Craig Heptinstall- Crh13

CS27020 Assignment

Career Planner

Craig Heptinstall- Crh13
3/1/2013

Table of Contents

Objectives

From the given assignment, to show that I have an understanding and more clarity before performing it, I have clarified that I have been asked to:

- Create and un-normalized structured design of a database regarding a client who wishes to keep track of job applications following the given example job application form. I have also been asked to provide its primary key, and any functional dependencies.
- Document the methods I proceeded with in order to bring the structure to third normal form including the use of the functionality dependencies.
- Produce and demonstrate a PostgreSQL database based from my structure, and screenshot a collection of various queries illustrating its functionality.

Scope

This document provides my solution to the given assignment, in designing, re-structuring and implementing a database required and provided by a 'client'. I have made the most out of documenting my work and have attempted the tasks set within the assignment to the best of my ability.

Assignment: Career Planner

Un-normalized structure, with primary key and dependencies

I have chosen a composite primary key, consisting of Company name and of Role. This is because either of the company name or role alone will not suffice as an appropriate identifier for a unique role. For instance, if the primary key was simply the company name, it is possible for more than one job to be applied for in that company. If it was simply the role as the primary key, it is possible that the same role could be applied for at different companies. I have named the entity 'Application' to begin with, which contains all the attributes within an application. This entity is currently in an un-normalized structure.

I have also named appropriately the attributes from the clients example form provided in the assignment, while ensuring the names are still understandable, yet not too long. For instance, I have shortened the fields 'My relevant qualifications' to 'MyRelQual' ensuring it still makes sense. It was important to name these correctly and not confuse others together to prevent myself from being led astray when going through the normalisation process. Below outlines the un-normalized structure:

APPLICATION (<u>comp_name, role</u>, comp_info, deadline, (req_str, req_qual, my_rel_str, str_evid, str_satis, my_rel_qual, my_rel_qual_date, my_rel_qual_grade, my_aim, my_aim_rel), cv, cov_letter, date_sent, response, interv_date, outcome, reflection)

As you can see, I have used the correct notation for the structure, and have bracketed attributes that are repeating groups. The repeating groups are attributes within the application that can have multiple values, for instance a job could have more than one requirement of qualifications. Having repeating groups in the un-normalized structure clarifies that the current structure is not in first normal form. I have broken down the repeated groups in the next section, explaining them further.

Repeating groups:

I have now broken down the repeated groups into their nested versions:

(req_qual, my_rel_qual, my_rel_qual_date, my_rel_qual_grade) Repeats for each qualification

(req_str, my_rel_str, str_evid, str_satis) Repeats for each strength

(my_aim, my_aim_rel) Repeats for each aim

I have placed the required qualification in a repeated group with the relevant qualification on the assumption that the applicant should always fill in a corresponding relevant qualification for each required one. I have done the same with the strengths entity.

Functional dependencies have been portrayed below, which represent attributes that are functionally dependent upon another at any one time. The attributes first in the notations represent the determinant, and any attributes following are the objects of the determinant.

Functional dependencies:

comp_name → comp_info

The functional dependency here states that the company information relies on the name of the company identifying it.

role → {req_str, req_qual, req_qual, cv, cov_letter}

I have created a functional dependency here concerning the role on an application. Dependents on a role are qualifications, strengths and the CV or cover letter for that role. I chose not to include attributes such as deadline, date sent, interview date, response and feedback because these are not solely dependent on the role and need the company name combined with this to be a functional dependency. I have gone on the assumption here that all similar roles should have similar required qualifications and required strengths. An applicant would usually use the same cv or covering letter links too.

req_str → my_rel_str → {str_evid, str_satis}

The strength evidence and strength satisfaction attributes both are functionally dependent on the relative strength attribute, as both hold the information about what evidence of each strength is present and the satisfaction involved. In turn, each relevant strength is dependent on a required strength. I have gone on the assumption here that for each required strength, there is one matching relevant strength.

req_qual → my_rel_qual → {my_rel_qual_date, my_rel_qual_grade}

This functional dependency is similar to the previous one, as each piece of information regarding a qualification relies on the qualification itself, and also from the required qualification.

my_aim → my_aim_rel

The last functional dependency I will describe is the aim and aim relevance dependency. For each aim, there would be a relevance to the application. I have assumed here that this alike the previous two dependencies will rely on a one to one relationship.

How the un-normalized structure was brought to third normal form

As required for the second section of the assignment, I will now attempt to bring the un-normalized structure from first normal form through to third normal form. Ensuring that I achieve third normal form for my entities and attributes will mean a better implemented database with minimal anomalies when data is present. For each stage of the normalisation process, I will ensure to describe each step clarifying how I achieved each stage.

First normal form

To bring my data to first normal form, the data must be free of repeating groups. To complete this, using my repeating groups declared earlier in this assignment I have removed these from the 'Application' set and created new entities for each repeated group. To ensure it is correctly first normal form, the primary key from 'Application' has been copied to the new entities, and then the primary key of each has been identified. For the 'Application qualifications' entity, I have decided to use required qualifications attribute as part of the composite key. This is the same format for the other new entities. You will see that each entity is now named with a '1' on the end to clarify that they are in first normal form.

APPLICATION-1 (comp_name, role, comp_info, deadline, cv, cov_letter, date_sent, response, interv_date, outcome, reflection)

APPLICATION_QUALIFICATIONS-1 ((comp_name, role)*, req_qual, my_rel_qual, my_rel_qual_date, my_rel_qual_grade)

APPLICATION_STRENGTHS-1 ((comp_name, role)*, req_str, My_rel_str str_evid, str_satis)

APPLICATION_AIMS-1 ((comp_name, role)*, my_aim, my_aim_rel)

Second normal form

To ensure that the data moves into second normal form, I have ensured that there are no partial functional dependencies. For example to do this, I needed to check if any attributes from entities of the previous normal form were only dependent on a section of the primary key. In this case, I could see that the 'my_rel_qual' was only dependent on the 'req_qual'. To solve this, I simply moved this to a new entity, copied the current primary key which is the company name and role, and selected the required qualification attribute to be added to this composite key. I repeated this for the other entities to get the structure to second normal form:

APPLICATION-2 (comp_name, role, comp_info, deadline, cv, cov_letter, date_sent, response, interv_date, outcome, reflection)

APPLICATION_REQUIRED_QUALIFICATIONS-2 ((comp_name, role)*, req_qual)

APPLICATION_REQUIRED_STRENGTHS-2 ((comp_name, role)*, req_str)

APPLICATION_AIMS-2 ((comp_name, role)*, my_aim, my_aim_rel)

APPLICATION_MY_QUALIFICATIONS-2(req_qual*, my_rel_qual, my_rel_qual_date, my_rel_qual_grade)

APPLICATION_MY_STRENGTHS-2(req_str*, my_rel_str, str_evid, str_satis)

Third normal form

In the final step of normalisation, the rule that had to be enforced to make sure my structure was in third normal form was if it followed both rules previously and that no attributes were non-transitively dependent on the primary key. From looking at my previous form of the structure, I found that there were no such issues with my entities therefore I should keep them the same. To simply refine the entities ready for implementation, I have removed the number '2' from the same of each entity ready for creating in postgres.

APPLICATION (comp_name, role, comp_info, deadline, cv, cov_letter, date_sent, response, interv_date, outcome, reflection)

REQUIRED_QUALIFICATIONS ((comp_name, role)*, req_qual)

REQUIRED_STRENGTHS ((comp_name, role)*, req_str)

AIMS ((comp_name, role)*, my_aim, my_aim_rel)

MY_QUALIFICATIONS (req_qual*, my_rel_qual, my_rel_qual_date, my_rel_qual_grade)

MY_STRENGTHS (req_str*, my_rel_str, str_evid, str_satis)

Screenshots demonstrating PostgreSQL implementation of database including queries

Now that I have completed bring the data to third normal form, I can implement the database. Looking through the entities from above, I must firstly choose the type of data representing each attribute in each table. For instance, choosing a text based type or char based data type for a descriptive attribute would suit better than a numerical type such as an int. SQL allows a wide range of data types, giving me a great deal of choice on the possible appropriate types. I have shown my data choices for each attribute in the below table:

Chosen attributes data types

Attribute	**Data type chosen**	**Reason**
Comp_name	Char(25)	Company names are usually less than 25 characters
Role	Varchar(30)	Role names are usually short, although it is possible that a description could come with the role, therefore a variable char allows for this
Comp_info	Text	Company information could be a good deal of length, therefor text is best
Deadline	Date	A strict date format would stop entries that are not dates
Cv	Varchar(40)	If the cv is a url, a longer variable char should be allowed, and be variable in case the site name is long.

Cov_letter	Varchar(40)	If the cover letter is a url, a longer variable char should be allowed, and be variable in case the site name is long.
Date_sent	Date	A strict date format would stop entries that are not dates
Response	ENUM	Having an Enum would mean specific values only would be allowed for the response. For instance, (‘yes’, ‘no’) would be possible values.
Interv_date	Timestamp	A time and date is usually required when organising an interview, hence the timestamp option allows for time and date.
outcome	ENUM	An Enum for the outcome would be useful in the case where I wanted only text entered here (‘successful’, ‘failed’ ‘on-going’).
Reflection	Text	A reflection could be any size, so text would suffice
Req_qual	Varchar(40)	A variable char here would mean although it is usually under 40 character as this would be represented by a sentence it would allow for more. This save memory when possible.
Req_str	Varchar(40)	
My_aim	Varchar(40)	
My_aim_rel	Text	More than a sentence could be used here and the length could be unknown.
My_rel_qual	Varchar(30)	A variable char here would mean although it is usually under 30 character as this would be represented by a sentence it would allow for more. This save memory when possible.
My_rel_qual_date	Date	The date format would force a specific date format for data such as the qualification date.
My_rel_qual_grade	Char(5)	Grades are usually in the format of a letter or ‘pass’ so 5 characters would be enough for lots of eventualities.
My_rel_str	Varchar(40)	Variable chars here would mean although it is usually under these character limits as this would be represented by a
Str_evid	Varchar(50)	
Str_satis	Varchar(20)	

		sentence it would allow for more. This save memory when possible.

Creation of database

Now I have decided on the data types to be used within my database, I will now implement it. Using the third normal form list of entities and these data types, it will make it much easier to create the SQL statements required to create each table. Below I have listed the SQL statements containing each of the 'CREATE TABLE' statements. Each of my below table creations follow the same primary and foreign key constraints outlines in my third normal form.

```
 /*
 ENUM used to store specific possible values of the repsonse
 and the outcome of an appplication.
 */
CREATE TYPE response AS ENUM ('yes', 'no');
CREATE TYPE outcome AS ENUM ('successful', 'fail', 'ongoing');

/*
Create the table that holds the main data about aan appplication.
Composit primary key consisting of company name and job role.
*/
CREATE TABLE APPLICATION(
 comp_name char(25),
 role varchar(30),
 comp_info text,
 deadline date,
 cv varchar(40),
 cov_letter varchar(40),
 date_sent date,
 response response,
 interv_date timestamp,
 outcome outcome,
 reflection text,
 primary key(comp_name, role)
 );

/*
Create table containing list of required qualifications for each application.
Foreign key from APPLICATION, to get required qualifications for specific job.
Primary key as the required qualification
*/
CREATE TABLE REQUIRED_QUALIFICATIONS(
 comp_name char(20),
 role varchar(30),
 req_qual varchar(40) primary key,
 foreign key(comp_name, role) references APPLICATION(comp_name, role)
 );

/*

*/
CREATE TABLE REQUIRED_STRENGTHS(
 comp_name char(20),
 role varchar(30),
 req_str varchar(40),
 primary key(req_str),
 foreign key(comp_name, role) references APPLICATION (comp_name, role)
 );
```

```
/*

*/
CREATE TABLE MY_AIMS(
 comp_name char(20),
 role varchar(30),
 my_aim varchar(40),
 my_aim_rel text,
  foreign key(comp_name, role) references APPLICATION (comp_name, role)
 );

/*

*/
CREATE TABLE MY_QUALIFICATIONS(
 req_qual varchar(40),
 my_rel_qual varchar(30) primary key,
 my_rel_qual_date date,
 my_rel_qual_grade char(5),
 foreign key(req_qual) references REQUIRED_QUALIFICATIONS (req_qual)
 );

/*

*/
CREATE TABLE MY_STRENGTHS(
 req_str varchar(40),
 my_rel_str varchar(40),
 str_evid varchar(50),
 str_satis varchar(20),
 primary key (my_rel_str),
 foreign key(req_str) references REQUIRED_STRENGTHS (req_str)
  );
```

As you can see from the above statements, each one of my entities is being implemented as a table, including all the necessary primary and foreign keys. To ensure that the implementation was done as quick as possible, I created a.'SQL' file to include all my statements in, then ran this file in the postgres command line like so:

postgres=# \i 'C:\\Users\\Craig\\Desktop\\CS211\\commands.sql'

And to ensure that each time I re-implemented the tables to test data input etc. , I added a few 'DROP' (including cascading through relations) statements to the top of the SQL file which would remove the previously created tables (including the ENUM types). This was like so:

```
/*
Code Below Simple to empty the database for re-building.
*/
drop table APPLICATION cascade;
drop table REQUIRED_STRENGTHS cascade;
drop table MY_AIMS;
drop table MY_STRENGTHS;
drop table MY_QUALIFICATIONS;
drop table REQUIRED_QUALIFICATIONS cascade;
drop type response;
drop type outcome;

/*
Hide 'NOTICE:' messages on building of database.
*/
SET client_min_messages TO WARNING;
```

You may also see an extra line of code at the bottom of this set of statements using the command 'SET'. I added this simply to remove the NOTICE messages when the various different statements went through the command line. Once this command was added, and all the code was put through the postgres command line, the output was like so:

Inserting data

Now I have completed the implementation of my database, I will now show how I inserted and queried data within each of the tables. I chose to insert a fair amount of data (5 application examples) into the application table, and any relevant strengths, qualifications and aims for each of the applications using the 'INSERT' command in the postgres interface. Alike the creation of the tables, I chose to keep all the INSERT statements within an SQL file which could then be easily run within the command line to insert all data at once. Ensuring that a good amount of data is key for when it comes to testing it using queries. Having too little data in a table would mean that queries would not produce the results that I was expecting. Below displays the SQL file contents for inserting data:

```
/*
Data to be inserted into the application table. Consisting of 5
roles, four of them for the same company, two for the same role
to ensure the database can handle such data.
*/
INSERT INTO APPLICATION (comp_name, role, comp_info, deadline, cv,
 cov_letter, date_sent, response, interv_date, outcome, reflection) values
 ('Apple', 'Lead Tester', 'A large company that makes apples',
 '2013-03-01', 'http://website.co.uk/cv.pdf', 'http://website.co.uk/cov.pdf',
 '2012-12-20', 'yes', '2013-02-01 12:15:00', 'successful',
 'very happy got the job');
INSERT INTO APPLICATION (comp_name, role, comp_info, deadline, cv,
 cov_letter, date_sent, response, outcome) values
 ('Microsoft', 'Trainee Software Engineer',
 'The company behind the creation of windows', '2013-04-01',
 'http://jimmy.co.uk/cv.pdf', 'http://jimmy.co.uk/cov.pdf', '2012-12-20',
 'no', 'ongoing');
```

```
INSERT INTO APPLICATION (comp_name, role, comp_info, deadline, cv,
cov_letter, date_sent, response, interv_date, outcome, reflection) values
 ('Apple', 'Web Developer', 'A large company that makes apples',
 '2013-03-20', 'http://test.co.uk/cv.pdf', 'http://test.co.uk/cov.pdf',
 '2012-12-20', 'yes', '2013-02-28 12:15:00', 'fail',
 'better luck next time i guess.');
INSERT INTO APPLICATION (comp_name, role, comp_info, deadline, cv,
 cov_letter, date_sent, response, outcome ) values
 ('Google', 'Trainee Tester', 'Famous for their search engine',
 '2013-03-10', 'http://alongwebsitename.co.uk/cv.pdf',
 'http://alongwebsitename.co.uk/cov.pdf', '2012-12-20', 'no', 'ongoing' );
INSERT INTO APPLICATION (comp_name, role, comp_info, deadline, cv,
cov_letter, date_sent, response, interv_date, outcome, reflection) values
 ('Ubuntu', 'Trainee Software Engineer', 'An alternative way of running a PC',
 '2013-05-10', 'http://anotherwebsite.co.uk/cv.pdf',
 'http://anotherwebsite.co.uk/cov.pdf', '2012-10-20', 'yes',
 '2013-02-05 10:00:00','successful', 'Turned down the job' );

/*
Data to be inserted into required qualification table. contains
qualifications required for 4 of 5 jobs, and two jobs containing
two required qualifications.
*/
INSERT INTO REQUIRED_QUALIFICATIONS(comp_name, role, req_qual) values
 ('Apple', 'Lead Tester', 'CISCO Software Qualification'),
 ('Microsoft', 'Trainee Software Engineer', 'Computer Science Degree 1st'),
 ('Apple', 'Web Developer', 'Web Standards Award'),
 ('Google', 'Trainee Tester', 'Computer Science Degree 2:1'),
 ('Google', 'Trainee Tester', 'A-Level Maths: C');

 /*
Data inserted into required strengths table,with three strengths for one
job, and a single strength required for the others.
*/
INSERT INTO REQUIRED_STRENGTHS(comp_name, role, req_str) values
('Apple', 'Lead Tester', 'Team Working Skills'),
('Apple', 'Lead Tester', 'Cope Under Pressure'),
('Apple', 'Lead Tester', 'Formal Skills'),
('Microsoft', 'Trainee Software Engineer', 'Good Learning Speed'),
('Google', 'Trainee Tester', 'Good Ability To Lead A Team'),
('Ubuntu', 'Trainee Software Engineer', 'An Interest In Software');

/*

*/
INSERT INTO MY_AIMS(comp_name, role, my_aim) values
 ('Apple', 'Lead Tester', 'Learn insides and out of testing mac os'),
 ('Apple', 'Lead Tester', 'Build team building skills'),
 ('Apple', 'Lead Tester', 'Understand project management better'),
 ('Microsoft', 'Trainee Software Engineer', 'Understand project leadership better'),
 ('Google', 'Trainee Tester', 'Build Test Performance'),
 ('Ubuntu', 'Trainee Software Engineer', 'Understand software methods');
```

```
/*

*/
INSERT INTO MY_QUALIFICATIONS(req_qual, my_rel_qual, my_rel_qual_date, my_rel_qual_grade) values
('CISCO Software Qualification', 'CISCO Software Qualification', '2011-06-28', 'Pass'),
('Computer Science Degree 1st', 'Computer Science Degree ' ,'2012-01-29', '1st'),
('Web Standards Award', 'Web Standards Award','2012-12-01', 'Pass'),
('Computer Science Degree 2:1' ,'Computer Science Degree', '2013-01-20', '1st'),
('A-Level Maths: C', 'A-Level Maths','2010-07-01', 'C');

/*

*/
 INSERT INTO MY_STRENGTHS(req_str, my_rel_str, str_evid, str_satis) values
('Team Working Skills', 'Team Working Skills',
'Worked in team of nine in a project in university',
'Thoughrougly Enjoyed'),
('Cope Under Pressure', 'Cope Under Pressure',
'Had many assignments at once during my degree',
'Good feeling know i can handle these situations'),
('Formal Skills', 'Formal Skills', 'Worked in a professional envroment',
'Do not mind if dress code is formal or not'),
('Good Learning Speed', 'Good Learning Speed',
'Had to pick up certain skills fast for modules', 'Enjoy learning new things'),
('Good Ability To Lead A Team', 'Good Ability To Lead A Team',
'Was a deputy leader in a university project',
'Enjoyed organisin, keeping track of team members'),
('An Interest In Software', 'An Interest In Software',
'Always looking at software in free time', 'Love looking at software');
```

Performing suitable queries

Now I have implemented and filled my database with sample data, I can now perform a range of queries on the different tables and data. For each query, I will provide screenshots of each being performed in the command line, including an explanation for each stating the reason I performed such a query, and the result of each. I have ensured to spread queries out as much as possible across my database and test lots of possibilities. (You can find results in full view in a link in the summary)

Application Queries

SELECT * FROM APPLICATION;

In the first query, I wanted to simply check the whole job data for each entry existed:

```
 comp_name  |          role          |              comp_info              |  deadline  |                cv                 |
------------+------------------------+-------------------------------------+------------+-----------------------------------+
 Apple      | Lead Tester            | A large company that makes apples   | 2013-03-01 | http://website.co.uk/cv.pdf       | http
 Microsoft  | Trainee Software Engineer | The company behind the creation of windows | 2013-04-01 | http://jimmy.co.uk/cv.pdf | http
 Apple      | Web Developer          | A large company that makes apples   | 2013-03-20 | http://test.co.uk/cv.pdf          | http
 Google     | Trainee Tester         | Famous for their search engine      | 2013-03-10 | http://alongwebsitename.co.uk/cv.pdf | http
 Ubuntu     | Trainee Software Engineer | An alternative way of running a PC | 2013-05-10 | http://anotherwebsite.co.uk/cv.pdf | http
(5 rows)
```

SELECT * FROM APPLICATION WHERE OUTCOME='FAIL';

In this query, it tests to see if any entries have failed as job applications:

```
 comp_name  |     role      |          comp_info          |  deadline  |          cv           |       cov_letter       | date_sent
------------+---------------+-----------------------------+------------+-----------------------+------------------------+-----------
 Apple      | Web Developer | A large company that makes apples | 2013-03-20 | http://test.co.uk/cv.pdf | http://test.co.uk/cov.pdf | 2012-12-20
(1 row)
```

```
SELECT COMP_NAME FROM APPLICATION WHERE ROLE='TRAINEE SOFTWARE ENGINEER';
```

This query checks what companies jobs are marked as a trainee software engineer and returns the companies:

```
  comp_name
-------------
 Microsoft
 Ubuntu
(2 rows)
```

```
SELECT * FROM APPLICATION WHERE DEADLINE<'2013-04-01' ORDER BY DEADLINE;
```

This query was used to check if the date type correctly worked to identify jobs in a specific deadline:

```
 comp_name |    role    |          comp_info          |  deadline  |                cv                |           cov_letter
-----------+------------+-----------------------------+------------+----------------------------------+--------------------------------
 Apple     | Lead Tester    | A large company that makes apples | 2013-03-01 | http://website.co.uk/cv.pdf          | http://website.co.uk/cov.p
 Google    | Trainee Tester | Famous for their search engine    | 2013-03-10 | http://alongwebsitename.co.uk/cv.pdf | http://alongwebsitename.co
 Apple     | Web Developer  | A large company that makes apples | 2013-03-20 | http://test.co.uk/cv.pdf             | http://test.co.uk/cov.pdf
(3 rows)
```

```
SELECT COMP_NAME, ROLE, COMP_INFO FROM APPLICATION WHERE COMP_NAME='APPLE';
```

This application of a query shows searching jobs under a certain company:

```
 comp_name |    role    |          comp_info
-----------+------------+-----------------------------
 Apple     | Lead Tester    | A large company that makes apples
 Apple     | Web Developer  | A large company that makes apples
(2 rows)
```

```
SELECT COMP_NAME, ROLE, RESPONSE FROM APPLICATION WHERE RESPONSE='NO';
```

This query is useful for checking if the ENUM of ‘no’ was working as intended:

```
 comp_name |          role          | response
-----------+------------------------+----------
 Microsoft | Trainee Software Engineer | no
 Google    | Trainee Tester            | no
(2 rows)
```

Required qualification queries

```
SELECT * FROM REQUIRED_QUALIFICATIONS;
```

This simple checks to see if all the required qualification entries was in the table correctly:

```
 comp_name |          role          |          req_qual
-----------+------------------------+-----------------------------
 Apple     | Lead Tester            | CISCO Software Qualification
 Microsoft | Trainee Software Engineer | Computer Science Degree 1st
 Apple     | Web Developer          | Web Standards Award
 Google    | Trainee Tester         | Computer Science Degree 2:1
 Google    | Trainee Tester         | A-Level Maths: C
(5 rows)
```

```
SELECT * FROM REQUIRED_QUALIFICATIONS WHERE ROLE='LEAD TESTER';
```

I created this query to check all the companies and required qualifications concerning a certain role:

```
 comp_name |    role    |          req_qual
-----------+------------+-----------------------------
 Apple     | Lead Tester | CISCO Software Qualification
(1 row)
```

```
SELECT ROLE FROM REQUIRED_QUALIFICATIONS WHERE REQ_QUAL='COMPUTER SCIENCE DEGREE 1ST';
```

Here I used a query to test if the table could identify what roles required a specific qualification:

```
          role
-------------------------
 Trainee Software Engineer
(1 row)
```

My required strength queries

SELECT * FROM REQUIRED_STRENGTHS;

I created a query here to check if the required strengths table was populated correctly:

```
     comp_name      |          role          |          req_str
--------------------+------------------------+---------------------------
 Apple              | Lead Tester            | Team Working Skills
 Apple              | Lead Tester            | Cope Under Pressure
 Apple              | Lead Tester            | Formal Skills
 Microsoft          | Trainee Software Engineer | Good Learning Speed
 Google             | Trainee Tester         | Good Ability To Lead A Team
 Ubuntu             | Trainee Software Engineer | An Interest In Software
(6 rows)
```

SELECT * FROM REQUIRED_STRENGTHS WHERE ROLE='LEAD TESTER';

This query was used to see what requirements of strengths a certain role would require:

```
     comp_name      |    role     |       req_str
--------------------+-------------+---------------------
 Apple              | Lead Tester | Team Working Skills
 Apple              | Lead Tester | Cope Under Pressure
 Apple              | Lead Tester | Formal Skills
(3 rows)
```

SELECT ROLE FROM REQUIRED_STRENGTHS WHERE REQ_STR='FORMAL SKILLS';

I created a query to find out what roles would require a certain strength:

```
    role
------------
 Lead Tester
(1 row)
```

My aim queries

SELECT * FROM MY_AIMS;

This query was a check to see if the my_aims table had been correctly populated:

```
     comp_name      |          role          |                my_aim                |        my_aim_rel
--------------------+------------------------+--------------------------------------+--------------------------
 Apple              | Lead Tester            | Learn insides and out of testing mac os | Helps Testing Macs
 Apple              | Lead Tester            | Build team building skills           | Will Be A Better Leader
 Apple              | Lead Tester            | Understand project management better | Will Be A Better Leader
 Microsoft          | Trainee Software Engineer | Understand project leadership better | Easier to lead
 Google             | Trainee Tester         | Build Test Performance               | Find Testing Easier
 Ubuntu             | Trainee Software Engineer | Understand software methods          | Speed up my learning time
(6 rows)
```

SELECT * FROM MY_AIMS WHERE ROLE='LEAD TESTER';

This query find out what aims and aim relevancies apply to a certain job:

```
     comp_name      |    role     |                my_aim                |       my_aim_rel
--------------------+-------------+--------------------------------------+-------------------------
 Apple              | Lead Tester | Learn insides and out of testing mac os | Helps Testing Macs
 Apple              | Lead Tester | Build team building skills           | Will Be A Better Leader
 Apple              | Lead Tester | Understand project management better | Will Be A Better Leader
(3 rows)
```

SELECT ROLE FROM MY_AIMS WHERE MY_AIM='BUILD TEST PERFORMANCE';

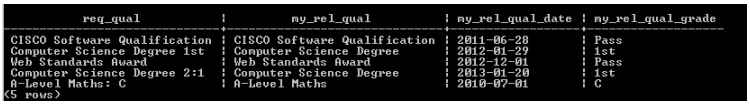
I performed a query here to look for roles which aims apply to:

```
      role
----------------
 Trainee Tester
(1 row)
```

My qualification queries

SELECT * FROM MY_QUALIFICATIONS;

This was a check to see if the my_qualifications table was populated correctly:

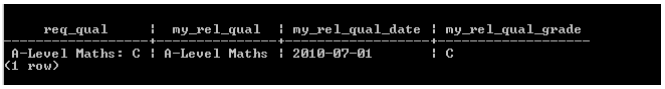


req_qual	my_rel_qual	my_rel_qual_date	my_rel_qual_grade
CISCO Software Qualification	CISCO Software Qualification	2011-06-28	Pass
Computer Science Degree 1st	Computer Science Degree	2012-01-29	1st
Web Standards Award	Web Standards Award	2012-12-01	Pass
Computer Science Degree 2:1	Computer Science Degree	2013-01-20	1st
A-Level Maths: C	A-Level Maths	2010-07-01	C

(5 rows)

SELECT * FROM MY_QUALIFICATIONS WHERE REQ_QUAL ='A-LEVEL MATHS: C';

I looked up to see what jobs require one of the applicants qualifications:

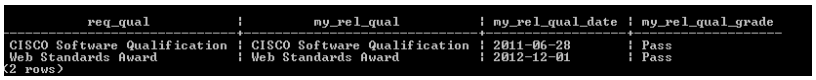


req_qual	my_rel_qual	my_rel_qual_date	my_rel_qual_grade
A-Level Maths: C	A-Level Maths	2010-07-01	C

(1 row)

SELECT * FROM MY_QUALIFICATIONS WHERE MY_REL_QUAL_GRADE='PASS';

This query was performed to check what required qualifications were held by the user that had been graded a pass:

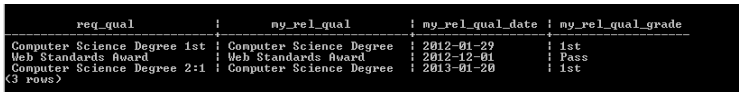


req_qual	my_rel_qual	my_rel_qual_date	my_rel_qual_grade
CISCO Software Qualification	CISCO Software Qualification	2011-06-28	Pass
Web Standards Award	Web Standards Award	2012-12-01	Pass

(2 rows)

SELECT * FROM MY_QUALIFICATIONS WHERE MY_REL_QUAL_DATE>'2011-07-01';

This checks the date type within the qualifications table to see what qualifications were gained after a certain date:



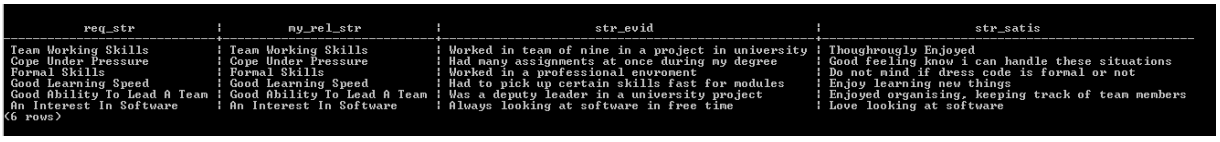
req_qual	my_rel_qual	my_rel_qual_date	my_rel_qual_grade
Computer Science Degree 1st	Computer Science Degree	2012-01-29	1st
Web Standards Award	Web Standards Award	2012-12-01	Pass
Computer Science Degree 2:1	Computer Science Degree	2013-01-20	1st

(3 rows)

My strength queries

SELECT * FROM MY_STRENGTHS;

This checks to see if the strengths table was populated correctly:



req_str	my_rel_str	str_evid	str_satis
Team Working Skills	Team Working Skills	Worked in team of nine in a project in university	Thoughrougly Enjoyed
Cope Under Pressure	Cope Under Pressure	Had many assignments at once during my degree	Good feeling know i can handle these situations
Formal Skills	Formal Skills	Worked in a professional enviroment	Do not mind if dress code is formal or not
Good Learning Speed	Good Learning Speed	Had to pick up certain skills fast for modules	Enjoy learning new things
Good Ability To Lead A Team	Good Ability To Lead A Team	Was a deputy leader in a university project	Enjoyed organising, keeping track of team members
An Interest In Software	An Interest In Software	Always looking at software in free time	Love looking at software

(6 rows)

SELECT * FROM MY_STRENGTHS WHERE REQ_STR='AN INTEREST IN SOFTWARE';

This query was used to see what the applicants strengths were matching a strength requirement:

req_str	my_rel_str	str_evid	str_satis
An Interest In Software	An Interest In Software	Always looking at software in free time	Love looking at software

(1 row)

```
SELECT STR_SATIS, MY_REL_STR FROM MY_STRENGTHS WHERE REQ_STR='GOOD ABILITY TO LEAD A TEAM';
```

In this final query, it was used to test what the satisfaction and relative strength of a specific required strength was.

```
                  str_satis                   |        my_rel_str
----------------------------------------------+--------------------------
 Enjoyed organising, keeping track of team members | Good Ability To Lead A Team
(1 row)
```

Summary

Following on from the implementation, I will now summarise the assignment, and talk about each stage in terms of how I think I performed and how I may have been able to improve if possible. To begin this summary, concerning the implementation of my postgres tables, insertion of data, queries and query results, and because reading screenshots within a PDF file is not very continent, I have provided links below to each of the necessary files which covers all the screenshots and queries covered in this assignment:

- SQL File containing CREATE statements
- SQL File containing INSERT statements
- SQL File containing QUERY statements
- TXT file containing result from SQL query

When beginning this assignment, I knew little about normalisation and implementing them on a database management system such as postgres. In order to know how to deal with the data presented to me, I began by reading a few tutorials covering the different stages of normalisation until I knew enough to try it with the data. The simplest section of this was to name the data appropriately. I thought that I named them well, keeping them short yet readable. It also made it easier to type them when implementing normalisation and SQL commands.

When it came to normalisation itself, I found it tricky in places for instance where to split off the repeating groups. In my opinion there were several options I could have chosen in terms of what the repeating groups were. This should not have been a problem though because the normalisation stage would tackle this anyway. After identifying the functional dependencies of the data set, it made it much easier to identify what tables should end up being present at the end of normalisation. Following the rules identified from research into normalisation, it made it much easier to see when something was first normal form and when something was second normal form for instance by looking at the entities containing attributes.

To implement the complete normalised tables, I decided to install the postgres application on my windows machine and found it incredibly easy to run a database locally allowing me to quickly implement and create each table and then populate the tables. Rather than run one command at a time, I found that by creating a file with the SQL extension and having all the commands there made it easier when I wanted to run the whole implementation. Whilst in the command line of postgres I found many useful commands to use such as hiding the 'NOTICE' lines when running certain commands or others like changing the layout of tables printed in the command window.

When it came to selecting what data types I needed to use for each attribute I had to ensure that the type kept a correct restraint on data inputted into the system. For instance, I ensured that where only a few discrete possible values could be entered.