|  |  |
| --- | --- |
| **Course Title** | Database Systems 1 |
| **Course Number** | CPS 510 |
| **Semester/Year** | F2021 |
| **Instructor** | Dr. Abdolreza Abhari |

**Application Name: Online Job Bank System.**

|  |  |
| --- | --- |
| **Assignment Number** | 10 |
| **Group** | 11 |
| **Assignment Title** | Final Project - Database Management System |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Full Name** | **Student Number** | **Group #** | **Section Number** | **Initial** |
| **Tusaif Azmat** | **500660278** | **11** | **04** | **T.A** |
| **Ankit Dheedsa** | **500975118** | **A.D** |
| **Mahdi Alam** | **500969935** | **M.A** |

**Table of Contents**

Application Description 3

Entity-Relationship Diagram 4

Schema Design 5

Design Views and Queries 10

UNIX Shell Implementation 21

Functional Dependencies and Database Normalization 24

Normalization 3NF 27

Normalization BCNF 29

GUI Python 35

Simple and Advanced Database Queries (with Relational Algebra) 39

Conclusion 42

**Application Description:**

This Online Job Bank application will work the same way as any other online job bank where job seekers and employers will use. We want to create an application similar to **Linkedin** and **Indeed** etc. These online job banks have been soaring in popularity and are crucial as they connect people all around the world on one platform.

We have selected this project because it will allow us to have hands-on experience with SQL database systems. We believe that we could use the experience and knowledge gained in this course to fully utilize it during the development process for this group project.

The main objective is to create a platform where users (Both Job Seekers and Employers) gain access to an online job bank database. Where they use as a job seeker will look through the database for jobs and be able to apply online using the provided services. Also an employer will gain access to the database and post new job postings and delete old postings. It will also have system administrator access along with all the company employees.

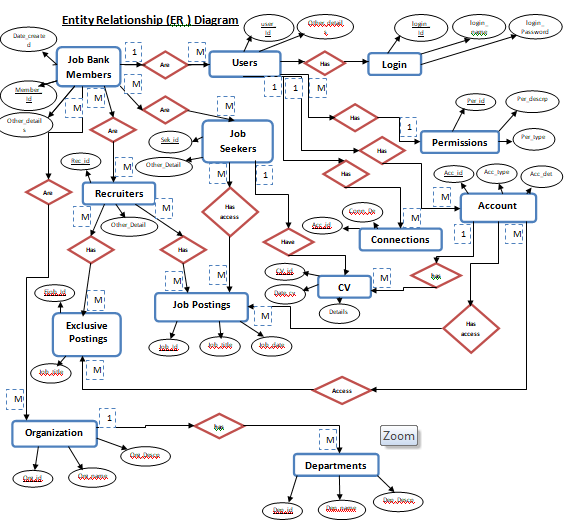
Our online Job Bank application will provide users with various job opportunities that can be accessed with just a click of a button. We will be taking inspiration for our application from other established services such as Linkedin and Indeed, we intend to incorporate networking components from Linkedin and job filtering components from Indeed. Our application will make possible the interactions between potential job candidates and hiring managers and provide an easy-to-use interface where users can easily identify and apply for jobs that are tailored to their needs.

In our DBMS, users can also opt for a subscription based premium service in which they can get further detailed information about jobs and a direct connection to interviewers. Large and small companies also reap the benefits of these subscription services as they could pool these applicants in a different bracket and hire them based on company needs. This application will have the ability to hold specific types of information about jobs that will make it easier for the user to follow.

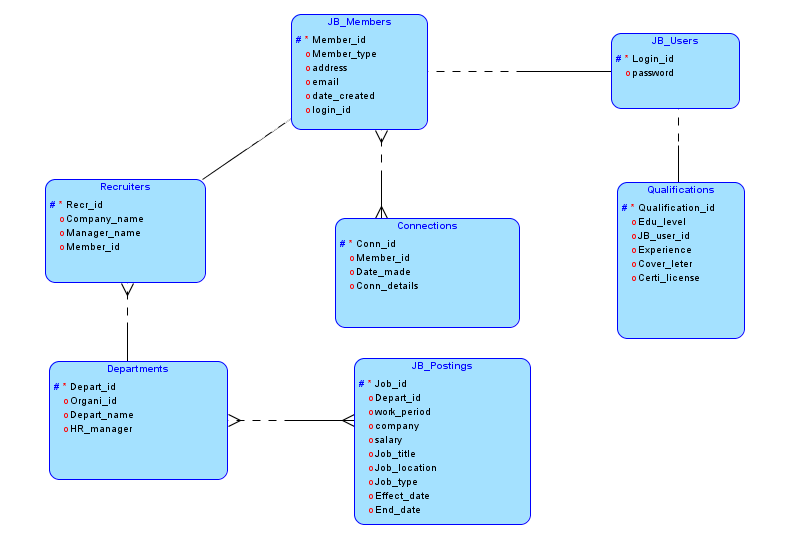
Some of the basic functions that our system will be capable of doing are outlined below:

|  |  |
| --- | --- |
| **Function** | **Description** |
| User Login | This function will allow users to log in after authentication. After log in they could access the services provided. |
| Search Jobs | This is how users will search for jobs based on the search criteria. Users will search jobs based on employers and current jobs etc. |
| Job posting | This is where users will post jobs and remove existing postings. This option will be provided to the companies/employers. |
| Job Application Status | This is where users can see what jobs they applied to, the status of their application, and where they can accept job offers. |
| Dashboard | This is essentially a home page geared for the user. They can see news related to their job offers, see surveys and reports, see their Job Application Status, Job postings geared to them, etc. |
| Notifications Log | View important updates in a list. One can see the contents of the update, the timestamp of the update, and occasionally a button to do certain actions related to the notification. |
|  |  |

**Initial ER\_Diagram:**



#### Finalized ER-diagram after BCNF Normalization



**Schema Design:**

**Source Code:**

DROP TABLE JB\_Users CASCADE CONSTRAINTS;

DROP TABLE JB\_Members CASCADE CONSTRAINTS;

DROP TABLE Recruiters CASCADE CONSTRAINTS;

DROP TABLE JB\_Postings;

DROP TABLE Qualifications;

DROP TABLE Memberships;

DROP TABLE Connections;

DROP TABLE HR\_Department;

CREATE TABLE JB\_Users (

login\_id VARCHAR2(25 CHAR),

user\_password VARCHAR2(25 CHAR),

CONSTRAINT user\_login\_id\_pk PRIMARY KEY(login\_id)

);

CREATE TABLE JB\_Members (

member\_id VARCHAR2(25 CHAR),

member\_type VARCHAR2(25 CHAR),

address VARCHAR2(25 CHAR),

org\_id VARCHAR2(25 CHAR),

email VARCHAR2(25 CHAR),

date\_Created VARCHAR2(25 CHAR),

member\_name VARCHAR2 (25 CHAR),

CONSTRAINT member\_pk PRIMARY KEY(member\_id),

CONSTRAINT JBUser\_fk FOREIGN KEY (member\_name) REFERENCES JB\_Users(login\_id) -- Foreign Key

);

CREATE TABLE Recruiters (

recr\_id VARCHAR2(25 CHAR),

member\_id VARCHAR2(25 CHAR),

job\_id VARCHAR2(25 CHAR),

date\_created VARCHAR2(25 CHAR),

email VARCHAR2(25 CHAR),

phone VARCHAR2(12 CHAR) DEFAULT '(000)-000-0000',

address VARCHAR2(25 CHAR),

CONSTRAINT Recruiters\_pk PRIMARY KEY(recr\_id),

FOREIGN KEY (member\_id) REFERENCES JB\_Members (member\_id)

);

CREATE TABLE JB\_Postings (

job\_id VARCHAR2 (20 CHAR),

work\_period NUMBER,

company VARCHAR2(25 CHAR),

salary NUMBER,

job\_title VARCHAR2(25 CHAR),

job\_location VARCHAR2(25 CHAR),

job\_type VARCHAR2(25 CHAR),

effective\_date VARCHAR2(25 CHAR),

end\_date VARCHAR2(25 CHAR),

CONSTRAINT jobs\_pk PRIMARY KEY (job\_id),--primary key

FOREIGN KEY (company) REFERENCES recruiters (recr\_id) --Foreign Key

);

CREATE TABLE Qualifications(

qualification\_id VARCHAR2(10 CHAR),

user\_degree VARCHAR2(50 CHAR) DEFAULT 'Undergraduate',

JB\_User\_id VARCHAR2(25 CHAR) NOT NULL,

years\_experience NUMBER CHECK (years\_experience BETWEEN 0 AND 30),

coverLetter VARCHAR2(1000 CHAR),

license VARCHAR2(25 CHAR),

CONSTRAINT qualification\_pk PRIMARY KEY (qualification\_id),--Primary ID

FOREIGN KEY (JB\_user\_id) REFERENCES JB\_Users (login\_id) --Foreign Key

);

CREATE TABLE Memberships (

permission\_id VARCHAR2(10 CHAR),

user\_id VARCHAR2(25 CHAR),

Type VARCHAR2(25 CHAR),

Description VARCHAR2(25 CHAR),

Start\_date VARCHAR2(25 CHAR),

End\_date VARCHAR2(25 CHAR),

CONSTRAINT permission\_pk PRIMARY KEY(permission\_id),

FOREIGN KEY (user\_id) REFERENCES JB\_Users (login\_id)

);

CREATE TABLE Connections (

conn\_id VARCHAR2(12 CHAR),

user\_id VARCHAR2(25 CHAR),

date\_made VARCHAR2(25 CHAR),

details VARCHAR2(150 CHAR),

CONSTRAINT conn\_pk PRIMARY KEY(conn\_id),

FOREIGN KEY (user\_id) REFERENCES JB\_Users (login\_id)

);

CREATE TABLE HR\_Department (

depart\_id VARCHAR2(25 CHAR),

organization\_id VARCHAR2(25 CHAR),

depart\_name VARCHAR2(50 CHAR),

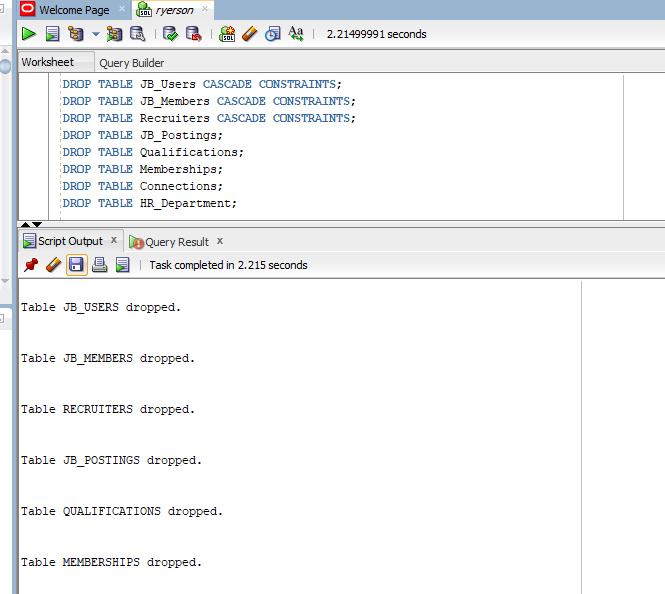
hr\_manager VARCHAR2(50 CHAR),

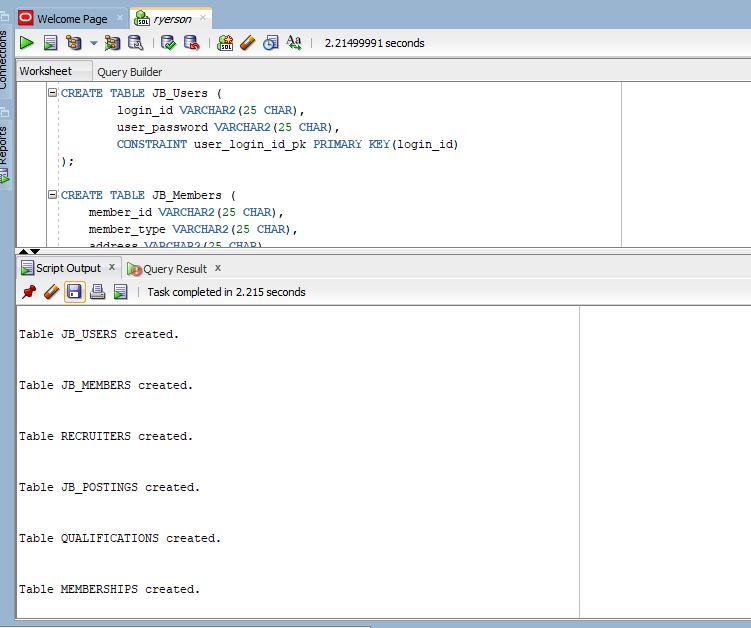
no\_employees NUMBER,

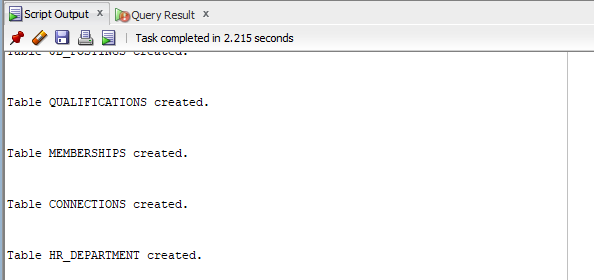
CONSTRAINT depart\_pk PRIMARY KEY(depart\_id),

FOREIGN KEY (organization\_id) REFERENCES Recruiters (recr\_id)

);







**Views and Queries**

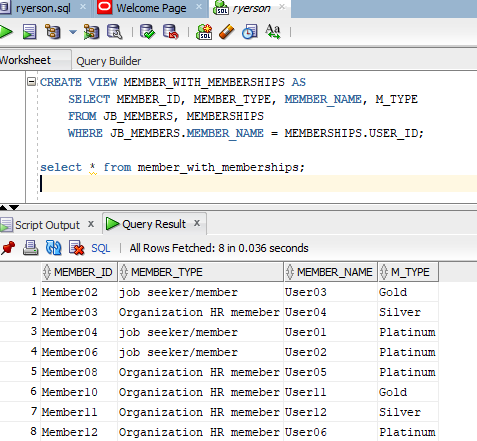
-- Table Views

CREATE VIEW MEMBER\_WITH\_MEMBERSHIPS AS

SELECT MEMBER\_ID, MEMBER\_TYPE, MEMBER\_NAME, M\_TYPE

FROM JB\_MEMBERS, MEMBERSHIPS

WHERE JB\_MEMBERS.MEMBER\_NAME = MEMBERSHIPS.USER\_ID;

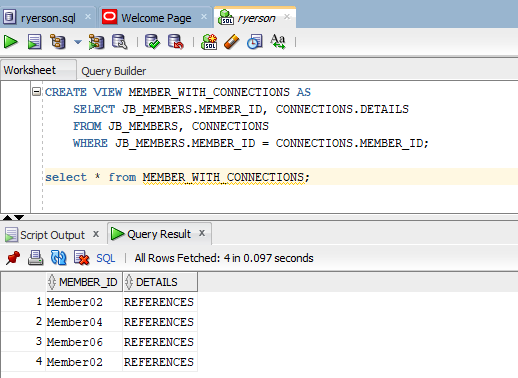
select \* from member\_with\_memberships;

CREATE VIEW MEMBER\_WITH\_CONNECTIONS AS

SELECT JB\_MEMBERS.MEMBER\_ID, CONNECTIONS.DETAILS

FROM JB\_MEMBERS, CONNECTIONS

WHERE JB\_MEMBERS.MEMBER\_ID = CONNECTIONS.MEMBER\_ID;

select \* from MEMBER\_WITH\_CONNECTIONS;

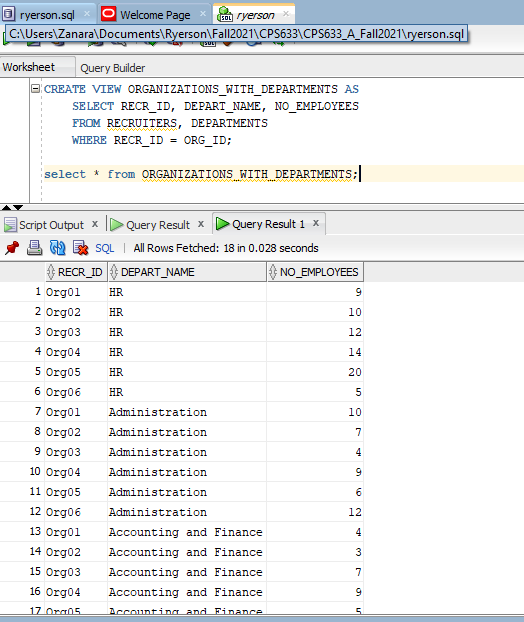
CREATE VIEW ORGANIZATIONS\_WITH\_DEPARTMENTS AS

SELECT RECR\_ID, DEPART\_NAME, NO\_EMPLOYEES

FROM RECRUITERS, DEPARTMENTS

WHERE RECR\_ID = ORG\_ID;

select \* from ORGANIZATIONS\_WITH\_DEPARTMENTS;

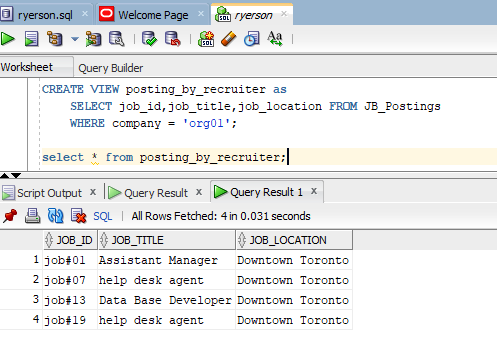


CREATE VIEW posting\_by\_recruiter as

SELECT job\_id,job\_title,job\_location FROM JB\_Postings

WHERE company = 'org01';

select \* from posting\_by\_recruiter;



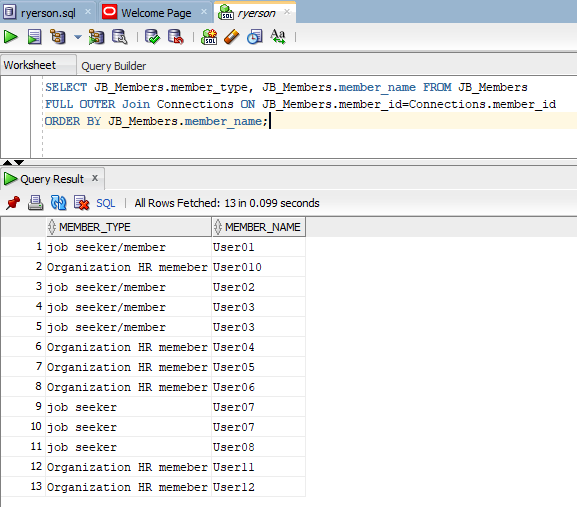
**-- advanced Queries**

-- Join Queries

SELECT JB\_Members.member\_type, JB\_Members.member\_name FROM JB\_Members

FULL OUTER Join Connections ON JB\_Members.member\_id=Connections.member\_id

ORDER BY JB\_Members.member\_name;



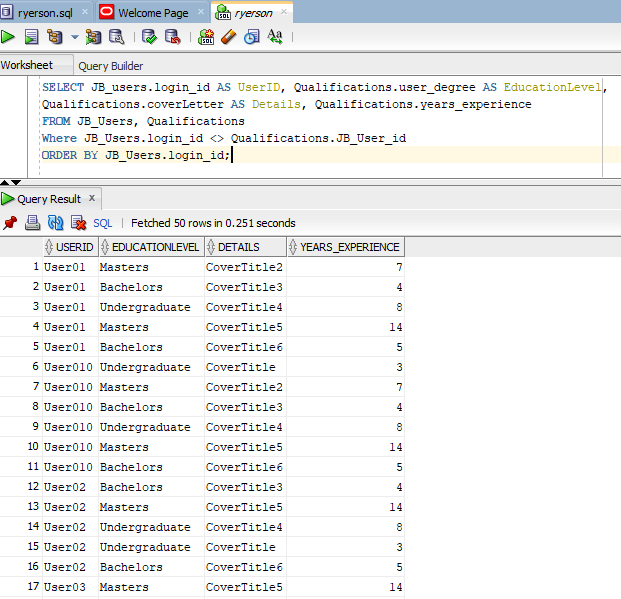
SELECT JB\_users.login\_id AS UserID, Qualifications.user\_degree AS EducationLevel,

Qualifications.coverLetter AS Details, Qualifications.years\_experience

FROM JB\_Users, Qualifications

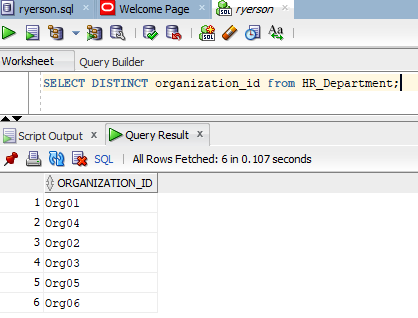
Where JB\_Users.login\_id <> Qualifications.JB\_User\_id

ORDER BY JB\_Users.login\_id;

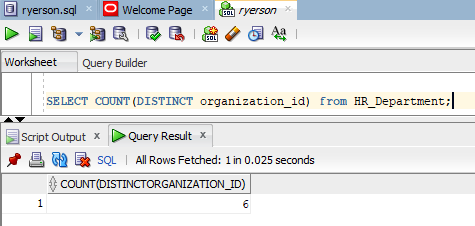


--DISTINCT Queries

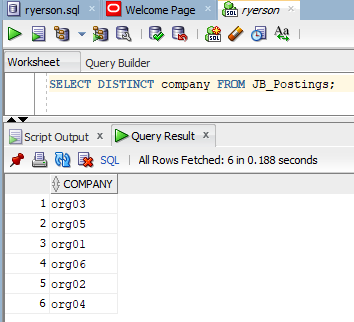
SELECT DISTINCT organization\_id from HR\_Department;



SELECT COUNT(DISTINCT organization\_id) from HR\_Department;

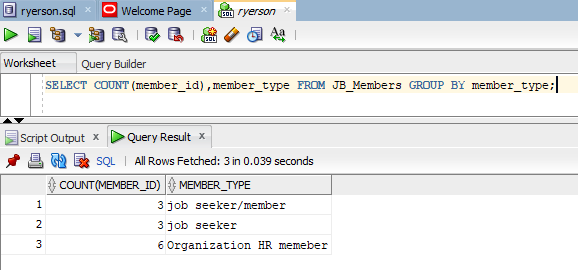


SELECT DISTINCT company FROM JB\_Postings;

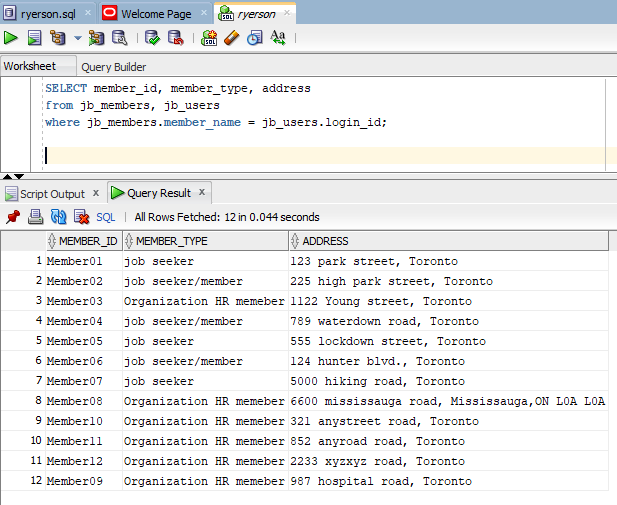


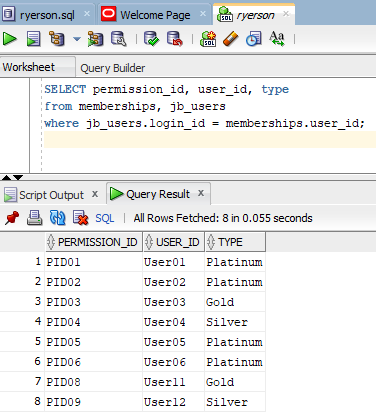
--Grouping/Sorting Commands

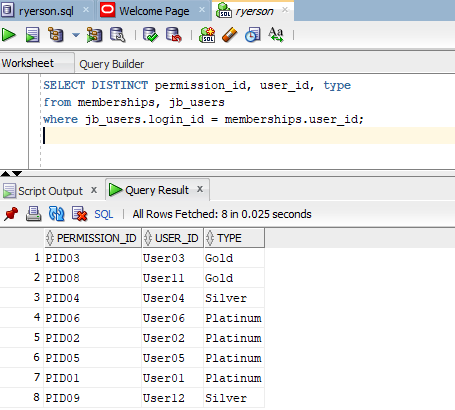
SELECT COUNT(member\_id),member\_type FROM JB\_Members GROUP BY member\_type;

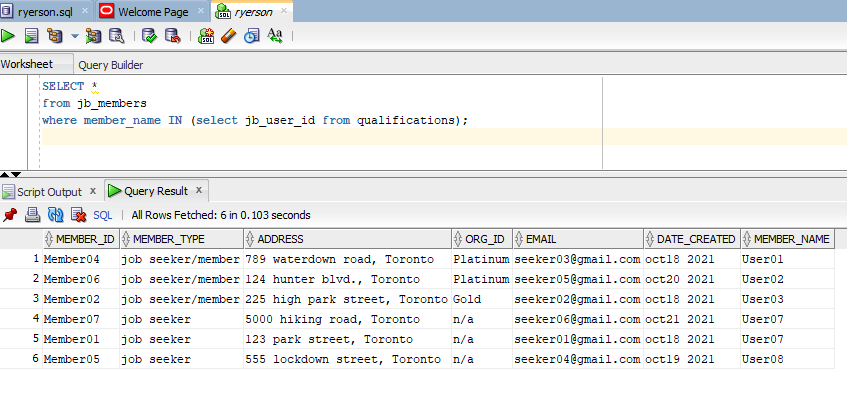


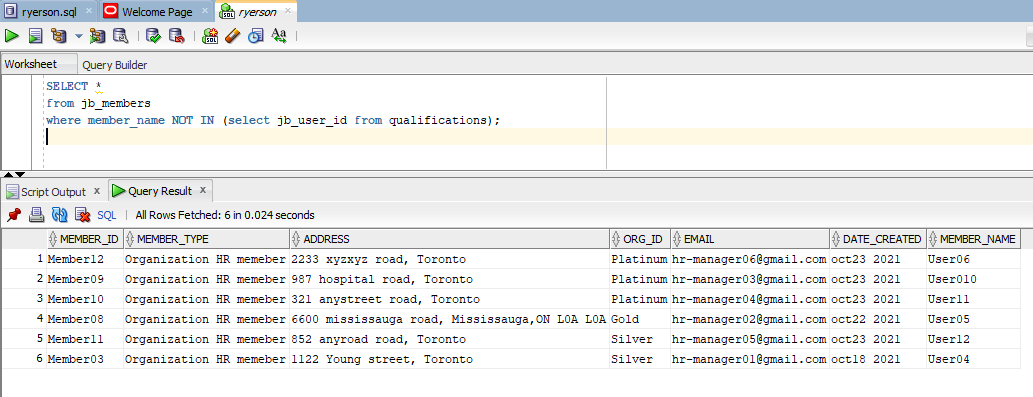
* Use of **Where** clause

****

****

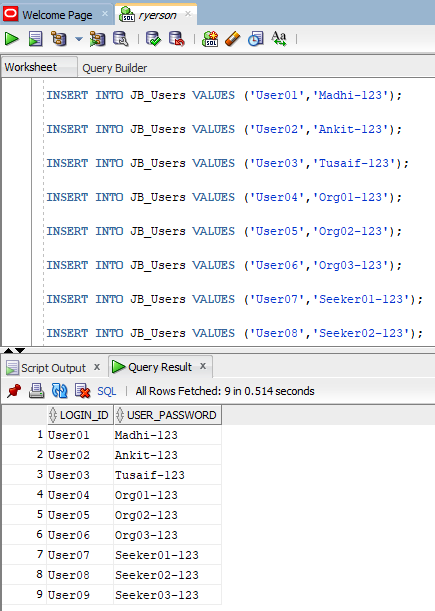
****

****

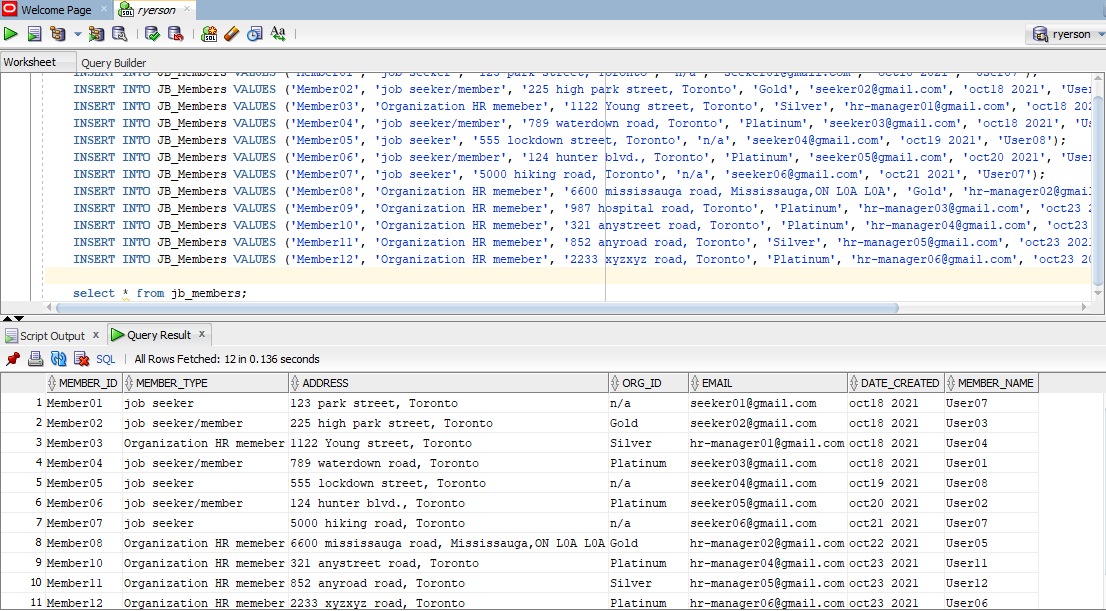
****

**Screen Shots:**

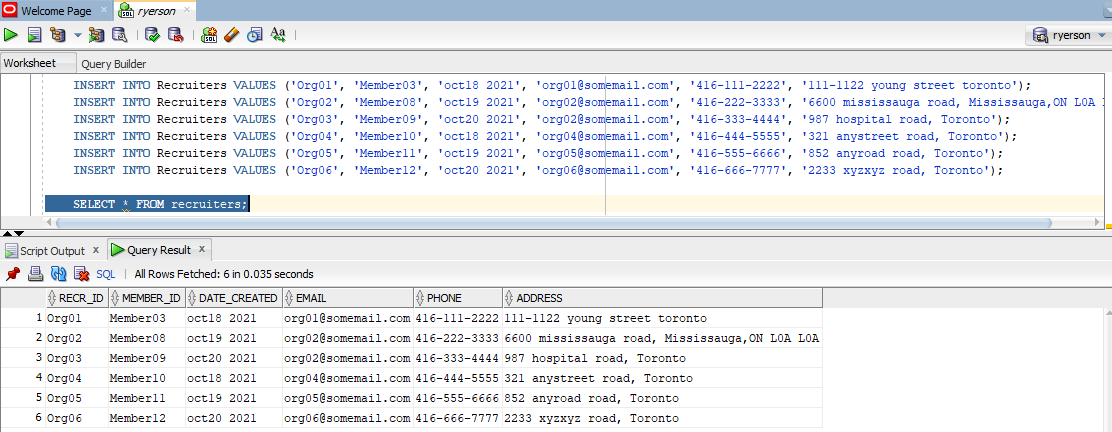
**JB\_Users Table Views and Queries.**

******

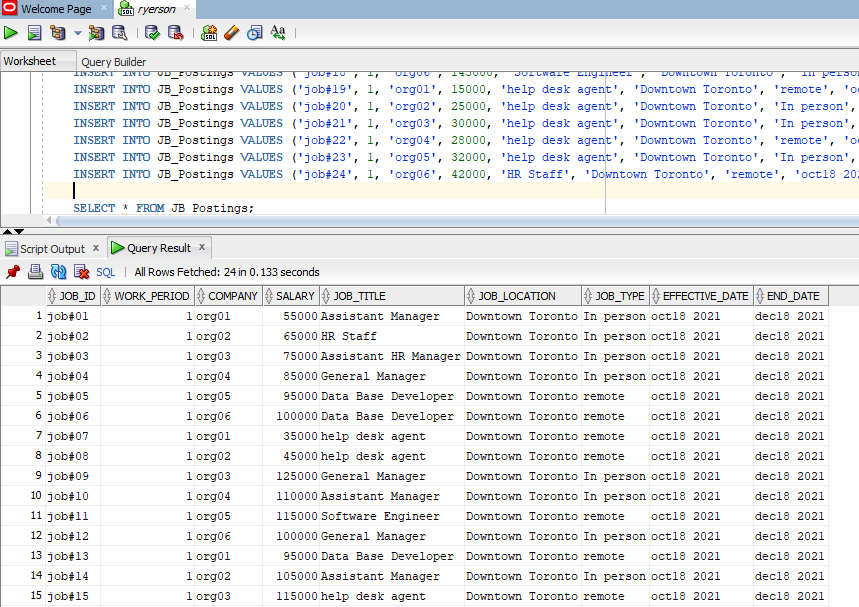
**JB\_Members Table Views and Queries.**

****

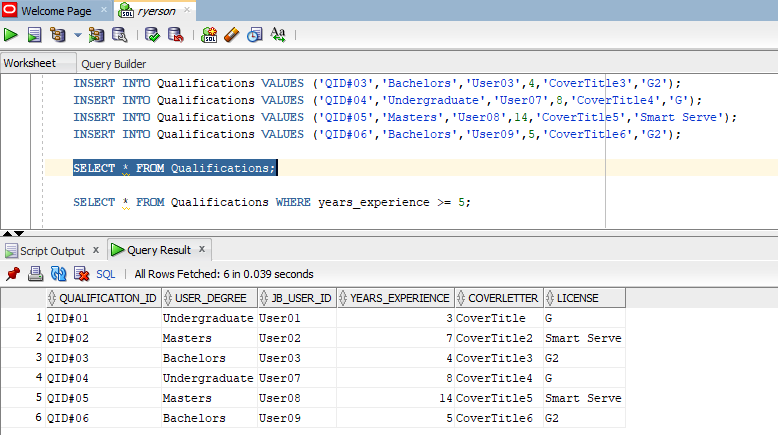
**JB\_Members Table Views and Queries.**

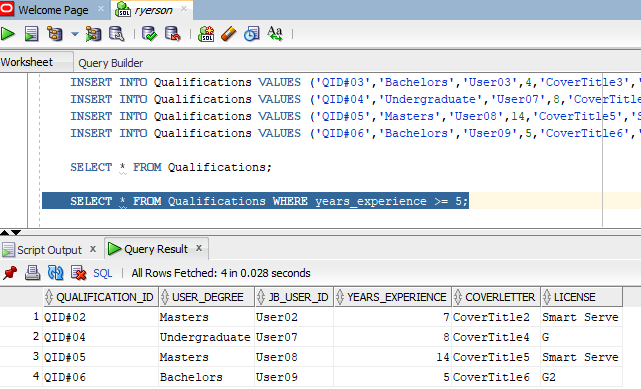
****

**JB\_Postings Table Views and Queries.**

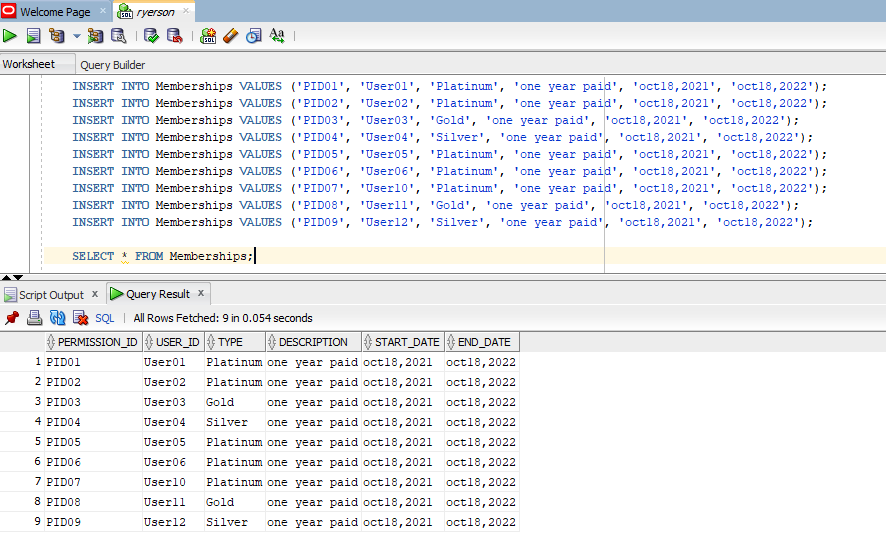
****

**Qualifications Table Views and Queries.**

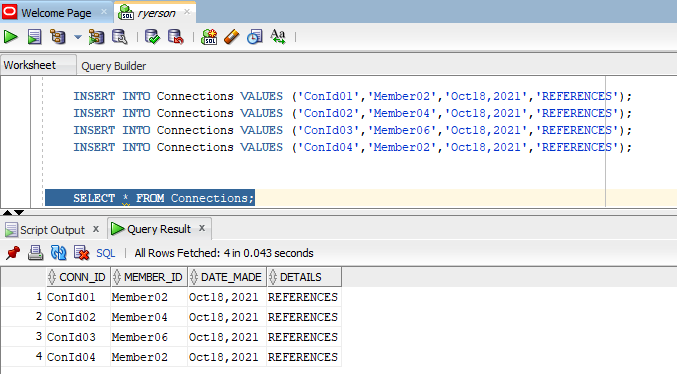




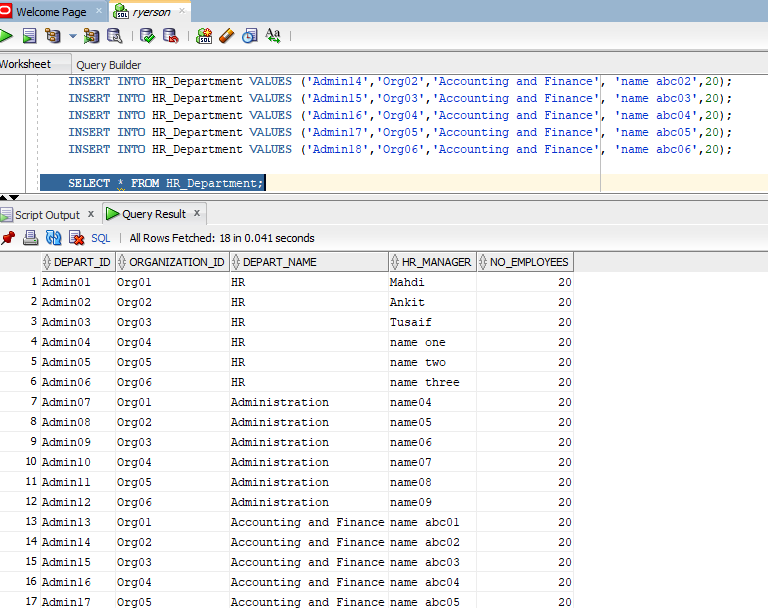
**Memberships Table Views and Queries.**



**Connections Table Views and Queries.**

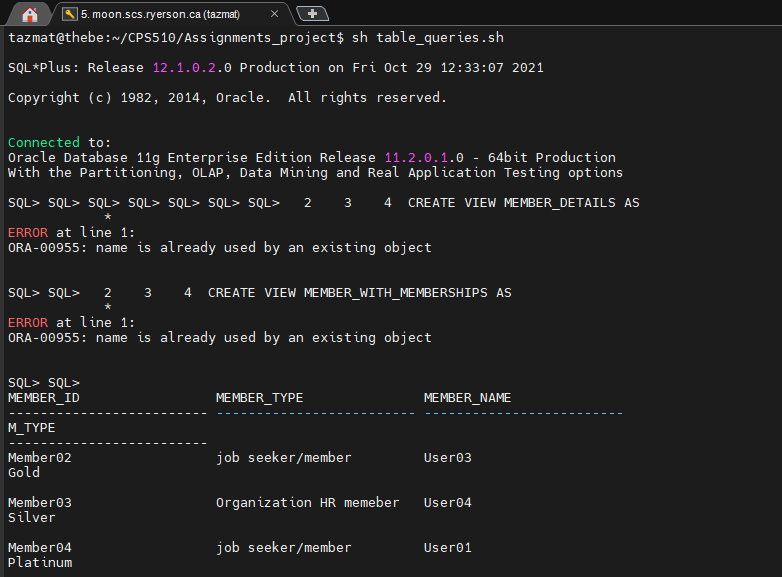


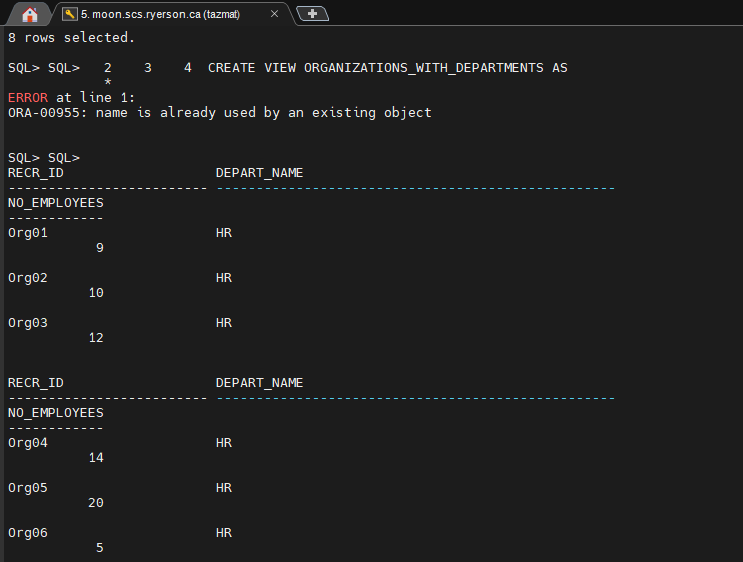
**HR\_Department Table Views and Queries.**

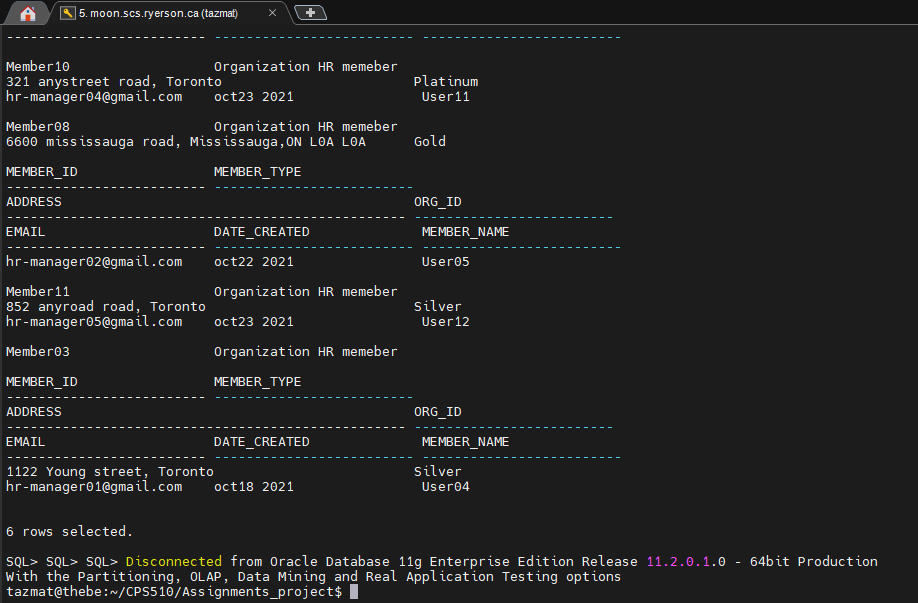
****

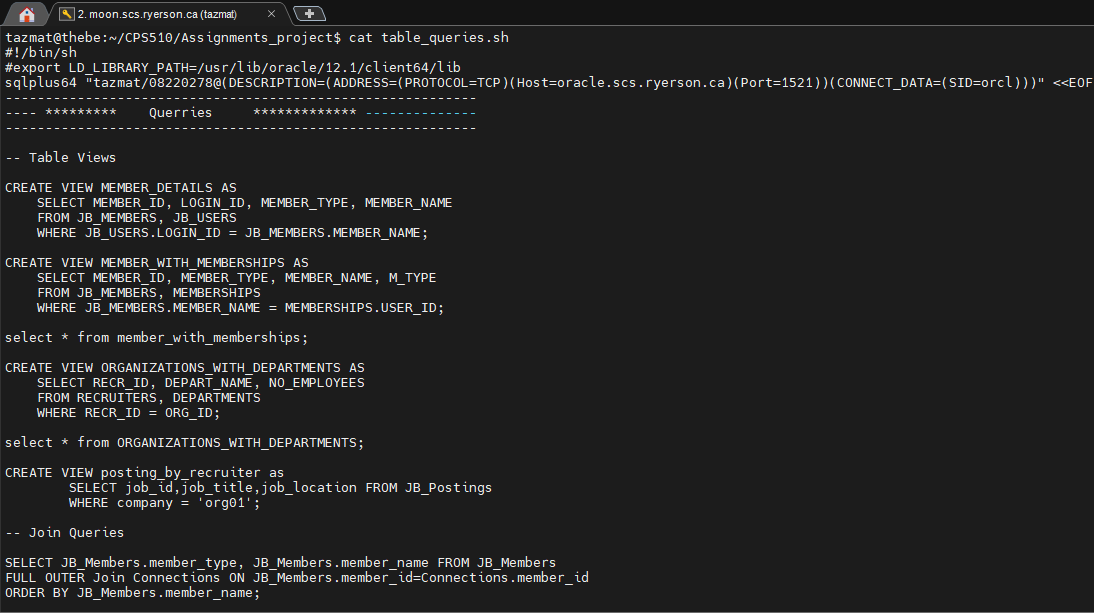
**Shell Script**

-- table\_queries.sh

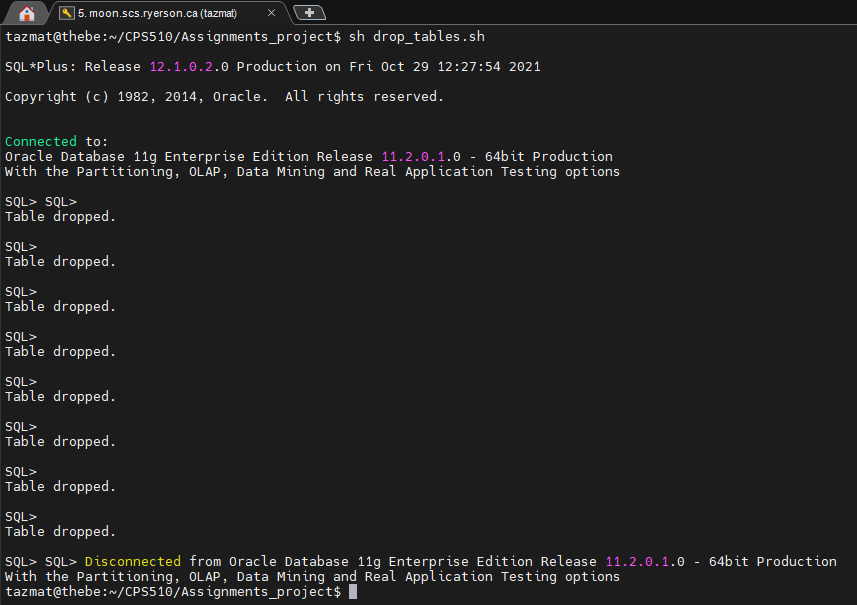
****

****

****

****

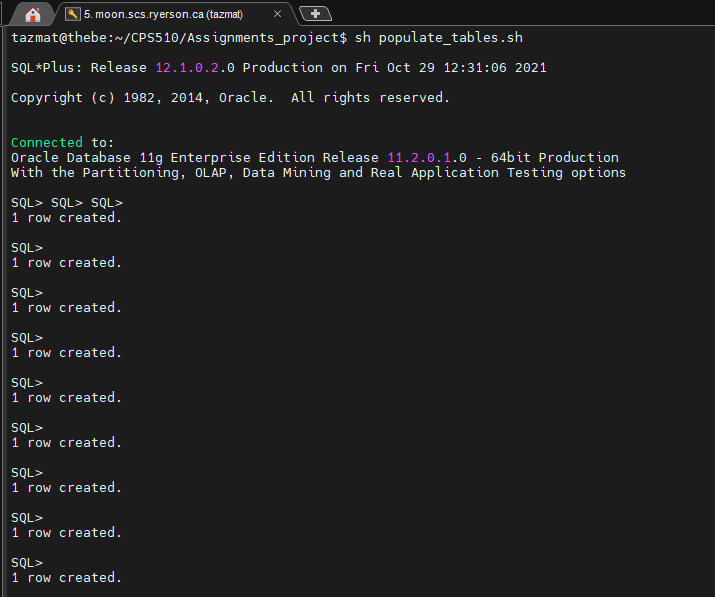
* **Drop tables**

****

* **Create tables**

****

* **Populate tables**



**Functional Dependencies**

**-- Functional Dependencies**

We have completed our database design and the next step is to normalize our DBMS design. In order to normalize our design, we first outlined the functional dependencies in the system. We normalized each table with 1NF and 2NF procedures.

We take each table and apply the normalization technique:

|  |  |  |
| --- | --- | --- |
| **JB\_Users** | | |
| PK | Login\_id  password | Char  Char |
| **JB\_Members** | | |
| PK  FK | Member\_id  Member\_type  Address  Email  Date\_created  Subscriptions  Login\_id | Char  Char  Char  Char  Date  Char  Char |

In the table JB\_Members,

Member\_type, Address, Email, Date\_created and Subscriptions depended on Member\_id.

So, Memebr\_id--🡪 Member\_type, Address,Email,Date\_created, subscriptions.

But Login\_id does not depend on Member\_id as it belongs to another table.

This table is of the form 1NF and 2NF.

In the table JB\_Users,

Password depend on Login\_id.

So, key Login\_id -🡪 password.

If we look at the relationship between these two tables, each JB\_Member will have one Login\_id, so Member\_id and login\_id will have one to one relationship and vice versa. This table is of the form 1NF and 2NF.

2.

|  |  |  |
| --- | --- | --- |
| **Recruiters** | | |
| PK  FK | recr\_id  Company\_name  Manager\_name  Member\_id | Char  Char  Char  Char |

In this table Recruiters recr\_id is a on which all the other attributes depend on company\_name and Manager\_name but Member\_id does not depend on recr\_id and that is okay as it belongs to another table and is a primary key to another table.

Represented as Recr\_id🡪company\_name,Manager\_name.

Recruiter Table has one to one relationship with JB\_Members table and holds dependencies. Recr\_id🡪Member\_id and vice versa. This table is of the form 1NF and 2NF.

3.

|  |  |  |
| --- | --- | --- |
| **Qualifications** | | |
| PK  FK | Qualification\_id  Edu\_level  Experience  Cover\_letter  Certi\_License  JB\_user\_id | Char  Char  Number  Char  Char  Char |

In this table Qualification\_id is a key on which all the other attributes depend.

Represented as Qulification\_id🡪Edu\_level,Experience,Cover\_leter and Certi\_license.

Whereas Jb\_user\_id is a key to another table and acts as forign key here and does not depends on Qualification\_id.

Having a foreign key makes the relationship between Qualifications table and JB\_Users table and we can describe that as a one to one relationship. As you see, each JB\_user\_id is associated with only one Qualification\_id. In other words each JB\_user will have one qualification to hold. This table is of the form 1NF and 2NF.

4.

|  |  |  |
| --- | --- | --- |
| **HR\_Department** | | |
| PK  FK | Depart\_id  Depart\_name  HR\_Manager  Org\_id | Char  Char  Number  Char |

In the table Departments key Depart\_id has attributes that directly depend on it.

We could represent by Depart\_id🡪Depart\_name,Hr\_Manger

Whereas Org\_id is a foreign key and doesn’t depend on depart\_id.

The table departments hold relation with Recruiters table as one to many relationships. That means Recruiter could have many departments but one department would have only one recruiter/company. Relationship between the two tables (Recruiter and Departments) will represent in term of functional dependencies as Depart\_id depend on Recr\_id (Depart\_id🡪Depart\_id).

5.

|  |  |  |
| --- | --- | --- |
| **JB\_Postings** | | |
| PK  FK | Job\_id  Company  Salary  Job\_title  Job\_location  Job\_type  Effect\_date  End\_date  depart\_id | Char  Char  Number  Char  Char  Char  Date  Date  Char |

This Table holds functional dependency with the HR\_Department table as Job\_id from JB\_Postings table depends on Depart\_id.

It holds many to one relationship with the HR\_department table as one HR will have many job postings.

If you look at the table JB\_posings all the attributes depend on the Job\_id key of the table.

We can represent this as Job\_id 🡪company,Salary,Job\_tile,Job\_location,Job\_type,Effect\_date and End\_date.

This table is of the form 1NF and 2NF.

6.

|  |  |  |
| --- | --- | --- |
| **Connections** | | |
| PK  FK | Conn\_id  Date\_made  Conn\_details  Member\_id | Char  Date  Char  Char |

This table holds the many-to-many relationship with JB\_Memebrs table as each connection will have many members and many connections belong to one member.

So conn\_id depends on Member\_id (conn\_id 🡪 member\_id) and member\_id depends on conn\_id (member\_id 🡪 conn\_id).

If we look at the conn\_id key all the other attribute depends on Conn\_id so date\_made and Conn\_details. This table is of the form 1NF and 2NF.

**Conclusion:** All the tables in our DBMS are normalized to 1NF and 2NF form.

**-- Normalization/ 3NF**

In order to normalize our design, we first outlined the functional dependencies in the system. We have normalized each table with 1NF and 2NF procedure and now we will apply the 3rd NF.

In our database no table use any transitive keys to make primary keys, so our database if of form 3NF.

We take each table and apply the 3NF normalization technique:

|  |  |  |
| --- | --- | --- |
| **JB\_Members** | | |
| PK  FK | Member\_id  Member\_type  Address  Date\_created  Subscriptions  Login\_id | Char  Char  Char  Char  Date  Char  Char |

All the non key attributes in the table depends on the primary key Member\_id and all non-key attribute are non-transitively dependent on Member\_id that makes it to 3NF.

JB\_Members (Member\_id, Member\_type,Address,Email,Date\_created,Subscriptions).

So, Member\_id--🡪 { Member\_type, Address,Date\_created, subscriptions}.

This table has only one candidate key and that is the primary key (Member\_id) of the table.

But Login\_id does not depend on Member\_id as it’s belonging to another table. This table is of the form 3NF.

2.

|  |  |  |
| --- | --- | --- |
| **JB\_Users** | | |
| PK | Login\_id  Password | Char  Char |

In the table JB\_Users, all the non-key attributes depends on key attribute which is Login\_id depends on Password.

Also there is no transitive key in the table.

So, key Login\_id -🡪 password.

**Note:** If we look at the relationship between these two tables, each JB\_Member will have one Login\_id, so Member\_id and Login\_id will have one to one relationship and vice versa. Both tables are functionally dependent and all primary keys are non-transitive. This table is of the form satisfies all three form 1NF, 2NF and 3NF.

3.

|  |  |  |
| --- | --- | --- |
| **Recruiters** | | |
| PK  FK | recr\_id  Company\_name  Manager\_name  Member\_id | Char  Char  Char  Char |

In this table Recruiters (recr**\_id**, Company\_name, Manager\_name) all the non-key attributes depends on recr\_id and that is a primary key of the table. This table has no transitive keys as all keys are unique.

Represented as Recr\_id🡪company\_name,Manager\_name.

This table is of the form 3NF.

4.

|  |  |  |
| --- | --- | --- |
| **Qualifications** | | |
| PK  FK | Qualification\_id  Edu\_level  Experience  Cover\_letter  Certi\_License  JB\_user\_id | Char  Char  Number  Char  Char  Char |

In this table Qualification\_id is a key on which all the other non-key attributes depends on and there is no key that is transitively dependent of any other table keys. All the keys of the table are unique.

Represented as Qualification\_id🡪Edu\_level,Expeience,Cover\_leter and Certi\_license.

Whereas Jb\_user\_id is a key to another table and acts as forign key her and does not depends on Qualification\_id. This table is of the form 3NF.

5.

|  |  |  |
| --- | --- | --- |
| **HR\_Department** | | |
| PK  FK | Depart\_id  Depart\_name  HR\_Manager  Org\_id | Char  Char  Number  Char |

All the non key attributes in the table depends on the primary key Depart\_id and all non-key attribute is non-transitively dependent on Depart\_id that makes it to 3NF.

All the non-key attributes are unique.

We could represent by Depart\_id🡪Depart\_name,Hr\_Manger

Whereas Org\_id is a foreign key and doesn’t depends on depart\_id. This table is of the form 3NF.

6.

|  |  |  |
| --- | --- | --- |
| **JB\_Postings** | | |
| PK  FK | Job\_id  Company  Salary  Job\_title  Job\_location  Job\_type  Effect\_date  End\_date  depart\_id | Char  Char  Number  Char  Char  Char  Date  Date  Char |

If you look at the table JB\_posings all the non-key attributes depends on Job\_id key of the table. There are no transitive relations in the table.

We can represent this as Job\_id 🡪company,Salary,Job\_tile,Job\_location,Job\_type,Effect\_date and End\_date.

This table is of the form 3NF.

7.

|  |  |  |
| --- | --- | --- |
| **Connections** | | |
| PK  FK | Conn\_id  Date\_made  Conn\_details  Member\_id | Char  Date  Char  Char |

If we look at the conn\_id key all the other non-key attribute depends on Conn\_id so 🡪 date\_made and Conn\_details. This table is of the form 3NF.

**Conclusion:** All the tables in our DBMS are normalized to 1NF, 2NF and 3NF form as all keys are unique.

Normalization / BCNF

**Conversion of 3NF to BCNF Normalize form by using Algorithm**

**-- Normalization/ Conversion of 3NF to BCNF with the help of Algorithm.**

**--- We used Bernstein’s Algorithm to achieve this task:**

**Bernstein’s Algorithm - Broken down into 4 steps:**

## Determine all the functional dependencies

## a) Find and remove redundancies

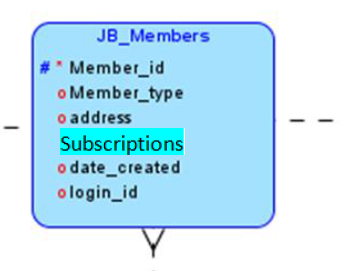
## b) Find and remove partial dependencies

## Find keys

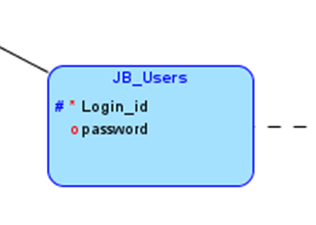
## Create tables

# Step 1 (finding all functional dependencies)

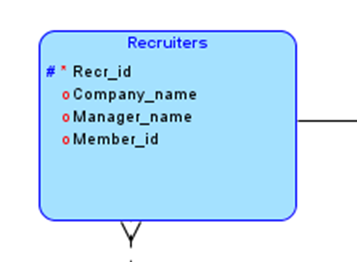
- **Member\_id** 🡪 {Member\_type, address, date\_created, Subscriptions}



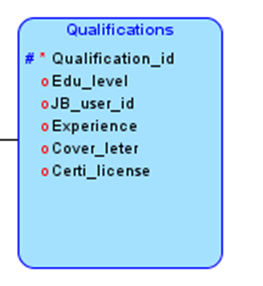
* **Login\_id** 🡪 { password}



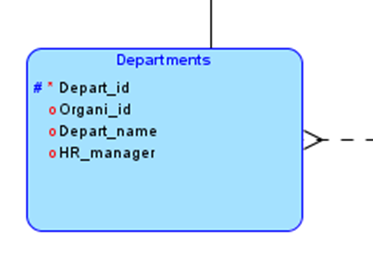
**--Recr\_id** 🡪 {company\_name, Manager\_name}



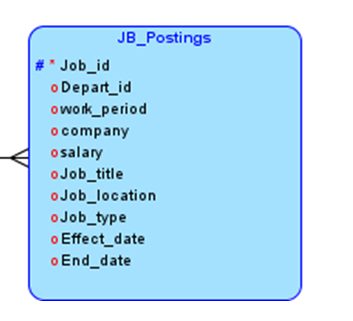
* **Qualification\_id🡪** {Edu\_level, Experience, cover\_letter, Certi\_license}



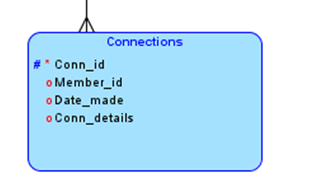
* **depart\_id** 🡪 {depart\_name, HR\_manager}



* **job\_id** 🡪 {work\_period, company, salary, job\_title, job\_location, job\_type, Effect\_date, End\_date}



* **Conn\_id** 🡪 {Date\_made, Conn\_details}



# Step 2a (Break RHS and find redundancies)

Get rid of Redundancies

- **Member\_id** 🡪 {Member\_type, address, date\_created, Subscriptions}

* + Reduced list of FD’s:
  + **Member\_id🡪** {Member\_type}
  + **Member\_id** 🡪{address}
  + **Member\_id** 🡪{date\_created}
  + **Member\_id** 🡪{Subscriptions}

# No redundancies

* **Login\_id** 🡪 { password}
  + Reduced list of FD’s:
  + **Login\_id** -> {password}

# No redundancies

* **Recr\_id** 🡪 {company\_name, Manager\_name}
  + Reduced list of FD’s:
  + **Recr\_id** 🡪 {company\_name}
  + **Recr\_id** 🡪 {Manger\_name}

# No redundancies

* **Qualification\_id 🡪** {Edu\_level, experience, cover\_letter, Certi\_license}
  + Reduced list of FD’s:
  + **Qualification\_id** 🡪 {user\_degree}
* **Qualification\_id** 🡪{ experience}
* **Qualification\_id** 🡪{cover\_letter}
* **Qualification\_id** 🡪{Certi\_license}

# No redundancies

* **depart\_id 🡪** {depart\_name, HR\_manager}
  + Reduced list of FD’s:
  + **depart\_id** 🡪 {depart\_name}
  + **depart\_id** 🡪 {HR\_manager}

# No redundancies

* **job\_id** 🡪 {work\_period, company, salary, job\_title, job\_location, job\_type, Effect\_date, End\_date}
  + Reduced list of FD’s:
  + **job\_id** 🡪{work\_period}
  + **job\_id** 🡪{company}
  + **job\_id** 🡪 {salary}
  + **job\_id** 🡪 {job\_title}
  + **job\_id** 🡪 {job\_location}
  + **job\_id** 🡪{job\_type}
  + **job\_id** 🡪{Effect\_date}
  + **job\_id** 🡪{End\_date}

# No redundancies

* **Conn\_id** 🡪 {money}
  + Reduced list of FD’s:
  + **conn\_id 🡪** {Date\_made}
  + **conn\_id 🡪** {conn\_details}

# No redundancies

**Step 2b (Minimize LHS, find and remove partial dependencies)**

* LHS is already minimized, therefore there are no partial dependencies

# Step 3(Find keys) (relational schema)

* **Member\_id🡪** {Member\_type, address, date\_created, subscriptions}
  + Attributes on RHS but not on LHS (cannot be keys)
    - Member\_type
    - Address
    - Date\_created
    - Subscriptions
  + Possible Keys
    - Member\_id
* **Login\_id** 🡪 { password}
  + Attributes on RHS but not on LHS (cannot be keys)
    - password
  + Possible Keys
    - Login\_id
* **Recr\_id** 🡪 {company\_name, manager\_name}
  + Attributes on RHS but not on LHS (cannot be keys)
    - Company\_name
    - Manger\_name
  + Possible Keys
    - Recr\_id
* **depart\_id** 🡪 {depart\_name, HR\_manager}
  + Attributes on RHS but not on LHS (cannot be keys)
    - depart\_name
    - HR\_Manger
  + Possible Keys
    - depart\_id
* **qualification\_id** 🡪 {Edu\_level, experience, cover\_letter, Certi\_license}
  + Attributes on RHS but not on LHS (cannot be keys)
    - Edu\_level
    - Experience
    - Cover\_letter
    - Certi\_license
  + Possible Keys
    - Qualification\_id
* **job\_id** 🡪 {work\_period, company, salary, job\_title, job\_location, job\_type, Effect\_date, End\_date}
  + Attributes on RHS but not on LHS (cannot be keys)
    - work\_period
    - company
    - salary
    - job\_title
    - job\_location
    - job\_type
    - Effect\_date
    - End\_date
  + Possible Keys
    - job\_id
* **Conn\_id** 🡪 {Date\_made, Conn\_details}
  + Attributes on RHS but not on LHS (cannot be keys)
    - Date\_made
    - Conn\_details
  + Possible Keys
    - Conn\_id

**Step 4(Make tables)**

**R1** (**member\_id**, member\_type, address, date\_created, subscriptions, login\_id)

With

FD: **member\_id** 🡪 { member\_type, address, date\_created , subscriptions }

**R2** (**Login\_id**, password)

With

FD: **user\_id** 🡪{ user\_password}

**R3** (**recr\_id**, company\_name, manager\_name, member\_id)

With

FD: **recr\_id** 🡪{company\_name, manager\_name }

**R4** (**depart\_id**, depart\_name, hr\_manager,Org\_id)

With

FD: **depart\_id** 🡪{ depart\_name, hr\_manager }

**R5** (**qualification\_id**, edu\_level, experience, cover\_letter, Certi\_license, user\_id)

With

FD: **qualification\_id** 🡪 { edu\_level, experience, cover\_letter, Certi\_license }

**R6** (**job\_id**, work\_period, company, salary, job\_title, job\_location, job\_type, effect\_date,end\_date, depart\_id)

With

FD: **job\_id** 🡪{, work\_period, company, salary, job\_title, job\_location, job\_type, effect\_date,end\_date }

**R7** (**conn\_id**, date\_made, conn\_details, member\_id)

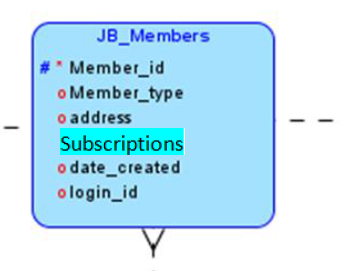
With

FD: **conn\_id** 🡪 { date\_made, conn\_details }

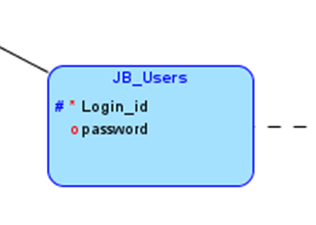
**BCNF (Boyce/Codd Normal Form)**

# Step 1 (finding all functional dependencies (List of all attributes and FDs))

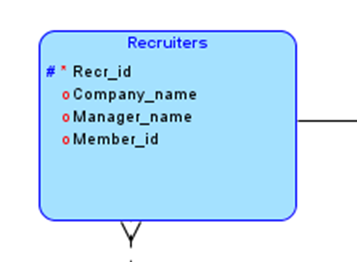
- **member\_id** 🡪 { member\_type, address, date\_created, subscriptions, login\_id }



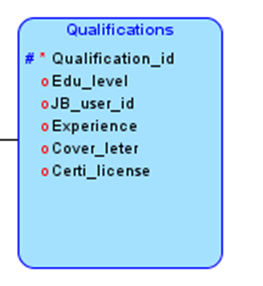
* **Login\_id** 🡪 { password}



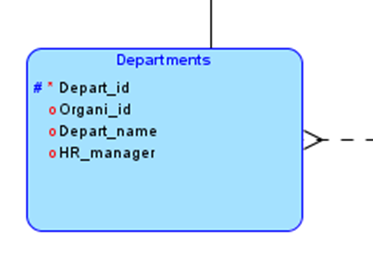
* **Recr\_id** 🡪 {company\_name, Manager\_name, member\_id}



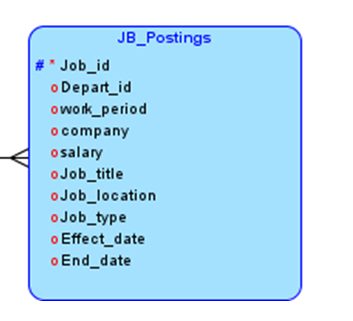
* **Qualification\_id🡪** {Edu\_level, Experience, cover\_letter, Certi\_license, jb\_user\_id}



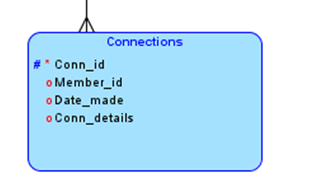
* **depart\_id** 🡪 {depart name, HR\_manager, Organi\_id}



* **job\_id** 🡪 {work\_period, company, salary, job\_title, job\_location, job\_type, Effect\_date, End\_date, depart\_id, Depart\_id}



* **Conn\_id** 🡪 {Date\_made, Conn\_details, member\_id}



# Step 2 (make sure that the left hand side are keys, if not decompose)

Consider the relation schema

**R1** (**member\_id**, member\_type, address, date\_created, subscriptions, login\_id)

With FD

**member\_id** 🡪 { member\_type, address, date\_created , subscriptions }

This schema has one candidate key

member\_id

Therefore this schema is in BCNF

Consider the relation schema

**R2** (**Login\_id**, password)

With FD

**Login\_id** 🡪 {password}

This schema has one candidate key

login\_id

Therefore this schema is in BCNF

Consider the relation schema

**R3** (**recr\_id**, company\_name, manager\_name, member\_id)

With FD

**recr\_id** 🡪 {company\_name, manager\_name }

This schema has one candidate key

recr\_id

Therefore this schema is in BCNF

Consider the relation schema

**R4** (**depart\_id**, depart\_name, hr\_manager,Org\_id)

With FD

**depart\_id** 🡪 { depart\_name, hr\_manager }

This schema has one candidate key

depart\_id

Therefore this schema is in BCNF

Consider the relation schema

**R5** (**qualification\_id**, edu\_level, experience, cover\_letter, Certi\_license, user\_id)

With FD

**qualification\_id** 🡪 { edu\_level, experience, cover\_letter, Certi\_license }

This schema has one candidate key

qualification\_id

Therefore this schema is in BCNF

Consider the relation schema

**R6** (**job\_id**, work\_period, company, salary, job\_title, job\_location, job\_type, effect\_date,end\_date, depart\_id)

With FD

**job\_id** 🡪 { work\_period, company, salary, job\_title, job\_location, job\_type,effect\_date,end\_date}

This schema has one candidate key

job\_id

Therefore this schema is in BCNF

Consider the relation schema

**R7** (**conn\_id**, date\_made, conn\_details, member\_id)

With FD

**conn\_id** 🡪 { date\_made, conn\_details }

This schema has one candidate key

conn\_id

Therefore this schema is in BCNF

**Step 3 (final BCNF schema for R)**

**R1** (**member\_id**, member\_type, address, date\_created, subscriptions, login\_id)

**R2** (**Login\_id**, password)

**R3** (**recr\_id**, company\_name, manager\_name, member\_id)

**R4** (**depart\_id**, depart\_name, hr\_manager,Org\_id)

**R5** (**qualification\_id**, edu\_level, experience, cover\_letter, Certi\_license, user\_id)

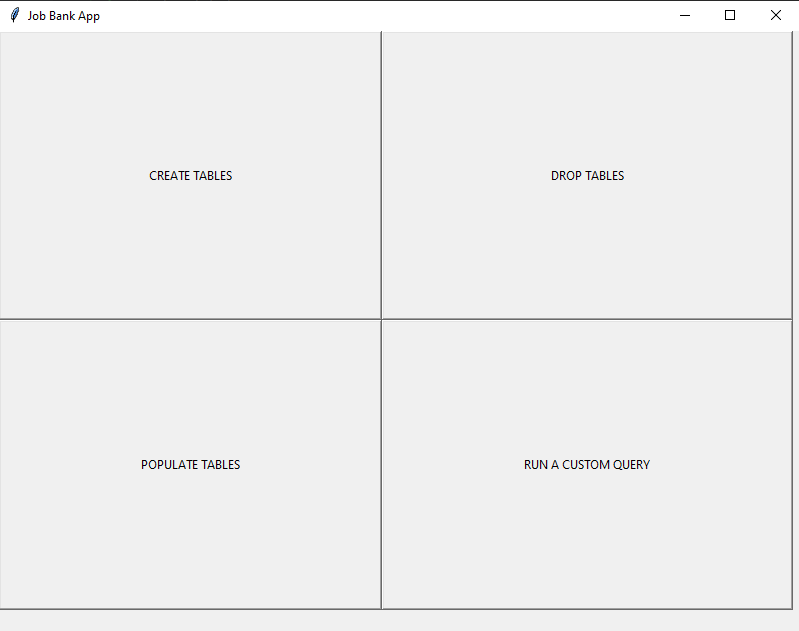
**R6** (**job\_id**, work\_period, company, salary, job\_title, job\_location, job\_type, effect\_date,end\_date, depart\_id)

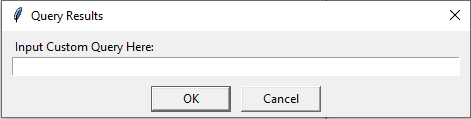
**R7** (**conn\_id**, date\_made, conn\_details, member\_id)

Note: All the tables are of the form BCNF and we didn’t need to combine any tables for this stage.

**Graphical User Interface Python Code:**

#!/usr/bin/env python  
import tkinter as tk  
import cx\_Oracle  
import connect\_info  
from tkinter import N, S, W, E, simpledialog  
  
  
def find\_query\_name(query):  
 *"""  
 Used to find a query name  
 """* return query[:query.find('(')]  
  
  
class main\_window:  
 *"""  
 The main application window containing up to 4 buttons:  
 CREATE,POPULATE,DROP Tables, and Run a custom query  
 """* def \_\_init\_\_(self):  
 self.root = tk.Tk()  
 self.root.title('Job Bank App')  
 self.root.geometry("600x400")  
 self.label = tk.Label(self.root)  
  
 self.buttonTopLeft = tk.Button(self.root,  
 text='Please Connect to DataBase',  
 command=self.connect\_to\_db)  
 self.buttonTopRight = tk.Button(self.root,  
 text='DROP TABLES',  
 command=self.dropTables)  
 self.buttonBottomLeft = tk.Button(self.root,  
 text='POPULATE TABLES',  
 command=self.populateTables)  
 self.buttonBottomRight = tk.Button(self.root,  
 text='RUN A CUSTOM QUERY',  
 command=self.runCustomQuery)  
  
 self.buttonTopLeft.grid(column=0, row=0, sticky=N + S + E + W)  
 self.buttonTopRight.grid(column=1, row=0, sticky=N + S + E + W)  
 self.buttonBottomLeft.grid(column=0, row=1, sticky=N + S + E + W)  
 self.buttonBottomRight.grid(column=1, row=1, sticky=N + S + E + W)  
 self.label.grid(column=2, row=2, sticky=N + S + E + W)  
  
 # set weight to make the window responsive  
 for i in range(2):  
 self.root.grid\_rowconfigure(i, weight=1)  
 for i in range(2):  
 self.root.grid\_columnconfigure(i, weight=1)  
  
 # temporary hide the 3 other buttons  
 self.buttonTopRight.grid\_remove()  
 self.buttonBottomLeft.grid\_remove()  
 self.buttonBottomRight.grid\_remove()  
 # run the mainloop  
 self.root.mainloop()  
  
 def quit(self):  
 self.root.destroy()  
 self.cursor.close()  
 self.connection.close()  
  
 def createTables(self):  
 create\_file = open('createTables.txt', 'r')  
 create\_tables\_list = create\_file.read().split(';')  
 try:  
 for table in create\_tables\_list:  
 print(f"Executing {find\_query\_name(table)}")  
 self.connection.cursor().execute(table)  
 string = "Successfully created tables"  
 self.connection.cursor().execute('CREATE TABLE Qualifications (qualification\_id VARCHAR2(10 CHAR),edu\_level VARCHAR2(50 CHAR),JB\_User\_id VARCHAR2(25 CHAR),experience NUMBER,coverLetter VARCHAR2(1000 CHAR),certi\_license VARCHAR2(25 CHAR),PRIMARY KEY(qualification\_id),FOREIGN KEY(JB\_User\_id) REFERENCES JB\_Users(login\_id));')  
 except:  
 string = 'Some Tables may not be created...'  
 string = "Successfully created tables"  
 newWin = tk.Tk()  
 textBox = tk.Text(newWin)  
 textBox.insert(tk.INSERT, string)  
 textBox.config(state=tk.DISABLED)  
 textBox.pack()  
 newButton = tk.Button(newWin, text='close', command=newWin.destroy)  
 newButton.pack()  
 newWin.mainloop()  
  
 def populateTables(self):  
 populate\_file = open("populateTables.txt", "r")  
 populate\_tables\_string = populate\_file.read().split(';')  
 try:  
 for table in populate\_tables\_string:  
 print(f"Executing {find\_query\_name(table)}")  
 self.connection.cursor().execute(table)  
 string = "Successfully populated tables"  
 except:  
 string = 'some table might not got populated...'  
 string = "Successfully populated tables"  
 newWin = tk.Tk()  
 textBox = tk.Text(newWin)  
 textBox.insert(tk.INSERT, string)  
 textBox.config(state=tk.DISABLED)  
 textBox.pack()  
 newButton = tk.Button(newWin, text='close', command=newWin.destroy)  
 newButton.pack()  
  
 newWin.mainloop()  
  
 def dropTables(self):  
 drop\_file = open('dropTables.txt', 'r')  
 drop\_tables\_list = drop\_file.read().split(";")  
 try:  
 for drop\_table in drop\_tables\_list:  
 self.connection.cursor().execute(drop\_table)  
 string = "Successfully dropped tables"  
 except:  
 string = 'Some tables might not got dropped.'  
 string = "Successfully dropped tables"  
 newWin = tk.Tk()  
 textBox = tk.Text(newWin)  
 textBox.insert(tk.INSERT, string)  
 textBox.config(state=tk.DISABLED)  
 textBox.pack()  
 newButton = tk.Button(newWin, text='close', command=newWin.destroy)  
 newButton.pack()  
 newWin.mainloop()  
  
 def runCustomQuery(self):  
 try:  
 query = simpledialog.askstring(title="Query Results", prompt=("Input Custom Query Here: " + " " \* 100))  
 self.cursor.execute(query)  
  
 result = self.cursor.fetchall()  
 print(result)  
 string = ''  
  
 for res in result:  
 for x in res:  
 string = string + str(x) + ', \n'  
 string = string + '\n'  
 except cx\_Oracle.Error as error:  
 string = error  
  
 newWin = tk.Tk()  
 textBox = tk.Text(newWin)  
 textBox.insert(tk.INSERT, string)  
 textBox.config(state=tk.DISABLED)  
 textBox.pack()  
 newButton = tk.Button(newWin, text='close', command=newWin.destroy)  
 newButton.pack()  
  
 newWin.mainloop()  
  
 def connect\_to\_db(self):  
 *"""  
 Connect to the Oracle DMBS and restore the right layout through  
 self.restore\_layout()  
 """* self.connection = None  
 try:  
 self.connection = cx\_Oracle.connect(  
 connect\_info.username,  
 connect\_info.password,  
 connect\_info.dsn,  
 encoding=connect\_info.encoding)  
 self.cursor = self.connection.cursor()  
  
 # show the version of the Oracle Database  
 string = 'Connected successfully to: ' + self.connection.version  
 print(self.connection.version)  
 except cx\_Oracle.Error as error:  
 string = error  
 print(error)  
  
 # restore the layout to the correct one  
 self.restore\_layout()  
  
 newWin = tk.Tk()  
 textBox = tk.Text(newWin)  
 textBox.insert(tk.INSERT, string)  
 textBox.config(state=tk.DISABLED)  
 textBox.pack()  
 newButton = tk.Button(newWin, text='close', command=newWin.destroy)  
 newButton.pack()  
  
 newWin.mainloop()  
  
 def restore\_layout(self):  
 *"""  
 Change the text on the right button and restore the other 3 buttons in the grid  
 """* # change the text on the top left button  
 self.buttonTopLeft.configure(text="CREATE TABLES", command=self.createTables)  
  
 # return buttons to the window  
 self.buttonTopRight.grid()  
 self.buttonBottomLeft.grid()  
 self.buttonBottomRight.grid()  
 self.root.geometry("800x600")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main\_window()





**Simple and Advanced Database Queries (with Relational Algebra):**

-----------------------------------------------------------

---- \*\*\*\*\*\*\*\*\* Queries \*\*\*\*\*\*\*\*\*\*\*\*\* --------------

-----------------------------------------------------------

**SQL**

SELECT \*

FROM JB\_Postings

WHERE job\_location = 'Toronto';

**Relational Algebra**

𝛔 Location = ‘Toronto’ (JB\_postings)

**SQL**

SELECT \*

FROM Qualifications

WHERE years\_experience >= 5;

**Relational Algebra**

𝛔 year\_experience ≥ 5 (Qualifications)

**SQL**

SELECT \*

FROM JB\_Members

WHERE Subscription = 'Premium' AND member\_type = 'recruiter';

**Relational Algebra**

𝛔 (Subscription = ‘Premium’) AND (member\_type = ‘recruiter’) (JB\_Members)

**SQL**

SELECT \*

FROM Department

ORDER BY Org\_id DESC, depart\_name ASC;

**Relational Algebra**

𝛔 ( (Org\_id DESC) AND (depart\_name ASC)) (Department)

**SQL**

SELECT company\_name AS company

FROM JB\_Recruiters

ORDER BY Manager\_name;

**Relational Algebra**

𝛔 company\_name ASC 𝚷 Manager\_name (JB\_Recruiters)

**SQL**

SELECT job\_location, COUNT(job\_id) AS INTEGER\_located

FROM JB\_Postings

GROUP BY job\_location;

**Relational Algebra**

𝚷 job\_location, job\_id (𝓖 job\_location )( JB\_Postings)

**SQL**

SELECT login\_id, edu\_level, experience

FROM Qualifications, JB\_Members

24

WHERE ((edu\_level = 'University of Toronto' OR edu\_level = Seneca

College')

AND experience BETWEEN 2 AND 10);

**Relational Algebra**

𝚷 login\_id, edu\_level, experience (𝛔 (edu\_level = ‘University of Toronto) OR (edu\_level = ‘Seneca College’) AND

(experience ≥ 2) AND (experience ≤ 10)) (Qualifications >< JB\_Members)

**SQL**

SELECT subscription, login\_id, edu\_level, job\_title

FROM JB\_Postings j, Qualifications q, JB\_Users a, JB\_Members s

WHERE (s.subscription = ‘Gold’ AND q.edu\_level = Seneca College');

**Relational Algebra**

𝚷 subscription, login\_id, edu\_level, job\_titile (𝛔 (subscription = ‘Gold’) AND (user\_degree = ‘Seneca College’) )(JB\_Postings ><

Qualifications >< JB\_Users >< JB\_Members)

**SQL**

SELECT JB\_Members.member\_id, JB\_Members.date\_created, Connections.date\_made,

Connections.details

FROM JB\_Members

INNER JOIN Connections

ON JB\_Members.member\_id = Connections.member\_id;

**Relational Algebra**

𝚷 member\_id, date\_created, date\_made, details (𝛔 member\_id = member\_id) (JB\_Members Connections)

**Conclusion:**

## Working on this Online Job Bank Database Management System has provided us with a foundation in all the aspects of database design and implementation. Through the course of the 10 assignments, we realized how important it was to represent data in a clear and concise manner. With the theoretical knowledge and the understanding of entity relationship diagrams, relational schema design, functional dependencies, normalization, etc., we were able to turn various meaningless sets of data into a useful and accessible database.

## In the technical aspect, we familiarize ourselves with the services provided by Oracle 12g and SQL. Using the tools provided, we learned how to create tables, drop tables, populate tables and create custom queries that can display query information in any format. Finally, creating a GUI using python also familiarized us with the interactions between the front-end and back-end components of the database.

## This project was also crucial in exercising our teamwork and communication skills along with improving our technical abilities with project management and software development.

## Overall, working on the job bank DBMS was different, challenging yet an enjoyable experience. It truly unlocked our potential to use the knowledge and skills that we have acquired in this course for the future career development.