 100 characters was made as this seemed like a fairly reasonable size considering the type of data likely to stored in this field. There are a couple of exceptions to the 100 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, it will most likely require more space than 100 characters, and I feel 500 characters is a reasonable amount of space to specify these characteristics.

  - 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, MSc, Phd 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]

  - Additionally, I have also set some large attributes such as feedback and reflection to be a varchar of 2000 characters because I am unsure if this will entered as a link to the information (see above) or will be directly entered into the database. If the former is the case then 2000 characters are needed to support the maximum practical URL length, while if the latter is the case then 2000 characters should be plenty of space to allow direct input of the data into a record.

- **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.

- **result** - This is a custom enumeration that I have created to represent the state of the application's outcome. The possible values are:

---

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

- ACCEPTED - If the application was successful.
- REJECTED - If the application failed.
- N/A - If the application is pending and has not been accepted or rejected. The outcome field will take this value by default.
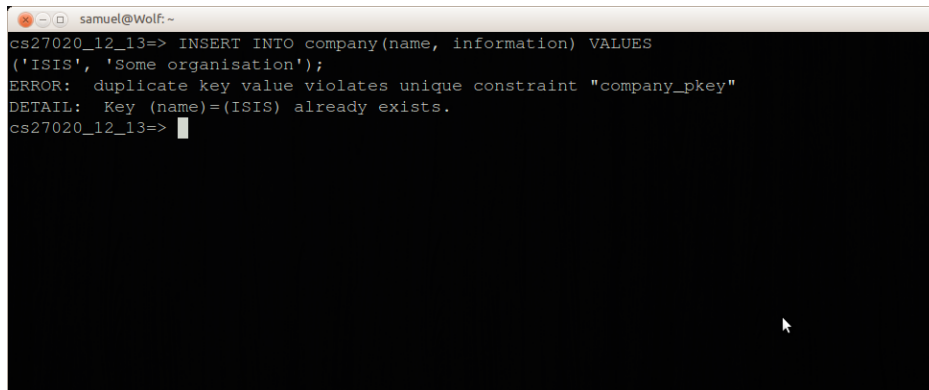
I decided to use an enumeration for this value because I felt this was the simplest way to represent three distinct states that the application can be in. It also leaves the design open for extension, as more states can easily be added in the future (such as a state for pending interview or pending response etc.)

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. I have also created a selection of erroneous queries to demonstrate the constraints placed in effect the attributes.

3.2.1 Testing with Erroneous Queries
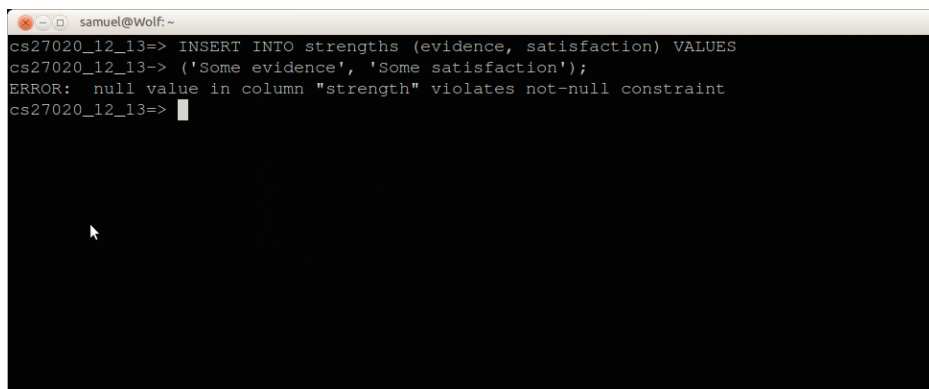
The first query I created demonstrates the primary key constraint that I have used to restrict the company name to be not null and unique. I have used the same technique multiple times across the implementation. When a record is inserted that has a primary key that already exists, the database with throw an error. And example of this is shown below:

Figure 3: Attempting to create a record with a primary key that already exists

Primary key constraints also force the user to always specify a primary key, the following example fails because it leaves of the non-optional primary key field.
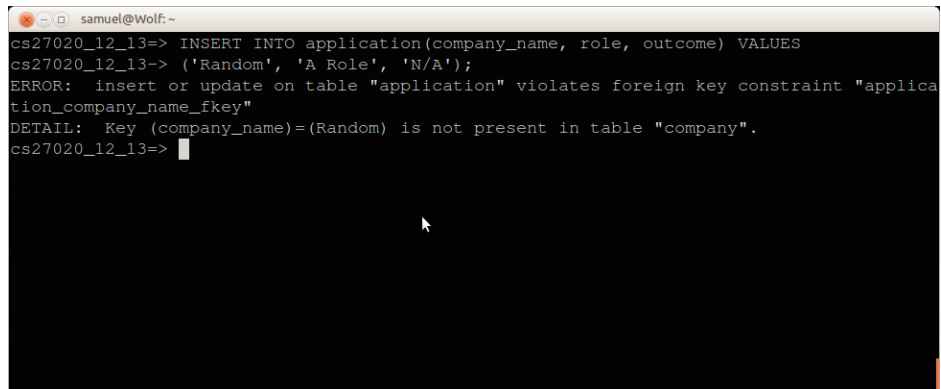
Figure 4: Attempting to create a record where the primary key is empty
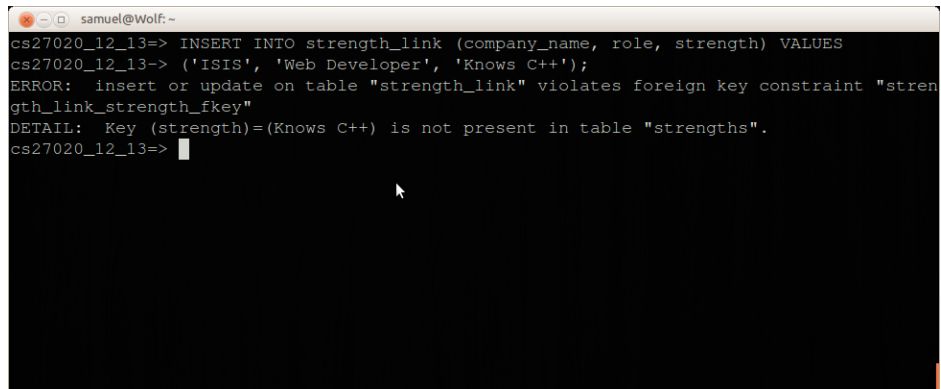
The next query that I have created demonstrates the use of a foreign key constraint on the application's table. Like the use of the primary key constraint, I have used the foreign key constraint across the design to ensure that data is correctly referenced from the right relation and that a record with a foreign key that does not exist is not inserted. Below is an example of this:

Figure 5: Testing the database with a foreign key that does not exist
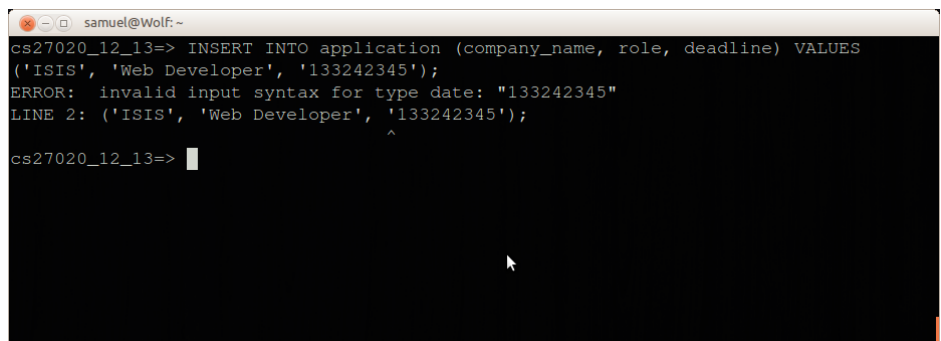
This behaviour can be used to prevent a linking table from connecting a strength or qualification that does not exist to an application, as shown in the following screenshot:

Figure 6: Testing attempting to link a non existent foreign field to an application.

Another query shows an example of the type constraints that I have used as part of my design. Many fields require a specific type of data, such as a date. Setting the attribute to only accept data in the format of a date ensures the database only accepts correctly formatted input. An example of this is shown below:

Figure 7: Testing the date type constraint

3.2.2 Testing the Database Operation

Now that I have shown that the constraints set on the database are working, I have produced several queries to show its operation. Note that this section contains several screen shots of console output with very wide output which is difficult to fit to the width of a page, however the screen shots are of high enough resolution to allow the reader to zoom in and view them more clearly. In order to demonstrate the functionality of the database, I have created an example set of data using the following sql file:

Listing 2: PostgreSQL statements used to insert some example data into the database

```
/*
        CS27020 Career Planner Assignment
        PostgreSQL for inserting some example data into the CS27020 assigment database
        Author:  Samuel Jackson (slj11@aber.ac.uk)
        Date:   13/2/13
*/

−− Insert the data using the following terminal command:
−− psql −h db.dcs.aber.ac.uk −U slj11 −d cs27020_12_13 < insert.sql

−−First make some companies
INSERT INTO company (name, information) VALUES
        ('ISIS', 'Neutron Source'),
        ('HP', 'We make printers'),
        ('Aberystwyth University', 'Teaching and stuff.'),
        ('CERN', 'Smashing things together since 1954');


−− make some of my strengths
INSERT INTO strengths (strength, evidence, satisfaction) VALUES
        ('Punctual', 'I own a watch', 'I love using this strength'),
        ('Teamwork', 'I can work in a team', 'I love working in teams'),
        ('Flexibility', 'I can be very flexible', 'I can touch my toes');

−−make some qualifications up
INSERT INTO qualifications (qualification, qualification_date, grade) VALUES
        ('Computing', '5/8/10', 'A'),
        ('Biology', '5/8/10', 'B+'),
        ('Software Engineering MEng', '2/5/12', '1:1'),
        ('Robotics', '5/8/14', 'MSc');

−−Create a new application
INSERT INTO application (company_name, role, deadline, CV, covering_letter,
        date_sent, response, interview_date, outcome, feedback, reflection) VALUES
        ('ISIS', 'Software Engineer', '1/3/13', 'http://mycv.com', 'http://mycoveringletter.com',
                '2/2/13', 'Link to response', '3/4/13', 'ACCEPTED', 'Some outcome', 'Could have been better'),
        ('ISIS', 'Web Developer', '1/1/13', 'http://asite.com', 'http://anothersite.com',
                '6/8/13', 'Link to response', '7/8/13', 'REJECTED', 'An outcome', 'Who knows'),
        ('HP', 'Web Developer', '3/3/13', 'http://www.somethinghp.com', 'http://www.someotherthing.hp',
                '2/2/13', 'A response', '3/5/13', 'N/A', 'What was the outcome', 'some reflection'),
        ('HP', 'Software Developer', '12/8/13', 'http://www.somethinghp.com', 'http://www.someotherthing.hp',
                '13/4/13', 'A response', '13/8/13', 'N/A', 'Was not good', 'I will do better next time'),
        ('Aberystwyth University', 'Demonstrator', '12/8/13', 'http://aber.ac.uk', 'http://aber.ac.uk',
                '13/4/13', 'A response', '13/8/13', 'REJECTED', 'Failed to pass induction', 'Get better interpersonnal skills');


−−make up some aims
INSERT INTO aims(company_name, role, personal_aim, relevance) VALUES
        ('ISIS', 'Software Engineer', 'Learn to fly a plane', 'Not really very relevant'),
        ('ISIS', 'Software Engineer', 'Learn C++', 'Extremely Relevant'),
        ('HP', 'Web Developer', 'Learn to fly a plane', 'Somewhat relevent');

−−Add some required strengths with an application
INSERT INTO req_strengths (company_name, role, strength) VALUES
        ('ISIS', 'Software Engineer', 'Leadership'),
        ('ISIS', 'Software Engineer', 'Punctual'),
        ('HP', 'Web Developer', 'Punctual');

−−Add some required qualifications with an application
INSERT INTO req_qualifications (company_name, role, qualification) VALUES
```

('ISIS', 'Software Engineer', 'Computing'),
('HP', 'Web Developer', 'Mathematics'),
('ISIS', 'Software Engineer', 'Mathematics'),
('Aberystwyth University', 'Demonstrator', 'Robotics MSc'),
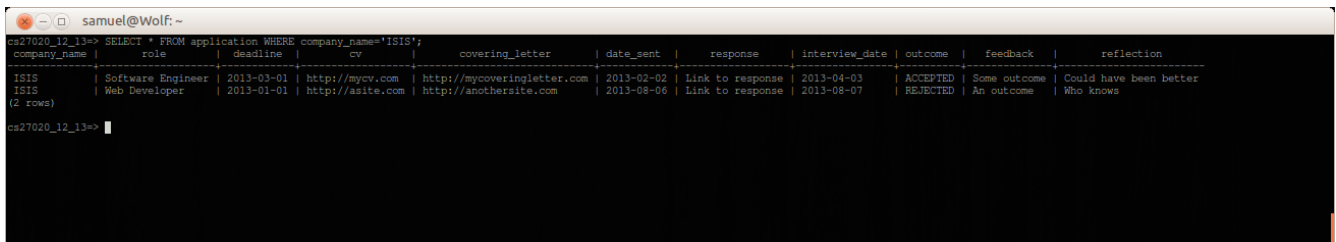('Aberystwyth University', 'Demonstrator', 'Intelligent Systems MSc');

−−Link some strengths with an application
INSERT INTO strength_link (company_name, **role**, strength) **VALUES**
('ISIS', 'Software Engineer', 'Punctual'),
('HP', 'Web Developer', 'Punctual'),
('ISIS', 'Software Engineer', 'Teamwork'),
('Aberystwyth University', 'Demonstrator', 'Flexibility');

−−Link some qualifications with an application
INSERT INTO qualification_link (company_name, **role**, qualification) **VALUES**
('ISIS', 'Software Engineer', 'Computing'),
('HP', 'Web Developer', 'Biology'),
('ISIS', 'Software Engineer', 'Biology'),
('ISIS', 'Software Engineer', 'Software Engineering MEng'),
('Aberystwyth University', 'Demonstrator', 'Robotics');

The following screen captures demonstrate how PostgreSQL queries can be used to find, sort and display the data stored in the database. The first example shows an example of how all applications to a company can be selected from the database using the following query:

SELECT ∗ **FROM** application **WHERE** company_name='ISIS';

Figure 8: Selecting all applications made to a company

Using this type of statement clearly demonstrates how different lists of data can be recalled with minimal effort.

The next query is used to select all qualifications made since the last time an application was made to a given company. This requires the use of two select queries in one statement. This demonstrates how results can be filtered using data from another table, as well as the application of in built postgreSQL (i.e. the max function) and logical operators.

SELECT ∗ **FROM** qualifications **WHERE** qualification_date >
(**SELECT max**(deadline) **FROM** application **WHERE** company_name='ISIS');

Figure 9: Selecting all qualifications obtained since the date of an application

As stated earlier in this document, I have used an enumeration type for the result of an application and split this from the feedback section as given in the brief. This was so that I can easily filter applications by their current status. The following example shows how to get a list of all applications that were rejected.

SELECT ∗ **FROM** application **WHERE** outcome='REJECTED';

Figure 10: Selecting all applications that have been rejected.

The next query shows how data can be ordered based on the fields returned. The following example demonstrates this by selecting all applications where the deadline has not yet passed and orders them by the earliest date.

SELECT ∗ **FROM** application **WHERE** deadline >= **current_date**
        ORDER BY deadline **ASC**;

Figure 11: Selecting all applications open applications and order them by the earliest deadline.

The following, more complex, query demonstrates how data can be selected from two separate tables and combined to produce a composite results. In the example, all qualifications associated with a particular application are selected using a single query.
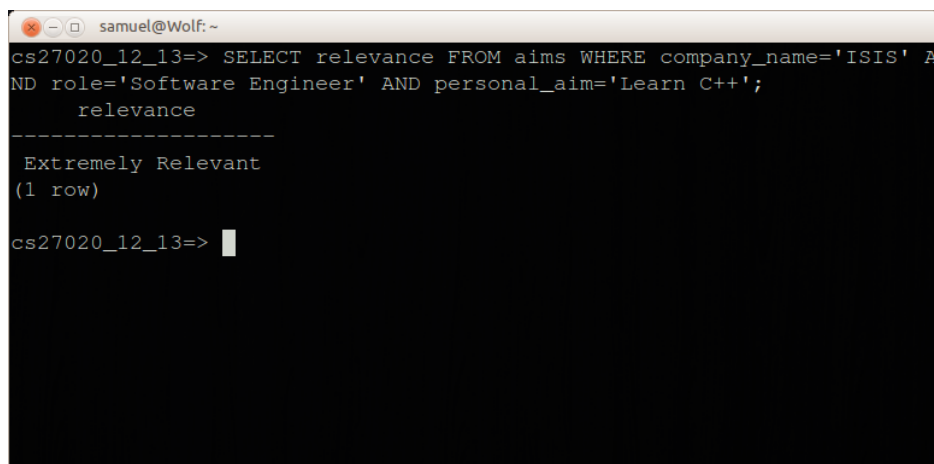
SELECT qualifications.qualification, qualification_date, grade
 FROM qualifications
 INNER JOIN qualification_link
 ON qualifications.qualification = qualification_link.qualification]
 WHERE company_name='ISIS' **AND role**='Software Engineer'
 ORDER BY qualification_date **DESC**;

Figure 12: Selecting all qualifications associated with a particular application.

The next listing and screen shot pair shows how we can use the using the company name, role and an associated aim we can select the relevance of an aim to the application.

SELECT relevance **FROM** aims **WHERE** company_name='ISIS'
 AND role='Software Engineer' **AND** personal_aim='Learn C++';

Figure 13: Select the relevance of a personal aim to the qualification.

The following query shows the use of the composite primary key to select a list of required strengths for a particular application.

13

SELECT ∗ FROM req_strengths
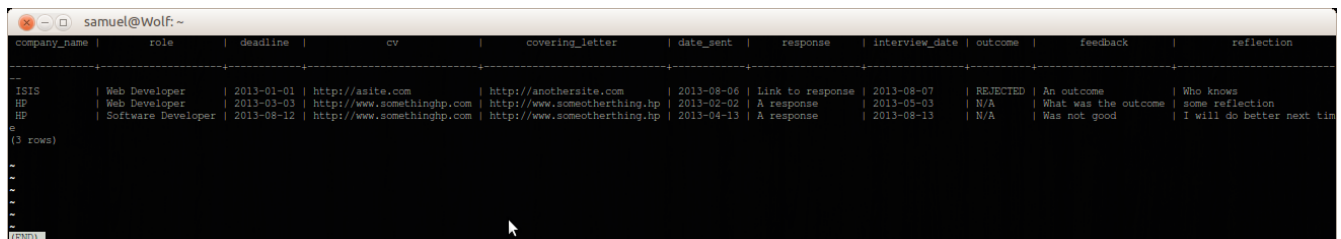   **WHERE** company_name='ISIS' **AND role**='Software Engineer';

Figure 14: Select a list of the required strengths for an application.

One final screen shot shows the a table can be searched for records that contains data that is like what we are looking for. In this example, I use a query to find all applications that have the word developer in them:

SELECT ∗ FROM application **WHERE role LIKE** '%Developer%';

Figure 15: Select a list of job roles containing the keyword "developer".