COMP. 6108

Assignment 1 Part 2

George Czabania

27002246

15th October 2013

Table of Contents

Introduction 3

Logical Design 4

The Integrated Normalised Entity-Relationship Diagram 4

Physical Design 5

Relations 5

Data Volume Map 6

Data Usage Maps 7

All Students and Matching Vacancies 7

All Students Interested in One Vacancy 8

Considerations for De-normalisation 9

Partitioning 12

Advantages 12

Final Entity Relationship Diagram 13

Data Dictionary 14

Location 14

Employer 14

Vacancy 14

Student 15

StudentEmployed 16

Skill 16

StudentSkill 16

VacancySkill 17

Applicant 17

Appendix 18

Communications with client 18

Part 1 18

Part 2 18

# Introduction

In this report, I will take the logical design of the database from Part 1 of the Assignment and convert it into a physical design and implementation.

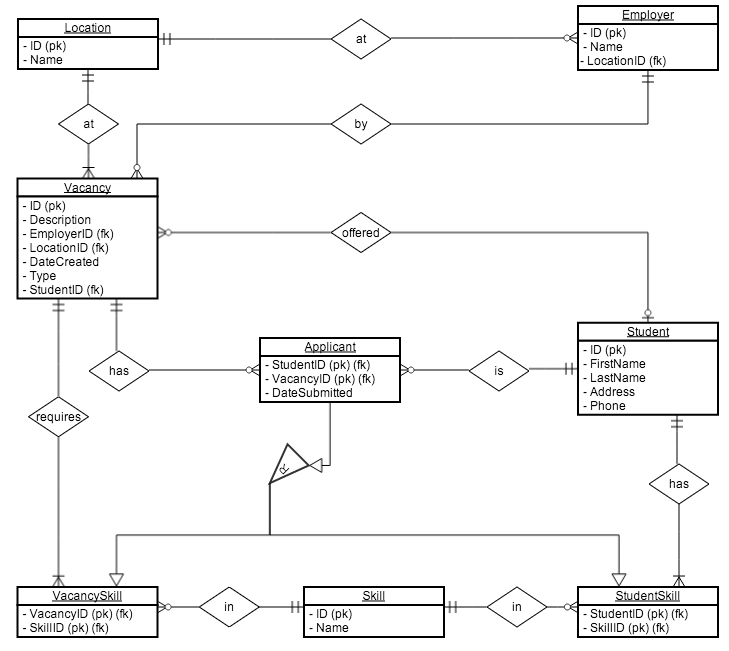
This process will include modeling the data volume and access frequency for the two most important processes as well as considering possible de-normalisation methods to improve performance.

This will result in an SQL script file that can be used to generate the tables and populate them with example data.

# Logical Design

## The Integrated Normalised Entity-Relationship Diagram

This is the integrated normalized entity-relationship diagram from Part 1 of the Assignment.



# Physical Design

## Relations

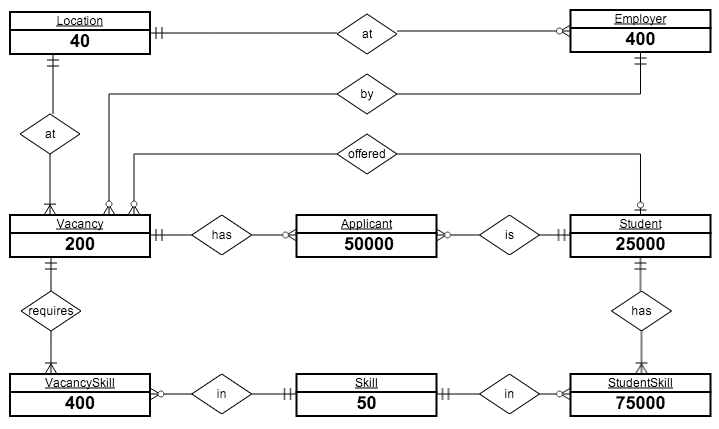
**Tables**

* Location ( ID, Name )
* Vacancy ( ID, Description, EmployerID, LocationID, DateCreated, Type, StudentID )
* Employer ( ID, Name, LocationID )
* VacancySkill ( VacancyID, SkillID )
* Skill ( ID, Name )
* StudentSkill ( StudentID, SkillID )
* Student ( ID, FirstName, LastName, Address, Phone )
* Applicant ( StudentID, VacancyID, DateSubmitted )

**Foreign Keys**

* Vacancy
  + EmployerID: ON DELETE NO ACTION
  + LocationID: ON DELETE NO ACTION
  + StudentID: ON DELETE SET NULL
* Employer
  + LocationID: ON DELETE NO ACTION
* VacancySkill
  + SkillID: ON DELETE CASCADE
  + VacancyID: ON DELETE CASCADE
* StudentSkill
  + SkillID: ON DELETE CASCADE
  + StudentID: ON DELETE CASCADE
* Applicant
  + StudentID: ON DELETE CASCADE
  + VacancyID: ON DELETE CASCADE

## Data Volume Map



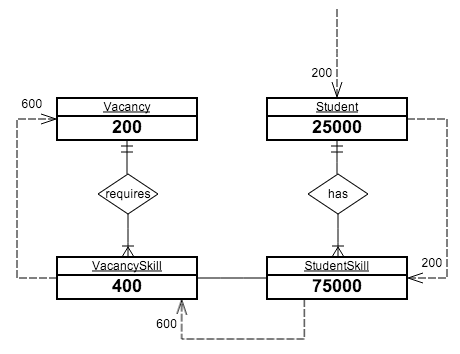
These estimated volumes were gathered from the client:

* Location: 40
* Employer: 400
* Vacancy: 200
* Student: 25000
* Skill: 50
* Applicant: (2 per Student) = 50000
* VacancySkill: (2 per Vacancy) = 400
* StudentSkill: (3 per Student) = 75000

## Data Usage Maps

### All Students and Matching Vacancies

This process is run 200 times a day.

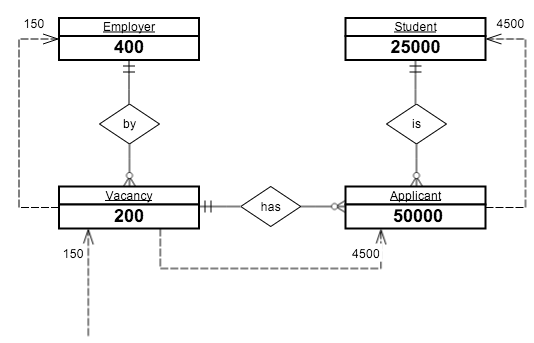


Assumptions:

* For each request for a Student, one request will need to be made to get a list of their Skills.
* On average, Students have three Skills. For each of these Skills, we will need to check it against every VacancySkill and match the SkillID attribute.
* For each VacancySkill we need to get the Vacancy, this takes a single request per VacancySkill.

### All Students Interested in One Vacancy

This process is run 150 times a day.



Assumptions:

* For each Vacancy we need to send one request to get the Employer.
* On average, a Vacancy will have 30 Applicants. For each of these Applicants, we need to send a request (30 \* 150 = 4500).
* For each Applicant we need to send one request to get the Student details of who applied.

## Considerations for De-normalisation

These considerations will be based on the two main processes of the system:

* All Students and Matching Vacancies
* All Students Interested In One Particular Vacancy

**Joining Tables to create a One-To-One relationship**

This de-normaliztion method is used to increase performance by reducing the number of joins that must be done when querying information across multiple tables.

While not very useful in this system, and example scenario would be if the majority of employers only listed very few vacancies, and there were are large number of employers.

This could possibly be useful for the All Students and Matching Vacancies process, as an inner join must be made to connect the Vacancy with the VacancySkill.

To turn this into a one-to-one relationship, we could merge VacancySkill and Vacancy together. This would duplicate a lot of data, but would possibly improve performance.

**Vacancy**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **s** | **Description** | **EmployerID** | **LocationID** | **DateCreated** | **Type** |
| 1 | Cashier | 2 | 3 | 2013-10-22 | 0 |

**Skill**

|  |  |
| --- | --- |
| **VacancyID** | **SkillID** |
| 1 | 10 |
| 1 | 20 |
| 1 | 30 |

**Vacancy + Skill**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Description** | **EmployerID** | **LocationID** | **DateCreated** | **Type** | **SkillID** |
| 1 | Cashier | 2 | 3 | 2013-10-22 | 0 | 10 |
| 1 | Cashier | 2 | 3 | 2013-10-22 | 0 | 20 |
| 1 | Cashier | 2 | 3 | 2013-10-22 | 0 | 30 |

As you can see each vacancy is duplicated for every skill it requires. This makes maintaining it a lot harder, as rows could easily get out of sync if only one instance of them is changed.

Vacancies only have two skills on average. So merging vacancy and vacancy skill together is only possibly becauce it is a one to many relationship.

Positive: less joining needs to be done in the SQL query (which is a relatively slow process)

Negative: duplication of data takes up more space, added complexity – changing a vacancy now takes longer because it needs to update all instances in the database. There is also the possibility of data-corruption if one instance of the vacancy is set to something different and doesn’t update the other instances. So one vacancy might get a student to apply for it, but not applied to the other instances of the same vacancy

**Horizontal partitioning**

This is when you distribute a tables records across multiple tables

Fox example, there are thousands of students stored in the database, but only the most recently added are regularly accessed. To improve performance, the Student table could be horizontally partitioned into sections of a 1000 students.

Students who are employed don’t need to be searched when looking for possibly students for a vacancy.

Student, Student\_Employed

This particular type of portioning is known as “key range partitioning” because the primary key is used to create the groups.

This is faster, as the DBMS has fewer students to sort through per table.

However more logic is required for the system to know which table to open to find a student.

Not going to do this as the database should be handle 25000 students assuming that most of the students already have gained employment and will be excluded from the results.

**Vertical Partitioning**

Need to store more information, as the primary key is duplicated on each sub-table.

This involves splitting up tables by distributing the columns of a table into several separate tables.

This is not very useful for this database, as all the tables have a small amount of columns that is accessed at the same time.

For example, the Student table is accessed very often to get each students first and last name. The student table also stores the phone number and address of each student, but this information is accessed rarely. To improve performance in this scenario, the student table could be vertically partitioned to create two tables:

**Student**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StudentID** | **FirstName** | **LastName** | **Phone** | **Address** |
| 2 | George | Smith | 123 | 20 Round St |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Student\_Name** | | |  | **Student\_Contact** | | |
| **StudentID** | **FirstName** | **LastName** |  | **StudentID** | **Phone** | **Address** |
| 2 | George | Smith |  | 2 | 123 | 20 Round St |

This improves performance for queries that only get the name of the student, or the phone and address. But slows down performance for queries that need both the name and phone or address.

Makes managing foreign keys harder, as you have to check referential integrity on both tables.

Not ideal use case, as phone and address data are not used in the two main processes. And moving them into a separate table doesn’t help us. It would if we accessed it separately to the name.

Also makes it harder to manage the database as the student information is split.

Not going to do this.

**Indexing**

Creating indexes for columns that are not primary keys.

We should index foreign keys, to make joining table faster.

However for both of the important processes, every attribute that is accessed is already a primary key and has a index.

Except for Vacancy.EmployerID.

We could index this field, but it wouldn’t help in either of the two processes, as we are querying the Vacancy table by the ID, which is already indexed.

### Partitioning

Splitting a table into multiple parts.

### Advantages

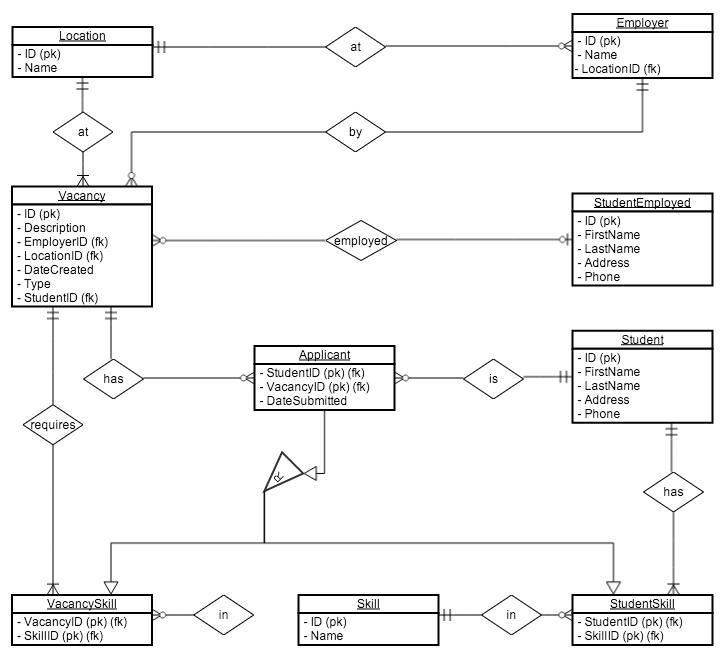
Advantages of using de-normalization.

* Harder to maintain
* Faster access times

**Disadvantages**

* Inconsistent speed access
* More complexity
* Extra space
* Slower update times

## Final Entity Relationship Diagram



The only change made was to horizontally partition the Student table and create the StudentEmployed table. This new table now holds all the students are employed at a Vacancy.

## Data Dictionary

### Location

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| ID | Location ID | int |  | 12 | ✔ | ✔ | ✔ |  |  |
| Name | Location Name | varchar | 30 | Lynmore | ✔ |  |  |  |  |

### Employer

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| ID | Employer ID | int |  | 48 | ✔ | ✔ | ✔ |  |  |
| Name | Employer Name | varchar | 30 | Wairiki | ✔ |  |  |  |  |
| LocationID | Employer location | int |  | 2 | ✔ |  |  | ✔ |  |

### Vacancy

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| ID | Vacancy ID | int |  | 15 | ✔ | ✔ | ✔ |  |  |
| Description | Vacancy Description | varchar | 30 | Lynmore | ✔ |  |  |  |  |
| EmployerID | Vacancy Employer | int |  | 32 | ✔ |  |  | ✔ |  |
| LocationID | Vacancy Location | int |  |  | ✔ |  |  | ✔ |  |
| DateCreated | Date Vacancy was created | timestamp |  | 2013-10-22 18:45:00 | ✔ |  |  |  | CURRENT TIMESTAMP |
| Type | Full time or part time | tinyint |  | 0 | ✔ |  |  |  |  |
| StudentID | The Student employed | int |  | 30 |  |  |  | ✔ |  |

### Student

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| ID | Student ID | int |  | 2732 | ✔ | ✔ | ✔ |  |  |
| FirstName | Student First Name | varchar | 30 | Jim | ✔ |  |  |  |  |
| LastName | Student Last Name | varchar | 30 | Smith | ✔ |  |  |  |  |
| Address | Student Address | varchar | 100 | 19 Devon St Rotorua | ✔ |  |  |  |  |
| Phone | Student Phone Number | varchar | 20 | 021 793 082 | ✔ |  |  |  |  |

### StudentEmployed

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| ID | Student ID | int |  | 2732 | ✔ | ✔ | ✔ |  |  |
| FirstName | Student First Name | varchar | 30 | Jim | ✔ |  |  |  |  |
| LastName | Student Last Name | varchar | 30 | Smith | ✔ |  |  |  |  |
| Address | Student Address | varchar | 100 | 19 Devon St Rotorua | ✔ |  |  |  |  |
| Phone | Student Phone Number | varchar | 20 | 021 793 082 | ✔ |  |  |  |  |

### Skill

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| ID | Skill ID | int |  | 2 | ✔ | ✔ | ✔ |  |  |
| Name | Skill Description | varchar | 20 | Word Processing | ✔ |  |  |  |  |

### StudentSkill

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| SkillID | Student Skill | int |  | 12 | ✔ | ✔ | ✔ | ✔ |  |
| StudentID | Student ID | int |  | 23 | ✔ | ✔ | ✔ | ✔ |  |

### VacancySkill

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| SkillID | Vacancy Skill | int |  | 52 | ✔ | ✔ | ✔ | ✔ |  |
| StudentID | Vacancy ID | int |  | 123 | ✔ | ✔ | ✔ | ✔ |  |

### Applicant

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Size** | **Example** | **Required** | **Indexed** | **Primary Key** | **Foreign Key** | **Default Value** |
| StudentID | Student ID | int |  | 12 | ✔ | ✔ | ✔ | ✔ |  |
| VacancyID | Vacancy ID | int |  | 32 | ✔ | ✔ | ✔ | ✔ |  |
| DateCreated | Date Application Created | datetime |  | 2013-10-22 19:42:20 | ✔ |  |  |  | CURRENT TIMESTAMP |

# Appendix

## Communications with client

### Part 1

1. How many entries do you expect to store in each of these tables?

* Locations: **40**
* Employers: **400**
* Students: **25000**
* Skills: **50**
* Applicants (for students who have applied for a vacancy): **Applicants are students (on average 2 applications per student)**
* Student Skills (for skills that students have): **On average a student has 3 skills. You may calculate this.**
* Vacancy Skills (for skills required by vacancy): **On average a vacancy requires 2 skills. You may calculate this.**

1. How often will each of these tables be accessed (per hour)?

* **Limit this to the specified processes (two). The frequency of usage of such processes is mentioned in the assignment.**

### Part 2

1. How many entries do you expect to store in each of these tables?

* Vacancies: **200**

1. On average, how many applicants are there for each vacancy?

* **30 applicants for one vacancy.**