5 Nov 2023

Assignment

Database design & development

**Vijay Kumar**

**Bachelor of Computing System**

**Unitec New Zealand**

**Shiu, Ram**

Sponsor/Owner

Aotearoa Motokā Rihi

33 Carrington Road, Mt. Albert

Auckland, 0612

123-456-7890

sram@unitec.co.nz

Prepared By

Vijay, Kumar

Chris Escandor

Jatinder Singh

Table of Contents

[Task 1 - Introduction: 1](#_Toc149026647)

[Task 2 - Data Volume Map: Looking Glass Recruitment 2](#_Toc149026648)

[Task 3.1 - Data Usage Map: All Vacancies Filled 3](#_Toc149026649)

[Task 3.2 - Data Usage Map: All Unfilled Vacancies with Candidates Interested. 4](#_Toc149026650)

[Task 4 - Business processes to physical design techniques matrix table using the business processes. 5](#_Toc149026651)

[Task 5 6](#_Toc149026652)

[1 - Enter, modify, or delete candidates. 6](#_Toc149026653)

[2 - Enter, modify or delete vacancies. 6](#_Toc149026654)

[3 - Add or remove skills or qualifications to or from a candidate. 7](#_Toc149026655)

[4 - Add or remove skills or qualifications to or from a candidate. 8](#_Toc149026656)

[5 - Apply a candidate to a vacancy. 9](#_Toc149026657)

[6 - Mark a vacancy as filled. 10](#_Toc149026658)

[7 - All filled vacancies. 11](#_Toc149026659)

[8 - All unfilled vacancies. 12](#_Toc149026660)

[9 - All unfilled vacancies with candidates interested. 13](#_Toc149026661)

[10 - Vacancies by Category. 14](#_Toc149026662)

[11 - Vacancies by Employer. 14](#_Toc149026663)

[12 - All Candidates. 15](#_Toc149026664)

# Task 1 - Introduction:

The aim of this project is to provide a robust database management system for the Looking Glass Recruitment. The company operates as a human resources agency. The current informal paper-based system frequently encounters issues when it comes to aligning a candidate's skills and qualifications with those needed for a specific job opening, as well as maintaining precise records of an employer's vacancies.

With a growing number of job seekers (referred to as candidates within the agency) availing the services of Looking Glass Recruitment, there is a need for a database application to streamline the processes of monitoring employers' vacancies and candidates' particulars. The primary objective of this new application is to establish a sturdy system that ensures employers receive accurate candidate information and candidates are directed towards suitable vacancies. It is imperative that this new application enhances the efficiency of Looking Glass Recruitment's services, leading to heightened satisfaction levels for both employers and candidates, and a substantial increase in its employer clientele.

The database must be designed and optimized according to the business requirements listed in table below:

|  |  |
| --- | --- |
| Application Requirements | |
| Form Requirements | Reports Required |
| * Enter, modify, or delete candidates. * Enter, modify, or delete vacancies. * Add or remove skills or qualifications to or from a candidate. * Add or remove skills or qualifications to or from a vacancy. * Apply a candidate to a vacancy. * Mark a vacancy as filled. | * All filled vacancies. * All unfilled vacancies. * All unfilled vacancies with candidates interested. * Vacancies by category * Vacancies by employer * All candidates |

The two most important processes are:

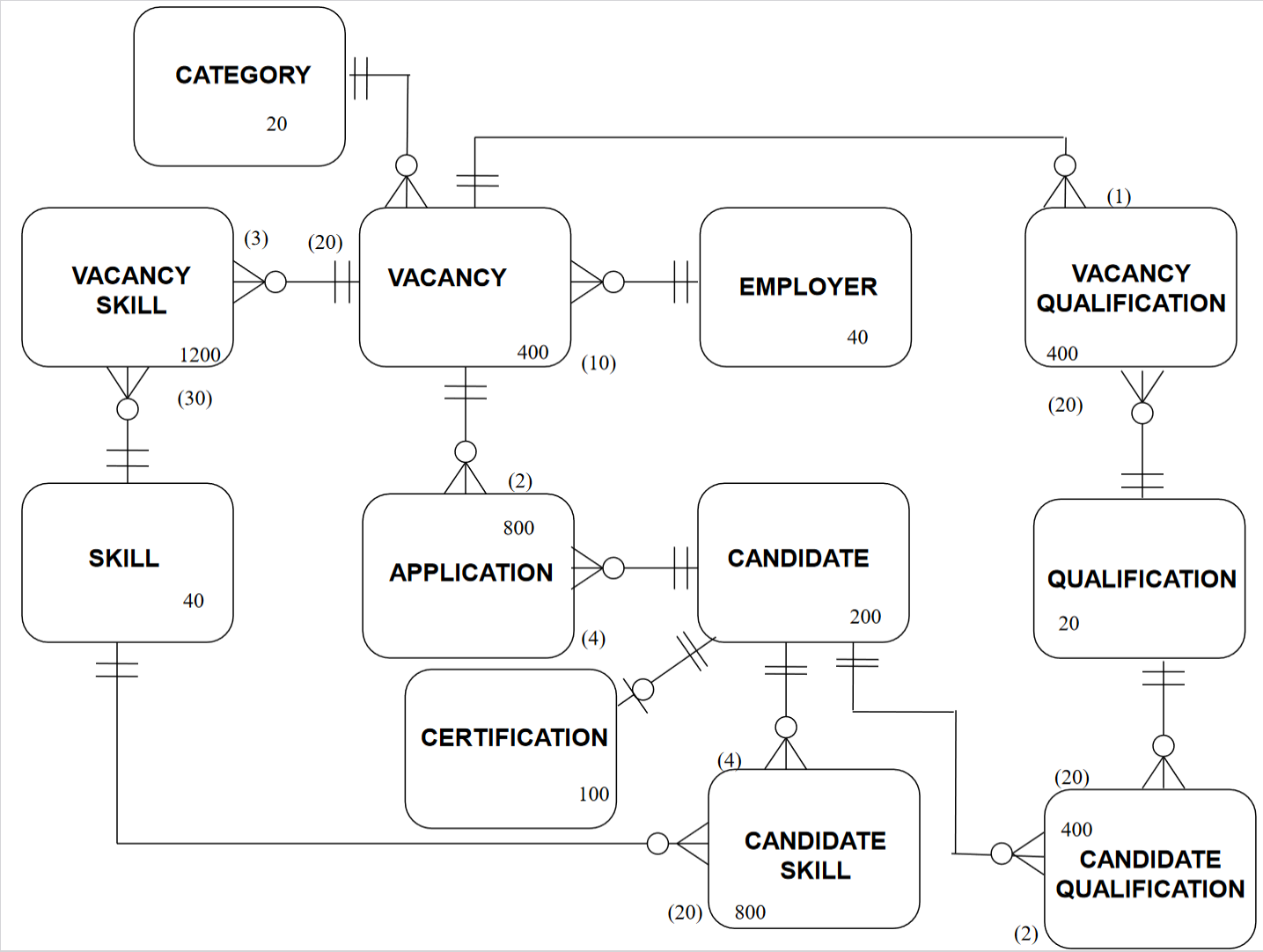
* All filled vacancies

On average 5 times a day

* All unfilled vacancies with candidates interested.

On average 4 times a day

# Task 2 - Data Volume Map: Looking Glass Recruitment



# Task 3.1 - Data Usage Map: All Vacancies Filled

A diagram of a job

Description automatically generated

# Task 3.2 - Data Usage Map: All Unfilled Vacancies with Candidates Interested.

A diagram of a job application

Description automatically generated

# 

# Task 4 - Business processes to physical design techniques matrix table using the business processes.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data Entity types  Business functions | One-one Relationship | One-many Relationship(reference) | Associative Entity | Duplication | Horizontal Partitioning | Vertical Partitioning | Indexing |
| 1.Enter, modify, or delete candidates. | Yes | No | No | No | No | Yes | Yes |
| 2.Enter, modify, or delete vacancies. | No | Yes | No | Yes | No | Yes | Yes |
| 3.Add or remove skills or qualifications to or from a candidate. | No | No | No | Yes | No | Yes | Yes |
| 4.Add or remove skills or qualifications to or from a vacancy. | No | No | Yes | Yes | No | Yes | Yes |
| 5.Apply a candidate to a vacancy. | No | No | Yes | Yes | Yes | Yes | Yes |
| 6.Mark a vacancy as filled. | No | Yes | Yes | Yes | Yes | Yes | Yes |
| 7.All filled vacancies. | No | Yes | Yes | Yes | Yes | Yes | No |
| 8.All unfilled vacancies. | No | Yes | Yes | Yes | Yes | Yes | No |
| 9.All unfilled vacancies with candidates interested. | No | No | Yes | Yes | Yes | Yes | No |
| 10.Vacancies by Category. | Yes | No | No | Yes | No | Yes | Yes |
| 11.Vacancies by Employer. | No | No | No | Yes | No | Yes | Yes |
| 12.All Candidates. | Yes | No | Yes | Yes | No | Yes | Yes |

# Task 5

## 1 - Enter, modify, or delete candidates.

1. List candidate details (candidate ID, last name, first name, street address, suburb, phone number, and status) by first name within last name.
2. List certification details (notes and certification date) for a selected candidate.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT \* FROM CANDIDATE  ORDER BY LASTNAME, FIRSTNAME. |
| b) | SELECT CERTIFICATIONDATE, NOTES FROM CERTIFICATION  WHERE CANDIDATEID =4; |
| Indexing | a) | Candidate Last name and First name are frequently used among four business processes (1,3,5,12) to sort the dataset. Indexing on First name and last name can improve read performance. The SQL to this indexing is below:  CREATE INDEX IDX\_FIRSTNAME\_LASTNAME ON CANDIDATE (LastName, FirstName); |
| Disadvantages | a) | Indexing comes with the cost of increased storage and maintenance overhead. Whenever data is inserted, updated, or deleted in “Candidate” table (business process 1), the index needs to be maintained, which can slightly slow down these operations. Business process 1 is not important process so the indexing first name last name is considered. |
|  |  |  |
|  |  |  |

## 2 - Enter, modify or delete vacancies.

1. List vacancy details (vacancy ID, description, type, status, hourly rate, employer name, and category name) by description.
2. List employers (employer ID, employer name) by employer name.
3. List categories (category ID, category name) by category name.

|  |  |  |
| --- | --- | --- |
| Current SQL |  | SELECT V. VacancyID, V.Description,V.Type, V.Status,V.HourlyRate,  E.EMPLOYERNAME, C.CATEGORYNAME  FROM  VACANCY V INNER JOIN  EMPLOYER E ON V.EMPLOYERID = E.EMPLOYERID  INNER JOIN  CATEGORY C ON C.CATEGORYID = V.CATEGORYID  ORDER BY DESCRIPTION; |
| b) | SELECT EMPLOYERID, EMPLOYERNAME FROM EMPLOYER  ORDER BY EMPLOYERNAME; |
| c) | SELECT \* FROM CATEGORY  ORDER BY CATEGORYNAME; |
| Indexing | a) | In query “b” the data is sorted by employer name, in this case employer name can be indexed to make query to run faster.    However, as per the description Employer number is likely to increase this makes it volatile, hence Indexing for employer name won’t be carried out. |
| b) | In query “c” the data is sorted by category name, in this case category name can be indexed to make query to run faster.  Categories unlikely to change often, this concludes to use indexing on category name. SQL below:  CREATE INDEX IDX\_CATEGORYNAME ON CATEGORY (CategoryName); |
| Disadvantages | | It consumes extra storage when indexing categoryname.  Overhead on Insert, Update, and Delete Operations for category table. |
| Revised SQL | | No change in SQL |

## 3 - Add or remove skills or qualifications to or from a candidate.

1. List candidate details (candidate ID, last name, and first name) by first name within last name.
2. List skills (skill id and description) by description.
3. List skills (skill id, description, and years) by description for a selected candidate.
4. List qualifications (qualification id, level and description) by description.
5. List qualifications (qualification id, level and description) by description for a selected candidate.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT CANDIDATEID, LASTNAME, FIRSTNAME FROM CANDIDATE  ORDER BY LASTNAME, FIRSTNAME; |
| b) | SELECT \* FROM SKILL  ORDER BY DESCRIPTION; |
| c) | SELECT CS.SKILLID, S.DESCRIPTION, CS.YEARS FROM CANDIDATESKILL CS  INNER JOIN SKILL S ON CS.SKILLID=S.SKILLID  WHERE CANDIDATEID = 2  ORDER BY DESCRIPTION; |
| d) | SELECT \* FROM QUALIFICATION  ORDER BY DESCRIPTION; |
| e) | SELECT CQ.QUALIFICATIONID, Q.QUAL\_LEVEL, Q.DESCRIPTION FROM CANDIDATEQUALIFICATION CQ  INNER JOIN QUALIFICATION Q ON Q.QUALIFICATIONID = CQ.QUALIFICATIONID  WHERE CQ.CANDIDATEID = 4; |
| Vertical Partitioning |  |  |
|  |  |

## 4 - Add or remove skills or qualifications to or from a candidate.

1. List vacancy details (vacancy ID, description, and employer name) for unfilled vacancies only by description.
2. List skills (skill id and description) by description.
3. List skills (skill id, description, and years) by description for a selected vacancy.
4. List qualifications (qualification id, level and description) by description.
5. List qualifications (qualification id, level and description) by description for a selected vacancy.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  V.VACANCYID,  V.DESCRIPTION,  E.EMPLOYERNAME  FROM  VACANCY V  INNER JOIN EMPLOYER E ON E.EMPLOYERID = V.EMPLOYERID  WHERE STATUS = 'Unfilled'  ORDER BY DESCRIPTION; |
| b) | SELECT \* FROM SKILL  ORDER BY DESCRIPTION; |
| c) | SELECT  VS.SKILLID,  S.DESCRIPTION,  VS.YEARS  FROM  VACANCYSKILL VS  INNER JOIN SKILL S ON S.SKILLID = VS.SKILLID  WHERE VS.VACANCYID = 16  ORDER BY S.DESCRIPTION; |
| d) | SELECT \* FROM qualification  ORDER BY DESCRIPTION; |
| e) | SELECT  VQ.QUALIFICATIONID,  Q.QUAL\_LEVEL,  Q.DESCRIPTION  FROM  VACANCYQUALIFICATION VQ  INNER JOIN QUALIFICATION Q ON Q.QUALIFICATIONID = VQ.QUALIFICATIONID  WHERE VQ.VACANCYID = 2  ORDER BY DESCRIPTION; |
| Indexing | a) | Qualification and Skill tables records are unlikely to change, frequently used, and need to be ordered by description, this makes it appropriate to do Indexing on both tables for description table. SQL for indexing below:   * CREATE INDEX SKILL\_DESCRIPTION\_ON SKILL (DESCRIPTION); * CREATE INDEX QUALIFICATION\_DESCRIPTION\_ON QUALIFICATION (DESCRIPTION); |

## 5 - Apply a candidate to a vacancy.

1. List vacancy details (vacancy ID, description, type, and employer name) for unfilled vacancies only by description.
2. List applications (candidate’s last name and first name) for a selected vacancy by candidate’s last name
3. List candidate details (candidate ID, last name, and first name) by first name within last name.
4. List skills (skill id, description, and years) by description for a selected vacancy.
5. List qualifications (qualification id, level and description) by description for a selected vacancy.
6. List skills (skill id, description, and years) by description for a selected candidate.
7. List qualifications (qualification id, level and description) by description for a selected candidate.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  V.VACANCYID,  V.DESCRIPTION,  E.EMPLOYERNAME  FROM  VACANCY V  INNER JOIN EMPLOYER E ON E.EMPLOYERID = V.EMPLOYERID  WHERE STATUS = 'Unfilled'  ORDER BY DESCRIPTION; |
| b) | SELECT  C.LASTNAME,  C.FIRSTNAME  FROM CANDIDATE C  INNER JOIN APPLICATION A ON C.CANDIDATEID = A.CANDIDATEID  WHERE A.VACANCYID = 8  ORDER BY C.LASTNAME; |
| c) | SELECT CANDIDATEID, LASTNAME, FIRSTNAME FROM CANDIDATE  ORDER BY LASTNAME, FIRSTNAME; |
| d) | SELECT  VS.SKILLID,  S.DESCRIPTION,  VS.YEARS  FROM  VACANCYSKILL VS  INNER JOIN SKILL S ON S.SKILLID = VS.SKILLID  WHERE VS.VACANCYID = 16  ORDER BY S.DESCRIPTION; |
| e) | SELECT  VQ.QUALIFICATIONID,  Q.QUAL\_LEVEL,  Q.DESCRIPTION  FROM  VACANCYQUALIFICATION VQ  INNER JOIN QUALIFICATION Q ON Q.QUALIFICATIONID = VQ.QUALIFICATIONID  WHERE VQ.VACANCYID = 2  ORDER BY DESCRIPTION; |
| f) | SELECT CS.SKILLID, S.DESCRIPTION, CS.YEARS FROM CANDIDATESKILL CS  INNER JOIN SKILL S ON CS.SKILLID=S.SKILLID  WHERE CANDIDATEID = 2  ORDER BY DESCRIPTION; |
| g) | SELECT CQ.QUALIFICATIONID, Q.QUAL\_LEVEL, Q.DESCRIPTION FROM CANDIDATEQUALIFICATION CQ  INNER JOIN QUALIFICATION Q ON Q.QUALIFICATIONID = CQ.QUALIFICATIONID  WHERE CQ.CANDIDATEID = 4; |
| Duplication |  | There is opportunity exist to duplicate the data |
|  |  |  |

## 6 - Mark a vacancy as filled.

1. List vacancy details (vacancy ID, description, type, hourly rate and employer name) for unfilled vacancies only by description.
2. List applications (status, candidate’s last name and first name) for a selected vacancy by candidate’s last name

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  V.VACANCYID,  V.DESCRIPTION,  V.TYPE,  V.STATUS,  V.HOURLYRATE,  E.EMPLOYERNAME  FROM  VACANCY V  INNER JOIN EMPLOYER E ON E.EMPLOYERID = V.EMPLOYERID  WHERE STATUS = 'Unfilled'  ORDER BY DESCRIPTION; |
| b) | SELECT  C.LASTNAME,  C.FIRSTNAME  FROM CANDIDATE C  INNER JOIN APPLICATION A ON C.CANDIDATEID = A.CANDIDATEID  WHERE A.VACANCYID = 8  ORDER BY C.LASTNAME; |
| De-Normalization opportunity | a) | There is De-normalization opportunity exist, Application is associative entity and has status as a non-key attribute, Application is child to Candidate and Vacancy entity. After de-normalization Application entity can be combined with Candidate table. This will enhance the performance of this business process above (b) as there will be no join when running the query.  Considering this is not an important business process and dataset will run 1200 times a day. |
| Disadvantages | a) |  |
|  |  |  |
|  |  |  |

## 7 - All filled vacancies.

1. For each filled vacancy list the vacancy id, description, type, status, category name, employer name, employer street address, employer suburb, descriptions of skills, and descriptions of qualifications.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  V.VACANCYID,  V.Description,  V.TYPE,  V.STATUS,  C.CATEGORYNAME,  E.EMPLOYERNAME,  E.STREETADDRESS,  E.SUBURB,  S.Description AS SKILL\_QUALIFICATION  FROM VACANCY V  INNER JOIN VACANCYSKILL VS ON V.VACANCYID = VS.VACANCYID  INNER JOIN SKILL S ON VS.SKILLID = S.SKILLID  INNER JOIN EMPLOYER E ON E.EMPLOYERID = V.EMPLOYERID  INNER JOIN CATEGORY C ON C.CATEGORYID = V.CATEGORYID  WHERE V.STATUS = 'Filled'  UNION  SELECT  V.VACANCYID,  V.Description,  V.TYPE,  V.STATUS,  C.CATEGORYNAME,  E.EMPLOYERNAME,  E.STREETADDRESS,  E.SUBURB,  Q.Description AS SKILL\_QUALIFICATION  FROM VACANCY V  INNER JOIN VACANCYQUALIFICATION VQ ON V.VACANCYID = VQ.VACANCYID  INNER JOIN QUALIFICATION Q ON VQ.QUALIFICATIONID = Q.QUALIFICATIONID  INNER JOIN EMPLOYER E ON E.EMPLOYERID = V.EMPLOYERID  INNER JOIN CATEGORY C ON C.CATEGORYID = V.CATEGORYID  WHERE V.STATUS = 'Filled' |
| Duplication | a) | There is an opportunity to duplicate “CATEGORYNAME” attribute into vacancy table. New column will be added into vacancy table with the name “CATEGORYNAME”.  This would eliminate the join requirements to retrieve category name form category table, thus enhancing the performance of this query as well queries used in other business processes 2,8 and 10. |
| Disadvantages | | Since “CATEGROYNAME” will exist in two tables (Categories and Vacancy) this leads extra consumption in disk space and data redundancy.  Maintenance will increase, updating and deleting data needs to happen in both tables. |
|  |  |  |
|  |  |  |
|  |  |  |

## 8 - All unfilled vacancies.

1. For each unfilled vacancy list the vacancy id, description, type, status, category name, employer name, employer street address, employer suburb, descriptions of skills, and descriptions of qualifications.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  V.VACANCYID,  V.Description,  V.TYPE,  V.STATUS,  C.CATEGORYNAME,  E.EMPLOYERNAME,  E.STREETADDRESS,  E.SUBURB,  S.Description AS SKILL\_QUALIFICATION  FROM VACANCY V  INNER JOIN VACANCYSKILL VS ON V.VACANCYID = VS.VACANCYID  INNER JOIN SKILL S ON VS.SKILLID = S.SKILLID  INNER JOIN EMPLOYER E ON E.EMPLOYERID = V.EMPLOYERID  INNER JOIN CATEGORY C ON C.CATEGORYID = V.CATEGORYID  WHERE V.STATUS = 'Unfilled'  UNION  SELECT  V.VACANCYID,  V.Description,  V.TYPE,  V.STATUS,  C.CATEGORYNAME,  E.EMPLOYERNAME,  E.STREETADDRESS,  E.SUBURB,  Q.Description AS SKILL\_QUALIFICATION  FROM VACANCY V  INNER JOIN VACANCYQUALIFICATION VQ ON V.VACANCYID = VQ.VACANCYID  INNER JOIN QUALIFICATION Q ON VQ.QUALIFICATIONID = Q.QUALIFICATIONID  INNER JOIN EMPLOYER E ON E.EMPLOYERID = V.EMPLOYERID  INNER JOIN CATEGORY C ON C.CATEGORYID = V.CATEGORYID  WHERE V.STATUS = 'Unfilled' |
| Horizontal Partitioning | a) | Vacancy table “Status” attribute is either “filled” or “unfilled”. When querying, where clause is used. In this case it is reasonable to perform Horizontal partitioning on Vacancy table. This will make the query run faster for both important business processes (7,9) and other business processes as well (5,6,8). The vacancy table will split up horizontally, VACANCY\_FILLED and VACANCY\_UNFILLED, based on the value of status field.  This will make business process faster as the data is already filtered. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 9 - All unfilled vacancies with candidates interested.

1. For each unfilled vacancy list the vacancy id, description, type, status, employer name, descriptions and years of skills, descriptions of qualifications and full names of the candidates.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  V.VACANCYID,  V.DESCRIPTION,  V.TYPE,  V.STATUS,  E.EMPLOYERNAME,  S.DESCRIPTION AS SKILL\_DESCRIPTION,  VS.YEARS,  Q.DESCRIPTION AS QUALIFICATION\_DESCRIPTION,  C.LASTNAME,  C.FIRSTNAME  FROM VACANCY V  INNER JOIN VACANCYSKILL VS ON VS.VACANCYID = V.VACANCYID  INNER JOIN SKILL S ON S.SKILLID = VS.SKILLID  INNER JOIN VACANCYQUALIFICATION VQ ON VQ.VACANCYID = V.VACANCYID  INNER JOIN QUALIFICATION Q ON VQ.QUALIFICATIONID = Q.QUALIFICATIONID  INNER JOIN EMPLOYER E ON V.EMPLOYERID = E.EMPLOYERID  INNER JOIN APPLICATION A ON A.VACANCYID = V.VACANCYID  INNER JOIN CANDIDATE C ON A.CANDIDATEID = C.CANDIDATEID  WHERE V.STATUS = 'Unfilled' |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 10 - Vacancies by Category.

1. For each vacancy list the vacancy id, description, hourly rate, and category name. Sort by category name.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  V.VACANCYID,  V.DESCRIPTION,  V.HOURLYRATE,  C.CATEGORYNAME  FROM VACANCY V  INNER JOIN CATEGORY C ON C.CATEGORYID = V.CATEGORYID  ORDER BY C.CATEGORYNAME |
| Denormalization opportunity | a) | One-to-many (reference data): Since category table is reference data to Vacancy table, there is opportunity to de-normalize the relationship between vacancy and category table. This removes extra join required to perform query for this business process. Category name will be combined with the Vacancy table. CategoryID attribute will be kept in the Vacancy table after De-normalization to satisfy the requirement of 2nd business process.  However, this de-normalization won’t be carried out due to its disadvantages, impact on important business process 7 and this business process is not an important business process. |
| Disadvantages | | * Can lead to redundant data storage. * There is a risk of inconsistencies if the category details change. |
|  |  |  |
|  |  |  |
|  |  |  |

## 11 - Vacancies by Employer.

1. For each vacancy list the vacancy id, type, status, and employer name. Sort by employer name.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  V.VACANCYID,  V.TYPE,  V.STATUS,  E.EMPLOYERNAME  FROM VACANCY V  INNER JOIN EMPLOYER E ON V.EMPLOYERID = E.EMPLOYERID  ORDER BY EMPLOYERNAME |
| Duplication | a) | There is opportunity to duplicate the “EMPLOYERNAME” into vacancy and enhance the performance of the above query.  Considering this is not as important business process, duplication won’t be carried out. |
| Revised SQL |  | No change. |
|  |  |  |
|  |  |  |
|  |  |  |

## 12 - All Candidates.

1. For each candidate list the candidate id, last name, first name, street address, suburb, phone number, status, and certification date (if applicable). Sort by first name within last name.

|  |  |  |
| --- | --- | --- |
| Current SQL | a) | SELECT  C.CANDIDATEID,  C.LASTNAME,  C.FIRSTNAME,  C.STREETADDRESS,  C.SUBURB,  C.PHONENUMBER,  C.STATUS,  CD.CERTIFICATIONDATE  FROM CANDIDATE C  LEFT JOIN CERTIFICATION CD ON C.CANDIDATEID = CD.CANDIDATEID  ORDER BY C.LASTNAME, C.FIRSTNAME |
| De-normalization opportunity | a) | There is one to one relationship between Candidate and Certification table due to this de-normalization opportunity exists. After de-normalizing, Certification table attributes will be added in the Candidate table. This will eliminate the join when querying this business process and the query will perform faster. For All Candidate’s business process only need to query on table instead two.  I decided not to carry out de-normalization opportunity described above because this is not the important business process and this will potentially slow down the other important business process (9) by increasing the size of the Candidate table vertically. |
| Revised SQL |  | No Change. |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Indexing | a) | Status column is quite often used and is part of the important of business processes (7,9), it is appropriate to index column status of vacancy table. SQL to Index status column below:   * CREATE INDEX VACANCY\_STATUS\_IX ON VACANCY(Status);   Disadvantage: If the ‘Status’ column is frequently updated (for example, if vacancies change from 'Unfilled' to 'Filled' often), the index may need to be updated frequently as well. This can lead to increased overhead during write operations, which means this business process (6) may have slight effect on performance from indexing status column. |