Online Job Bank DBMS

Table of Contents

Introduction/Overview

In this report, a database management system serving as the theoretical backend for an online job bank is proposed. The process of construction and implementation of said database is detailed from the conceptual (entity relationship diagrams, schema, etc.) to a final BCNF-normalised implementation in SQL, accompanied by a Java-created GUI to interact with it. Afterwards, a showcase of all the database's queries as converted to the relational algebra notation is shown.

As a brief overview, the database has eight data types:

- Jobs
- Job applications
- Job applicants
- Companies
- Recruiters
- Interviews
- Resumes

More information on each data type is listed below.

Jobs, representing job postings on the job bank, will be differentiated from one another using a unique ID (primary key). It also has various attributes, such as title, job description, requirements, type, location, salary, and working hours. A single job can receive many **job applications**, and each job is posted by one **recruiter**.

Job applications are submitted by **job applicants** to specific **jobs**. Each job application has attributes such as its application ID, which is the primary key attribute, along with the referenced **job** ID, the application date, the status of the application, an attached **resume**, any scheduled **interviews,** and optional additional files. A **job** can receive multiple applications, and applicants can submit applications to multiple **jobs**, but each application is always tied to one specific job.

Job applicants are the users looking for jobs. Differentiated by the applicant ID as a primary key, information such as their name, email, phone number, address, skills, education, birthdate, and work experience is stored as attributes. Job applicants can also have multiple **resumes,** letting them tailor them to different jobs and job applications.

Resumes are simply just resume file uploads by job applicants to use in their applications. They possess a resume ID, the primary key attribute, along with attributes such as file, job applicant ID (since the resume is linked to the job applicant), and an upload date. Job applicants may keep multiple resumes in their account, which can be attached to one or more job applications.

Interviews can be scheduled for job applications. Attributes include interview ID (primary key attribute), date, time, location/interview link, and the associated job application ID. Each interview is associated with a specific job application, and a single application may lead to

multiple interviews. This can be tracked within the system for scheduling and feedback purposes.

Employers are broken down into two entities: **companies** and **recruiters**.

A **Company** represents an organization that posts jobs. Each company has attributes such as company ID, the primary key attribute, as well as the company name, industry, jobs posted, recruiters, location, email, and phone number. Companies may have multiple recruiters and numerous jobs. Companies are referenced by jobs as their employers.

A **Recruiter** is a user who manages job postings, linked to a company. Recruiters have attributes such as the recruiter ID (primary key attribute), name, email, phone, a linked company ID, and the jobs they have posted. Each recruiter is linked to one company, but a company can employ multiple recruiters. Recruiters are responsible for creating jobs, reviewing job applications, and scheduling interviews.

System Functions for Applicants
Applicants can create an account using an email and a password. Setting up a profile involves entering personal information such as first and last name, and a resume. Additional information, such as phone number and location, can be added. Applicants can search for jobs using filters such as searching by keyword, location, salary, company, and skills in the job search. Once they see a job posting they like, they can apply for it. Each job application involves attaching a resume, any supporting documents, and answering questions from the recruiters. After applying, applicants can view all applied jobs and the application’s status (pending, rejected, accepted). Additionally, applicants have the option to bookmark jobs if they wish to review and apply later. Applicants can receive interviews from recruiters, which include details such as date, time, location, or an online interview link.

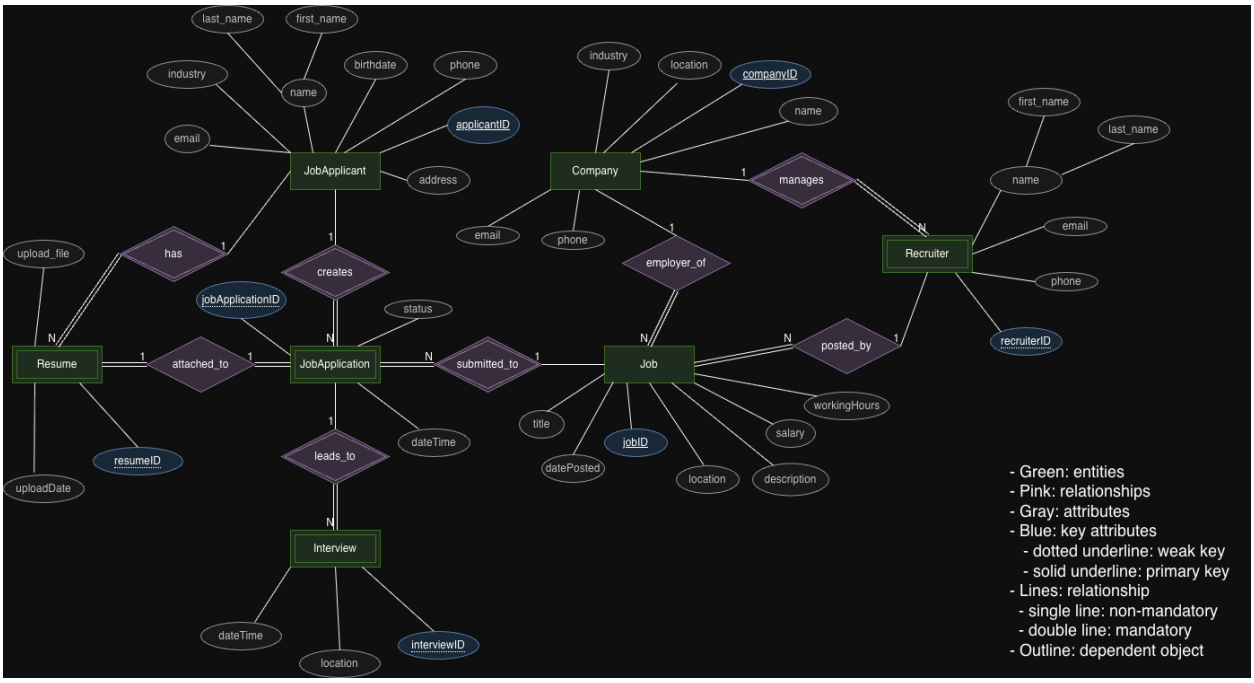
System Functions for Recruiters
Recruiters can create an account by entering an email and a password. Setting up a profile involves entering the first and last name, as well as entering the company they are associated with. Recruiters can create a job post by entering details such as job name, the company hiring, salary, full-time/part-time, location, and job description. Recruiters create application questions for applicants to answer. Furthermore, they can view job applications from jobs they have posted, shortlist applicants, and schedule interviews. Recruiters can also change an applicant's application status to either rejected or accepted. Recruiters can also remove job postings if needed.

System Functions for Companies
Companies can create an account by email and password, which is used for login. Furthermore, additional information is required, such as company name, industry, location, contact person, email, and phone number. Setting up a profile involves posting a company logo and description. Companies can view and manage recruiters working within the company and see what jobs they have posted. Companies can also view all jobs posted under the company, view job

applications for each job posting, shortlist applicants, and schedule interviews. Companies may remove job postings if needed.

Conceptual Design

ER Diagram



The following ER model is for our Online Job Bank system. It consists of 7 entities: JobApplicant, Company, Recruiter, Job, JobApplication, Resume, and Interview. Among these, Resume, JobApplication, Interview, and Recruiter are weak entities since they are all dependent on their related strong entity for identification. The model defines 8 relationships: has, manages, employer_of, posted_by, submitted_to, creates, leads_to, and attached_to. The model also addresses 8 cardinalities.

Attributes are categorized into primary key, weak, and regular attributes, with key identifiers such as applicantID, resumeID, companyID, recruiterID, jobID, interviewID, and jobApplicationID. Total participation is denoted by double lines, while bordered entities represent weak entities. Thus, this ER model captures multiple participation constraints, providing a comprehensive view of the relationships, dependencies, and data flow within our Online Job Bank System.

Database schema (table format)

JOB

jobID (primary key)	companyID (references **COMPANY**.companyID)	recruiterID (references **RECRUITER.** recruiterID)	employer	salary	workingHours	datePosted	location	title	description

JOB_APPLICATION

jobAppID (primary key)	jobID **(**references **JOB.**jobID**)**	applicantID (references **JOB_APPLICANT**.applicantID)	dateTime	status

RECRUITER

recruiterID (primary key)	companyID (references **COMPANY**.companyID)	first_name	last_name	email	phone

COMPANY

companyID (primary key)	name	industry	location	email	phone

JOB_APPLICANT

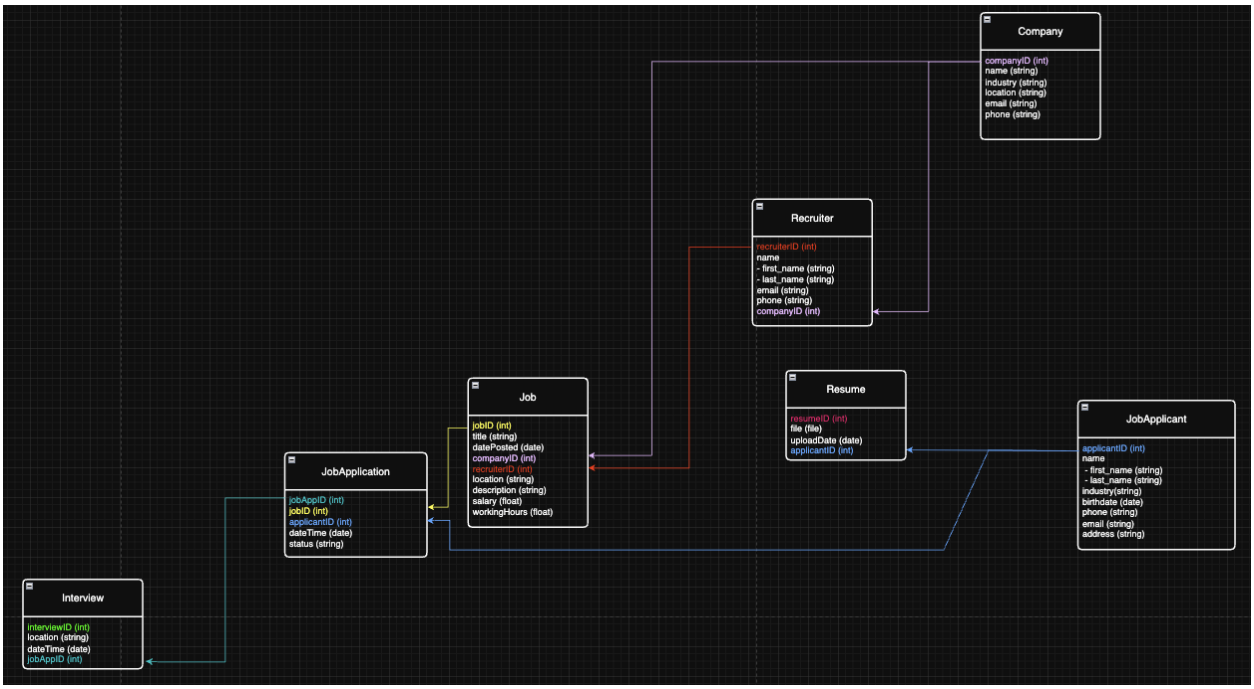
applicantID (primary key)	first_name	last_name	industry	birthdate	address	email	phone

RESUME

resumeID (primary key)	applicantID (references **JOB_APPLICANT.**applicantID**)**	upload_file	uploadDate

INTERVIEW

interviewID (primary key)	jobAppID (references **JOB_APPLICATION.**jobAppID**)**	dateTime	location



Below are some screenshots of our SQL code designed in Oracle using SQL Developer. The SQL statements create the tables in our system for the entities and define the attributes of each entity with their data type (INTEGER, VARCHAR(), FLOAT, DATE, etc.). The statements also included required constraints, such as PRIMARY KEY, NOT NULL, UNIQUE, and FOREIGN KEY.

Internal Design

Below are images of the database tables as made by SQL statements.

```sql
CREATE TABLE Company (
    companyID       INTEGER PRIMARY KEY,
    name            VARCHAR(30) NOT NULL UNIQUE,
    industry        VARCHAR(100),
    location        VARCHAR(200),
    email           VARCHAR(100),
    phone           VARCHAR(20)
);

CREATE TABLE Recruiter (
    recruiterID     INTEGER PRIMARY KEY,
    companyID       INTEGER NOT NULL,
    first_name      VARCHAR(30),
    last_name       VARCHAR(30),
    email           VARCHAR(100) NOT NULL UNIQUE,
    phone           VARCHAR(20),
    FOREIGN KEY (companyID) REFERENCES Company (companyID)
);

CREATE TABLE Job (
    jobID           INTEGER PRIMARY KEY,
    companyID       INTEGER NOT NULL,
    recruiterID     INTEGER NOT NULL,
    salary          FLOAT,
    workingHours    FLOAT,
    datePosted      DATE,
    location        VARCHAR(200),
    title           VARCHAR(100) NOT NULL,
    description     VARCHAR(500) NOT NULL,
    FOREIGN KEY (companyID) REFERENCES Company (companyID),
    FOREIGN KEY (recruiterID) REFERENCES Recruiter (recruiterID)
);
```

```sql
CREATE TABLE JobApplicant (
    applicantID     INTEGER PRIMARY KEY,
    first_name      VARCHAR(30) NOT NULL,
    last_name       VARCHAR(30),
    industry        VARCHAR(100),
    birthdate       DATE,
    address         VARCHAR(200),
    email           VARCHAR(100) NOT NULL UNIQUE,
    phone           VARCHAR(20)
);

CREATE TABLE JobApplication (
    jobAppID        INTEGER PRIMARY KEY,
    jobID           INTEGER NOT NULL,
    applicantID     INTEGER NOT NULL,
    dateTime        DATE,
    status          VARCHAR(20),
    FOREIGN KEY (jobID) REFERENCES Job (jobID),
    FOREIGN KEY (applicantID) REFERENCES JobApplicant (applicantID)
);

CREATE TABLE Resume (
    resumeID        INTEGER PRIMARY KEY,
    applicantID     INTEGER NOT NULL,
    upload_file     BLOB NOT NULL,
    uploadDate      DATE DEFAULT SYSDATE NOT NULL,
    FOREIGN KEY (applicantID) REFERENCES JobApplicant (applicantID)

);

CREATE TABLE Interview (
    interviewID     INTEGER PRIMARY KEY,
    jobAppID        INTEGER NOT NULL,
    dateTime        DATE DEFAULT SYSDATE NOT NULL,
    location        VARCHAR(100) NOT NULL,
    FOREIGN KEY (jobAppID) REFERENCES JobApplication (jobAppID)
);
```

Below are some screenshots showing our insert statements for each of our tables in SQL. These data values were used as the testbed for several of our queries, and are shown in the output of those test queries.

```
-- Inserting Data Values into Tables
-- Adds sample companies with their companyID, name, industry, location, email, and phone
INSERT INTO Company VALUES (1, 'Apple Canada', 'Software', '120 Bremner Boulevard Suite 1600, Toronto, Ontario, M5J 0A8', '', '647-943-4400');
INSERT INTO Company VALUES (2, 'Royal Bank of Canada (RBC)', 'Banking', 'Toronto, Ontario, Canada', 'recruitment@rbc.com', '1-800-769-2511');
INSERT INTO Company VALUES (3, 'AMD', 'Technology', 'Markham, Ontario, Canada', '', '905-882-2600');
INSERT INTO Company VALUES (4, 'SAMSUNG', 'Hardware', 'Vancouver, British Columbia', 'recruitment@samsung.com', '416-230-8121');

-- Adds sample recruiters linked to companies with their recruiterID, respective companyID (company that recruiter works for), first and last names, email, and phone.
INSERT INTO Recruiter VALUES (1, 1, 'Jane', 'Doe', 'jane.doe@apple.com', '647-222-3333');
INSERT INTO Recruiter VALUES (2, 2, 'Bob', 'William', 'bob.william@rbc.com', '416-111-1111');
INSERT INTO Recruiter VALUES (3, 3, 'John', 'Daniels', 'john.daniels@amd.com', '905-423-5678');
INSERT INTO Recruiter VALUES (4, 4, 'Jack', 'Jones', 'jack.jones@samsung.com', '647-333-4444');

-- Adds sample jobs posted by recruiters with jobID, respective recruiterID (recruiter who posted job), respective companyID (company that the job is for), salary, hours, date posted, location, title, and description
INSERT INTO Job VALUES (1, 1, 1, 'Apple Canada', 31.50, 36.25, TO_DATE('2025-09-22', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Software Engineer', 'Develop and maintain Apple software products.');
INSERT INTO Job VALUES (2, 2, 2, 'Royal Bank of Canada (RBC)', 24.50, 35.00, TO_DATE('2025-09-24', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Financial Analyst', 'Analyze financial data and provide insights for RBC clients.');
INSERT INTO Job VALUES (3, 3, 3, 'AMD', 28.50, 40.00, TO_DATE('2025-09-26', 'YYYY-MM-DD'), 'Markham, Ontario, Canada', 'Systems Design Engineer', 'Responsible for designing, integrating, and validating complex hardware and software systems.');
INSERT INTO Job VALUES (4, 4, 4, 'Samsung', 25, 42, DATE '2025-09-28', 'Vancouver, British Columbia, Canada', 'Software Developer', 'Develop and maintain applications for Samsung.');
INSERT INTO Job VALUES (5, 1, 1, 'Apple Canada', 31.50, 36.25, TO_DATE('2025-09-22', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Mobile Developer', 'Develop and maintain Apple mobile products and services.');

-- Adds sample job applicants with their first and last names, industry, birthdate, address, email, and phone.
INSERT INTO JobApplicant VALUES (1, 'Alice', 'Bob', 'Technology', TO_DATE('2002-04-13', 'YYYY-MM-DD'), '123 Bay Street, Toronto, Ontario, Canada, M1B 2K3', 'alice.bob@gmail.com', '416-123-455');
INSERT INTO JobApplicant VALUES (2, 'Jake', 'Blake', 'Technology', TO_DATE('2004-05-06', 'YYYY-MM-DD'), '456 Main Street, Markham, Ontario, Canada, L6H 1F3', 'jake.blake@hotmail.com', '647-444-1947');
INSERT INTO JobApplicant VALUES (3, 'Griffin', 'Walker', 'Banking', TO_DATE('2003-12-25', 'YYYY-MM-DD'), '789 Bond Avenue, Ajax, Ontario, Canada, L0H 1H9', 'griffin.walker@outlook.com', '905-289-9876');
INSERT INTO JobApplicant VALUES (4, 'Ed', 'Stephens', 'Software', TO_DATE('2004-01-25', 'YYYY-MM-DD'), '145 Bloor Avenue, Toronto, Ontario, Canada, M5B 3K9', 'ed.stephens@outlook.com', '905-444-2121');
INSERT INTO JobApplicant VALUES (5, 'Joe', 'Random', 'Technology', TO_DATE('2001-07-29', 'YYYY-MM-DD'), '7622 Markham Road, Markham, Ontario, Canada, L6H 9A3', 'joe.random@gmail.com', '647-543-2211');
INSERT INTO JobApplicant VALUES (6, 'Michael', 'Jordan', 'Hardware', TO_DATE('2002-11-09', 'YYYY-MM-DD'), '116 Bond Street, Hamilton, Ontario, Canada, L0P 1B9', 'michael.jordan@gmail.com', '416-989-7777');

-- Adds sample applications linking applicants (respective applicantID) to jobs (respective jobID) with jobAppID, status, and date.
INSERT INTO JobApplication VALUES (1, 1, 1, TO_DATE('2025-09-28', 'YYYY-MM-DD'), 'Rejected');
INSERT INTO JobApplication VALUES (2, 2, 2, TO_DATE('2025-09-29', 'YYYY-MM-DD'), 'Under Review');
INSERT INTO JobApplication VALUES (3, 3, 3, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Submitted');
INSERT INTO JobApplication VALUES (4, 1, 4, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview Pending');
INSERT INTO JobApplication VALUES (5, 3, 5, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview Pending');
INSERT INTO JobApplication VALUES (6, 4, 6, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview Pending');
```

```
-- Adds sample applications linking applicants (respective applicantID) to jobs (respective jobID) with jobAppID, status, and date.
INSERT INTO JobApplication VALUES (1, 1, 1, TO_DATE('2025-09-28', 'YYYY-MM-DD'), 'Rejected');
INSERT INTO JobApplication VALUES (2, 2, 2, TO_DATE('2025-09-29', 'YYYY-MM-DD'), 'Under Review');
INSERT INTO JobApplication VALUES (3, 3, 3, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Submitted');
INSERT INTO JobApplication VALUES (4, 1, 4, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview Pending');
INSERT INTO JobApplication VALUES (5, 3, 5, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview Pending');
INSERT INTO JobApplication VALUES (6, 4, 6, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview Pending');

-- Adds sample resumes for applicants with resumeID, respective applicantID (each resume belongs to an job applicant), upload_file (resume file), and uploadDate (date when resume was uploaded).
INSERT INTO Resume VALUES (1, 1, UTL_RAW.CAST_TO_RAW('Alice Bob'), TO_DATE('2002-04-13', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (2, 2, UTL_RAW.CAST_TO_RAW('Jake Blake Resume'), TO_DATE('2025-09-28', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (3, 3, UTL_RAW.CAST_TO_RAW('Griffin Walker Resume'), TO_DATE('2025-09-29', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (4, 4, UTL_RAW.CAST_TO_RAW('Ed Stephens Resume'), TO_DATE('2025-09-28', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (5, 5, UTL_RAW.CAST_TO_RAW('Joe Random Resume'), TO_DATE('2025-09-29', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (6, 6, UTL_RAW.CAST_TO_RAW('Michael Jordan Resume'), TO_DATE('2025-09-30', 'YYYY-MM-DD'));

-- Adds sample interviews linked to job applications (includes respective jobAppID) with interviewID, date, time, and location.
INSERT INTO Interview VALUES (1, 4, TO_DATE('2025-10-1 10:00 AM', 'YYYY-MM-DD HH:MI AM'), 'Toronto, Ontario, Canada');
INSERT INTO Interview VALUES (2, 5, TO_DATE('2025-10-2 11:00 AM', 'YYYY-MM-DD HH:MI AM'), 'Markham, Ontario, Canada');
INSERT INTO Interview VALUES (3, 6, TO_DATE('2025-10-3 1:00 PM', 'YYYY-MM-DD HH:MI PM'), 'Vancouver, British Columbia, Canada');
```

Some additional notes about the insert statements:

- TO_DATE() is used to convert a text string into a DATE data type
- Since BLOB data type is used for upload_file attribute in Resume table, UTL_RAW.CAST_TO_RAW is used for text since we are not uploading an actual file.

Below are some screenshots showing our tables after running our insert statements.

Company

	COMPANYID	NAME	INDUSTRY	LOCATION	EMAIL	PHONE
1	1	Apple Canada	Software	120 Bremner Boulevard Suite 1600, Toronto, Ontario, M5J 0A8	(null)	647-943-4400
2	2	Royal Bank of Canada (RBC)	Banking	Toronto, Ontario, Canada	recruitment@rbc.com	1-800-769-2511
3	3	AMD	Technology	Markham, Ontario, Canada	(null)	905-882-2600
4	4	SAMSUNG	Hardware	Vancouver, British Columbia	recruitment@samsung.com	416-230-8121

Recruiter

	RECRUITERID	COMPANYID	FIRST_NAME	LAST_NAME	EMAIL	PHONE
1	1	1	Jane	Doe	jane.doe@apple.com	647-222-3333
2	2	2	Bob	William	bob.william@rbc.com	416-111-1111
3	3	3	John	Daniels	john.daniels@amd.com	905-423-5678
4	4	4	Jack	Jones	jack.jones@samsung.com	647-333-4444

Job

	JOBID	COMPANYID	RECRUITERID	EMPLOYER	SALARY	WORKINGHOURS	DATEPOSTED	LOCATION	TITLE	DESCRIPTION
1	1	1	1	Apple Canada	31.5	36.25	22-SEP-25	Toronto, Ontario, Canada	Software Engineer	Develop and maintain Apple software products.
2	2	2	2	Royal Bank of Canada (RBC)	24.5	35	24-SEP-25	Toronto, Ontario, Canada	Financial Analyst	Analyze financial data and provide insights for RBC clients.
3	3	3	3	AMD	28.5	40	26-SEP-25	Markham, Ontario, Canada	Systems Design Engineer	Responsible for designing, integrating, and validating complex hardware and software systems
4	4	4	4	Samsung	25	42	28-SEP-25	Vancouver, British Columbia, Canada	Software Developer	Develop and maintain applications for Samsung.
5	5	1	1	Apple Canada	31.5	36.25	22-SEP-25	Toronto, Ontario, Canada	Mobile Developer	Develop and maintain Apple mobile products and services.

JobApplicant

	APPLICANTID	FIRST_NAME	LAST_NAME	INDUSTRY	BIRTHDATE	ADDRESS	EMAIL	PHONE
1	1	Alice	Bob	Technology	13-APR-02	123 Bay Street, Toronto, Ontario, Canada, M1B 2K3	alice.bob@gmail.com	416-123-455
2	2	Jake	Blake	Technology	06-MAY-04	456 Main Street, Markham, Ontario, Canada, L6H 1F3	jake.blake@hotmail.com	647-444-1947
3	3	Griffin	Walker	Banking	25-DEC-03	789 Bond Avenue, Ajax, Ontario, Canada, L0H 1H9	griffin.walker@outlook.com	905-289-9876
4	4	Ed	Stephens	Software	25-JAN-04	145 Bloor Avenue, Toronto, Ontario, Canada, M5B 3K9	ed.stephens@outlook.com	905-444-2121
5	5	Joe	Random	Technology	29-JUL-01	7622 Markham Road, Markham, Ontario, Canada, L6H 9A3	joe.random@gmail.com	647-543-2211
6	6	Michael	Jordan	Hardware	09-NOV-02	116 Bond Street, Hamilton, Ontario, Canada, L0P 1B9	michael.jordan@gmail.com	416-989-7777

JobApplication

	JOBAPPID	JOBID	APPLICANTID	DATETIME	STATUS
1	1	1	1	28-SEP-25	Rejected
2	2	2	2	29-SEP-25	Under Review
3	3	3	3	30-SEP-25	Submitted
4	4	1	4	30-SEP-25	Interview Pending
5	5	3	5	30-SEP-25	Interview Pending
6	6	4	6	30-SEP-25	Interview Pending

Resume

	RESUMEID	APPLICANTID	UPLOAD_FILE	UPLOADDATE
1	1	1	(BLOB)	13-APR-02
2	2	2	(BLOB)	28-SEP-25
3	3	3	(BLOB)	29-SEP-25
4	4	4	(BLOB)	28-SEP-25
5	5	5	(BLOB)	29-SEP-25
6	6	6	(BLOB)	30-SEP-25

Interview

	INTERVIEWID	JOBAPPID	DATETIME	LOCATION
1	1	4	01-OCT-25	Toronto, Ontario, Canada
2	2	5	02-OCT-25	Markham, Ontario, Canada
3	3	6	03-OCT-25	Vancouver, British Columbia, Canada

Simple Queries

Below are some screenshots of the simple queries we created and their results after running them.

Query that returns a distinct list of all the industries from the companies in the Company Table:

```
-- Simple queries
-- Query 1: Returns a distinct list of all the industries from the companies in the Company Table.
SELECT DISTINCT industry
FROM Company
WHERE industry IS NOT NULL
ORDER BY industry;
```

- Shows each unique industry once.
- Ensures the industry attribute is not a null value.

Output:

	INDUSTRY
1	Banking
2	Hardware
3	Technology

Query that lists all distinct emails of the recruiters from the Recruiter Table:

```
-- Query 2: Lists all distinct emails of the recruiters from the Recruiter table.
SELECT DISTINCT email
FROM Recruiter
ORDER BY email;
```

- Results are ordered by email address in alphabetical order (A to Z).

Output:

	EMAIL
1	bob.william@rbc.com
2	jack.jones@samsung.com
3	jane.doe@apple.com
4	john.daniels@amd.com

Query to list the number of jobs posted by each company in the Job Table:

```
-- Query 3: Counts the number of jobs posted by each company in the Jobs table.
SELECT companyID, COUNT(jobID) AS job_count, employer
FROM Job
GROUP BY companyID, employer
ORDER BY job_count DESC, companyID ASC;
```

- Job_count contains the number of jobs in each company.
- COUNT() is used to tally the number of jobs for each company.
- GROUP BY ensures that the count is calculated separately for each combination of company and employer.
- ORDER BY job_count DESC means to sort the companies from the highest number of jobs to the lowest number of jobs
    - The company with the most jobs appears first, and the one with the fewest jobs appears last.
- If two companies have the same number of jobs posted, then they are sorted numerically by companyID.

Output:

	COMPANYID	JOB_COUNT	EMPLOYER
1	1	2	Apple Canada
2	2	1	Royal Bank of Canada (RBC)
3	3	1	AMD
4	4	1	Samsung

Query that finds the youngest job applicant in the table of JobApplicant:

```
-- Query 4: Finds the youngest applicant in the table of Job Applicants.
SELECT DISTINCT first_name, last_name, birthdate
FROM JobApplicant
WHERE last_name IS NOT NULL AND birthdate IS NOT NULL
ORDER BY birthdate DESC;
```

- Ensures the last_name and birthdate attributes are not null values.
- Results are sorted from the earliest to the latest birthdate.

Output:

	FIRST_NAME	LAST_NAME	BIRTHDATE
1	Jake	Blake	06-MAY-04
2	Ed	Stephens	25-JAN-04
3	Griffin	Walker	25-DEC-03
4	Michael	Jordan	09-NOV-02
5	Alice	Bob	13-APR-02
6	Joe	Random	29-JUL-01

Query that counts the number of job applications from the JobApplications table that have been submitted to a Job posting:

```
-- Query 5: Counts the number of Job Applications from the Job Applications table that have been submitted to a Job posting.
SELECT jobID, COUNT(jobAppID) AS application_count
FROM JobApplication
GROUP BY jobID
ORDER BY application_count DESC, jobID ASC;
```

- COUNT() is used to tally the number of job applications for each job posting.
- GROUP BY ensures that the count is calculated separately for each job.
- ORDER BY application_count DESC means to sort the job applications from the highest number of job applications to the lowest number of job applications per job posting.
    - The job with the most job applications appears first, and the one with the fewest job applications appears last.
- If two jobs have the same number of job applications, then they are sorted numerically by jobID.

Output:

	JOBID	APPLICATION_COUNT
1	1	2
2	3	2
3	2	1
4	4	1

Query that lists all the job applicants who have uploaded a Resume, ordered by the upload date of the Resume:

```
-- Query 6: Lists all the applicants who have uploaded a resume, ordered by upload date of the resume
SELECT DISTINCT applicantID, uploadDate
FROM Resume
ORDER BY uploadDate DESC, applicantID ASC;
```

- ORDER BY uploadDate DESC means to sort the resumes from the latest upload date to the earliest upload date.
- If two job applicants have the same resume upload date, then it is sorted numerically by applicantID.

Output:

	APPLICANTID	UPLOADDATE
1	6	30-SEP-25
2	3	29-SEP-25
3	5	29-SEP-25
4	2	28-SEP-25
5	4	28-SEP-25
6	1	27-SEP-25

Query that counts the number of scheduled interviews per job application:

```
-- Query 7: Counts the number of scheduled interviews per job application.
SELECT jobAppID, COUNT(interviewID) AS interview_count
FROM Interview
GROUP BY jobAppID
ORDER BY interview_count;
```

- COUNT() is used to tally the number of interviews per job application
- GROUP BY ensures that the count is calculated separately for each job application.
- ORDER BY interview_count sorts the results from fewest to most interviews.

Output:

	JOBAPPID	INTERVIEW_COUNT
1	6	1
2	5	1
3	4	1

Query that lists all jobs from the Job table with ‘Software’ in the title of the job:

```
-- list all jobs with 'Software' in the title
SELECT *
FROM Job
WHERE Job.title LIKE 'Software%';
```

- Uses LIKE with 'Software%' to find titles starting with the word "Software"

Output:

	JOBID	COMPANYID	RECRUITERID	EMPLOYER	SALARY	WORKINGHOURS	DATEPOSTED	LOCATION	TITLE	DESCRIPTION
1	1	1	1	Apple Canada	31.5	36.25	22-SEP-25	Toronto, Ontario, Canada	Software Engineer	Develop and maintain Apple software products.
2	4	4	4	Samsung	25	42	28-SEP-25	Vancouver, British Columbia, Canada	Software Developer	Develop and maintain applications for Samsung.

Query that lists the number of job postings per company

```
-- list number of job postings per company
SELECT companyID, employer, COUNT(*) AS jobs_per_company
FROM Job
GROUP BY companyID, employer
ORDER BY companyID DESC;
```

- COUNT(*) counts how many jobs each company has posted
- GROUP BY groups results by companyID and employer
- ORDER BY companyID DESC shows companies in descending order by companyID

Output:

	COMPANYID	EMPLOYER	JOBS_PER_COMPANY
1	4	Samsung	1
2	3	AMD	1
3	2	Royal Bank of Canada (RBC)	1
4	1	Apple Canada	2

Query that lists the number of job applications per job applicant:

```
-- list number of applications per applicant
SELECT JobApplicant.applicantID, JobApplicant.first_name, JobApplicant.last_name, COUNT(*) AS applications_per_applicant
FROM JobApplication JOIN JobApplicant ON JobApplication.applicantID = JobApplicant.applicantID
GROUP BY JobApplicant.applicantID, JobApplicant.first_name, JobApplicant.last_name
ORDER BY applications_per_applicant DESC;
```

- JOIN links JobApplication with JobApplicant to connect applications to applicants.
- COUNT(*) counts how many applications each applicant has made.
- GROUP BY ensures counts are calculated separately per applicant.
- ORDER BY applications_per_applicant DESC shows applicants with the most applications first.

Output:

	APPLICANTID	FIRST_NAME	LAST_NAME	APPLICATIONS_PER_APPLICANT
1	3	Griffin	Walker	1
2	6	Michael	Jordan	1
3	5	Joe	Random	1
4	2	Jake	Blake	1
5	4	Ed	Stephens	1
6	1	Alice	Bob	1

Query that lists all of the resumes belonging to job applicant Jake Blake:

```
-- list all of Jake Blake's resumes
SELECT JobApplicant.first_name, JobApplicant.last_name, Resume.resumeID, Resume.upload_file, Resume.uploadDate
FROM Resume JOIN JobApplicant ON Resume.applicantID = JobApplicant.applicantID
WHERE JobApplicant.first_name = 'Jake' AND JobApplicant.last_name = 'Blake';
```

- JOIN links Resume with JobApplicant so we can filter resumes by applicant name
- WHERE ensures only resumes belonging to 'Jake Blake' are shown

Output:

	FIRST_NAME	LAST_NAME	RESUMEID	UPLOAD_FILE	UPLOADDATE
1	Jake	Blake	2	(BLOB)	28-SEP-25

Query that updates all job applications with ‘under review’ status to ‘rejected’ status:

```
-- update all 'under review' applications to 'rejected'
UPDATE JobApplication SET status = 'Rejected'
WHERE status = 'Under Review';
```

Output:

	JOBAPPID	JOBID	APPLICANTID	DATETIME	STATUS
1	1	1	1	28-SEP-25	Rejected
2	2	2	2	29-SEP-25	Rejected
3	3	3	3	30-SEP-25	Submitted
4	4	1	4	30-SEP-25	Interview Pending
5	5	3	5	30-SEP-25	Interview Pending
6	6	4	6	30-SEP-25	Interview Pending

Query that lists all recruiterIDs by the number of job postings per recruiterID:

```
-- list all recruiterIDs by number of job postings per ID
SELECT recruiterID, COUNT(*) as job_count
FROM Job
GROUP BY recruiterID
ORDER BY job_count DESC;
```

- COUNT(*) counts how many jobs each recruiter has posted
- GROUP BY groups results per recruiterID
- ORDER BY job_count DESC shows recruiters with the most job postings first

Output:

	RECRUITERID	JOB_COUNT
1	1	2
2	4	1
3	3	1
4	2	1

Query to update all interviews in the location of 'Ontario' to 'Virtual' location:

```
-- update all interviews in ontario to virtual interviews
UPDATE Interview SET location = 'Virtual'
WHERE location like '%Ontario%';
```

- WHERE uses LIKE '%Ontario%' to find interview locations that mention "Ontario"

Output:

	INTERVIEWID	JOBAPPID	DATETIME	LOCATION
1	1	4	01-OCT-25	Virtual
2	2	5	02-OCT-25	Virtual
3	3	6	03-OCT-25	Vancouver, British Columbia, Canada

Query that counts the number of applications that are under each status

```
SELECT Distinct location
FROM Job
ORDER BY location;
```

- DISTINCT is used to only show unique locations
- ORDER BY is used to order locations alphabetically.

Output

```
LOCATION
--------------------------------------------------
Markham, Ontario, Canada
Toronto, Ontario, Canada
Vancouver, British Columbia, Canada
```

Query that counts the number of applications that are under each status

```
SELECT status, COUNT(jobAppID) AS totalApplications
FROM JobApplication
GROUP BY status
ORDER BY totalApplications DESC;
```

- COUNT (JobAppID) counts the number of job applications in each status by counting the jobAppID
- GROUP BY is used to ensure that the count for each status is grouped separately
- ORDER BY DESC to order the status based on the count in descending order

Output

STATUS	TOTALAPPLICATIONS
Interview Pending	3
Rejected	1
Under Review	1
Submitted	1

Query that counts the number of companies that are operating in each industry

```
SELECT industry, COUNT(companyID) AS totalCompanies
FROM Company
GROUP BY industry
ORDER BY totalCompanies ASC;
```

- COUNT is used to count the number of companies in each industry using the companyID
- GROUP BY is used to ensure that the count for each industry is grouped separately
- ORDER BY ASCis used to order the industries by count in ascending order

Output

INDUSTRY	TOTALCOMPANIES
Hardware	1
Banking	1
Software	1
Technology	1

Query that counts the number of applicants interested in each industry

```
SELECT industry, COUNT(applicantID) AS totalApplicants
FROM JobApplicant
GROUP BY industry
ORDER BY totalApplicants DESC;
```

- COUNT is used to count the number of applicants that are interested in each industry using applicantID
- GROUP BY is used to ensure that the count for each industry is grouped separately
- ORDER BY DESC is used to order the industries in descending order

Output

INDUSTRY	TOTALAPPLICANTS
Technology	3
Banking	1
Software	1
Hardware	1

Query that lists all unique resume upload dates

```
SELECT DISTINCT uploadDate as upldDate
FROM Resume
ORDER BY upldDate;
```

- DISTINCT is used to get all distinct upload dates with no repeats
- ORDER BY is used to list the dates from oldest to newest

Output

UPLDDATE
02-04-13
25-09-28
25-09-29
25-09-30

Query that counts the number of interviews in each location

```
SELECT location, COUNT(interviewID) as totalInterviews
FROM Interview
GROUP BY location
ORDER BY totalInterviews ASC;
```

- COUNT is used to count the number of interviews for each location using interviewID
- GROUP BY is used to ensure that the count for each location is grouped separately
- ORDER BY ASC is used to order the locations by interview count in ascending order

Output

LOCATION	TOTALINTERVIEWS
Vancouver, British Columbia, Canada	1
Markham, Ontario, Canada	1
Toronto, Ontario, Canada	1

Query that counts the total number of recruiters working under each company by companyID

```
SELECT companyID, count(recruiterID) as totalRecruiters
FROM Recruiter
GROUP BY companyID
ORDER BY totalRecruiters DESC;
```

- COUNT is used to count the number of recruiters that work under each company using recruiterID
- GROUP BY is used to ensure that the count for each companyID is grouped separately
- ORDER BY DESC is used to order the companyIDs by their count in descending order

Output

```
 COMPANYID TOTALRECRUITERS
---------- ---------------
         1               1
         3               1
         4               1
         2               1
```

The following queries were created to update information for some of the attributes in the Company and Resume tables:

```
-- UPDATE, SET, WHERE queries: Used to update the attribute values of an existing row in a selected table.
-- Updates the location of the company whose companyID is 1 (Apple Canada).
UPDATE Company
SET location = 'Toronto, Ontario, Canada'
WHERE companyID = 1;

-- Updates the industry of the company whose companyID is 1 (Apple Canada).
UPDATE Company
SET industry = 'Technology'
WHERE companyID = 1;

-- Updates the uploadDate of the resume whose resumeID is 1.
UPDATE Resume
SET uploadDate = TO_DATE('2025-09-27', 'YYYY-MM-DD')
WHERE resumeID = 1;

-- Updates the upload_file of the resume whose resumeID is 1.
UPDATE Resume
SET upload_file = UTL_RAW.CAST_TO_RAW('Alice Bob Resume')
WHERE resumeID = 1;
```

The Company and Resume tables following the update:

Company

	COMPANYID	NAME	INDUSTRY	LOCATION	EMAIL	PHONE
1	1	Apple Canada	Technology	Toronto, Ontario, Canada	(null)	647-943-4400
2	2	Royal Bank of Canada (RBC)	Banking	Toronto, Ontario, Canada	recruitment@rbc.com	1-800-769-2511
3	3	AMD	Technology	Markham, Ontario, Canada	(null)	905-882-2600
4	4	SAMSUNG	Hardware	Vancouver, British Columbia	recruitment@samsung.com	416-230-8121

Resume

	RESUMEID	APPLICANTID	UPLOAD_FILE	UPLOADDATE
1	1	1	(BLOB)	27-SEP-25
2	2	2	(BLOB)	28-SEP-25
3	3	3	(BLOB)	29-SEP-25
4	4	4	(BLOB)	28-SEP-25
5	5	5	(BLOB)	29-SEP-25
6	6	6	(BLOB)	30-SEP-25

Advanced Join Queries

Below are some screenshots showing our advanced join queries and the results after running them.

Query that shows the job applicants who applied for jobs posted by the recruiter 'Bob William':

```
-- Job applicants who applied for jobs posted by the recrutier 'Bob William'
SELECT a.applicantID, a.first_name || ' ' || a.last_name AS applicant_name
FROM JobApplicant a, JobApplication ja, Job j, Recruiter r
WHERE r.first_name = 'Bob'
    AND r.last_name = 'William'
    AND r.recruiterID = j.recruiterID
    AND j.jobID = ja.jobID
    AND ja.applicantID = a.applicantID
ORDER BY a.applicantID asc;
```

Output:

	APPLICANTID	APPLICANT_NAME
1	2	Jake Blake

Query that shows the job applicants who applied for the 'Software Engineer' job at the company 'Apple Canada':

```
-- Job applicants who applied for the 'Software Engineer' job at the company 'Apple Canada'
SELECT a.applicantID, a.first_name || ' ' || a.last_name AS applicant_name
FROM JobApplicant a, JobApplication ja, Job j, Company c
WHERE j.title = 'Software Engineer'
    AND c.name = 'Apple Canada'
    AND j.companyID = c.companyID
    AND ja.jobID = j.jobID
    AND ja.applicantID = a.applicantID
ORDER BY a.applicantID asc;
```

Output:

	APPLICANTID	APPLICANT_NAME
1	1	Alice Bob
2	4	Ed Stephens

Query that shows the number of job applicants who have interviews for jobs posted by each recruiter:

```sql
-- Number of job applicants who have interviews for jobs posted by each recruiter
SELECT r.first_name || ' ' || r.last_name AS recruiter_name, COUNT(DISTINCT a.applicantID) AS applicant_count
FROM Interview i, JobApplication ja, Job j, Recruiter r, JobApplicant a
WHERE r.recruiterID = j.recruiterID
    AND j.jobID = ja.jobID
    AND ja.jobAppID = i.jobAppID
    AND ja.applicantID = a.applicantID
GROUP BY r.first_name, r.last_name
ORDER BY applicant_count DESC;
```

Output:

	RECRUITER_NAME	APPLICANT_COUNT
1	Jane Doe	1
2	John Daniels	1
3	Jack Jones	1

Note: for the next queries and views, the following lines were added to the database.

```sql
INSERT INTO Job VALUES (5, 1, 1, 'Apple Canada', 31.50, 36.25, TO_DATE('2025-09-22', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Mobile Developer', 'Develop and maintain Apple mobile products and services.');
INSERT INTO JobApplicant VALUES (7, 'Sam', 'Inactive', 'Technology', TO_DATE('2002-04-13', 'YYYY-MM-DD'), '123 Bay Street, Toronto, Ontario, Canada, M1B 2K3', 'sam.inactive@gmail.com', '416-123-8293');
INSERT INTO Resume VALUES (7, 6, UTL_RAW.CAST_TO_RAW('Michael Jordan Resume 2'), TO_DATE('2025-10-04', 'YYYY-MM-DD'));
INSERT INTO JobApplication VALUES (7, 4, 5, TO_DATE('2025-10-04', 'YYYY-MM-DD'), 'Interview Pending');
INSERT INTO Job VALUES (6, 1, 1, 'Apple Canada', 31.50, 36.25, TO_DATE('2025-10-04', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Janitor', 'Mop the floors and clean the bathrooms.');
```

Query that shows number of job postings per company with the company's name as well (and with two alternative notations)

```sql
-- list number of job postings per company (showing company name as well)
SELECT Company.companyID, Company.name, COUNT(jobID) as "Jobs Posted By Company"
FROM Job, Company
WHERE Job.companyID = Company.companyID
GROUP BY Company.companyID, Company.name
ORDER BY Company.companyID DESC;
-- alternative notation (we're just gonna use this because i dont like the ambiguity implicit joins)
SELECT Company.companyID, Company.name, COUNT(jobID) as "Jobs Posted By Company"
FROM Job JOIN Company ON Job.companyID = Company.companyID
GROUP BY Company.companyID, Company.name
ORDER BY Company.companyID DESC;
```

Output:

All rows fetched: 4 in 0.051 seconds

	COMPANYID	NAME	Jobs Posted By Company
1	4	SAMSUNG	1
2	3	AMD	1
3	2	Royal Bank of Canada (RBC)	1
4	1	Apple Canada	3

Query that shows applications to Apple Canada, and the status of those applications

```sql
-- list the job applicants, who have job applications to Apple Canada, and the status of those applications
SELECT JobApplicant.first_name, JobApplicant.last_name, JobApplication.status,  Job.title, Company.name, Company.companyID
FROM JobApplication JOIN Job ON JobApplication.jobID = Job.jobID
    JOIN Company on Job.companyID = Company.companyID
    JOIN JobApplicant ON JobApplication.applicantID = JobApplicant.applicantID
WHERE Company.name = 'Apple Canada';
```

Output:

	FIRST_NAME	LAST_NAME	STATUS	TITLE	NAME	COMPANYID
1	Alice	Bob	Rejected	Software Engineer	Apple Canada	1
2	Ed	Stephens	Interview Pending	Software Engineer	Apple Canada	1

Query that shows the resumes of those applicants who have pending interviews

```
-- list the (super awesome and effective) resumes from those who have a pending interview to a job
SELECT Resume.resumeID, Resume.upload_file, JobApplicant.first_name, JobApplicant.last_name, Job.title, Job.employer, JobApplication.status, Interview.dateTime, Interview.location
FROM Resume JOIN JobApplicant on Resume.applicantID = JobApplicant.applicantID
    JOIN JobApplication on JobApplicant.applicantID = JobApplication.applicantID
    JOIN Interview on JobApplication.jobAppID = Interview.jobAppID
    JOIN Job on JobApplication.jobID = Job.jobID
WHERE JobApplication.status = 'Interview Pending';
```

Output:

	RESUMEID	UPLOAD_FILE	FIRST_NAME	LAST_NAME	TITLE	EMPLOYER	STATUS	DATETIME	LOCATION
1	4	(BLOB)	Ed	Stephens	Software Engineer	Apple Canada	Interview Pending	01/10/25	Toronto, Ontario, Canada
2	5	(BLOB)	Joe	Random	Systems Design Engineer	AMD	Interview Pending	02/10/25	Markham, Ontario, Canada
3	6	(BLOB)	Michael	Jordan	Software Developer	Samsung	Interview Pending	03/10/25	Vancouver, British Columbia, Canada
4	7	(BLOB)	Michael	Jordan	Software Developer	Samsung	Interview Pending	03/10/25	Vancouver, British Columbia, Canada

Query that lists all job applicants who have interviews for jobs at Apple Canada

```
SELECT JobApplicant.applicantID, last_name, first_name
FROM JobApplicant, JobApplication, Interview, Company, Job
WHERE Company.name = 'Apple Canada' AND JobApplicant.applicantID = JobApplication.applicantID AND JobApplication.jobAppID = Interview.jobAppID
AND JobApplication.jobID = Job.jobID AND Job.companyID = Company.companyID
ORDER BY applicantID ASC;
```

Output:

	APPLICANTID	LAST_NAME	FIRST_NAME
1	4	Stephens	Ed

Query that lists all job applicants who applied for jobs posted by AMD

```
SELECT JobApplicant.applicantID, last_name, first_name
FROM JobApplicant, JobApplication, Job, Company
WHERE Company.name = 'AMD' AND Job.companyID = Company.companyID AND JobApplication.applicantID = JobApplicant.applicantID AND Job.jobID = JobApplication.jobID
ORDER BY jobAppID ASC;
```

Output

	APPLICANTID	LAST_NAME	FIRST_NAME
1	3	Walker	Griffin
2	5	Random	Joe

Query that lists jobIDs and shows which recruiter posted it, along with the recruiter's company name

```
SELECT Company.name as companyName, last_name, first_name, Job.jobID
FROM Recruiter, Job, Company
WHERE recruiter.companyID = Company.companyID AND Job.recruiterID = Recruiter.recruiterID
ORDER BY Company.name ASC;
```

Output

	COMPANYNAME	LAST_NAME	FIRST_NAME	JOBID
1	AMD	Daniels	John	3
2	Apple Canada	Doe	Jane	6
3	Apple Canada	Doe	Jane	5
4	Apple Canada	Doe	Jane	1
5	Royal Bank of Canada (RBC)	William	Bob	2
6	SAMSUNG	Jones	Jack	4

Views

Below are some screenshots showing our views and the results after running them.

ApplicantJobInfo - Shows each job applicant, along with the jobs they've applied for, including the company and recruiter:

```
-- View 1: ApplicantJobInfo - Shows each job applicant along with the jobs they've applied for, including the company and recruiter
CREATE VIEW ApplicantJobInfo AS
SELECT
    a.applicantID,
    a.first_name,
    a.last_name,
    j.title AS job_title,
    c.name AS company_name,
    r.first_name || ' ' || r.last_name AS recruiter_name
FROM JobApplicant a, JobApplication ja, Job j, Company c, Recruiter r
WHERE a.applicantID = ja.applicantID
    AND ja.jobID = j.jobID
    AND j.companyID = c.companyID
    AND j.recruiterID = r.recruiterID;

SELECT * FROM ApplicantJobInfo;
```

Output:

	APPLICANTID	FIRST_NAME	LAST_NAME	JOB_TITLE	COMPANY_NAME	RECRUITER_NAME
1	1	Alice	Bob	Software Engineer	Apple Canada	Jane Doe
2	2	Jake	Blake	Financial Analyst	Royal Bank of Canada (RBC)	Bob William
3	3	Griffin	Walker	Systems Design Engineer	AMD	John Daniels
4	4	Ed	Stephens	Software Engineer	Apple Canada	Jane Doe
5	5	Joe	Random	Systems Design Engineer	AMD	John Daniels
6	5	Joe	Random	Software Developer	SAMSUNG	Jack Jones
7	6	Michael	Jordan	Software Developer	SAMSUNG	Jack Jones

InterviewSchedules - Shows all scheduled interviews, including the job applicant details, the job title, recruiter, and location:

```
-- View 2: InterviewSchedules - Shows all scheduled interviews, including job applicant details, job title, recruiter, and location.
CREATE VIEW InterviewSchedules AS
SELECT
    i.interviewID,
    a.first_name || ' ' || a.last_name AS applicant_name,
    j.title AS job_title,
    r.first_name || ' ' || r.last_name AS recruiter_name,
    i.location
FROM Interview i, JobApplication ja, Job j, Recruiter r, JobApplicant a
WHERE i.jobAppID = ja.jobAppID
    AND ja.jobID = j.jobID
    AND ja.applicantID = a.applicantID
    AND j.recruiterID = r.recruiterID;

SELECT * FROM InterviewSchedules;
```

Output:

	INTERVIEWID	APPLICANT_NAME	JOB_TITLE	RECRUITER_NAME	LOCATION
1	1	Ed Stephens	Software Engineer	Jane Doe	Virtual
2	2	Joe Random	Systems Design Engineer	John Daniels	Virtual
3	3	Michael Jordan	Software Developer	Jack Jones	Vancouver, British Columbia, Canada

CompanyJobSummary - Summarizes how many job postings each company has and which recruiter manages them:

```sql
-- View 3: CompanyJobSummary - Summarizes how many job postings each company has and which recruiter manages them.
CREATE VIEW CompanyJobSummary AS
SELECT
    c.name AS company_name,
    r.first_name || ' ' || r.last_name AS recruiter_name,
    COUNT(j.jobID) AS total_jobs_posted
FROM Company c, Job j, Recruiter r
WHERE c.companyID = j.companyID
    AND j.recruiterID = r.recruiterID
GROUP BY c.name, r.first_name, r.last_name
ORDER BY c.name ASC;

SELECT * FROM CompanyJobSummary;
```

Output:

	COMPANY_NAME	RECRUITER_NAME	TOTAL_JOBS_POSTED
1	AMD	John Daniels	1
2	Apple Canada	Jane Doe	3
3	Royal Bank of Canada (RBC)	Bob William	1
4	SAMSUNG	Jack Jones	1

JobData - view of jobs with additional info applied (such as company name, applications received, etc.)

```sql
-- view of all jobs with additional data (such as application count, recruiter/company info, etc.)
CREATE or REPLACE VIEW JobData("Job ID", "Title", "Employer", "Employer ID", "Posted by", "Salary", "Working Hours", "Date Posted", "Location", "Description", "Applications Received") AS
    SELECT Job.jobID, Job.title, Job.employer, Company.companyID, Recruiter.first_name || ' ' || Recruiter.last_name, Job.salary, Job.workingHours, Job.datePosted, Job.location, Job.description,
        COUNT(JobApplication.jobAppID) AS "Applications Received"
        FROM Job LEFT JOIN JobApplication ON Job.jobID = JobApplication.jobID
            JOIN Company ON Job.companyID = Company.companyID
            JOIN Recruiter ON Job.recruiterID = Recruiter.recruiterID
        GROUP BY Job.jobID, Job.title, Job.employer, Company.companyID, Recruiter.first_name || ' ' || Recruiter.last_name, Job.salary, Job.workingHours, Job.datePosted, Job.location, Job.description
        ORDER BY Job.jobID;
```

Output:

	Job ID	Title	Employer	Employer ID	Posted by	Salary	Working Hours	Date Posted	Location	Description
1	1	Software Engineer	Apple Canada	1	Jane Doe	31.5	36.25	22/09/25	Toronto, Ontario, Canada	Develop and maintain Apple software products.
2	2	Financial Analyst	Royal Bank of Canada (RBC)	2	Bob William	24.5	35	24/09/25	Toronto, Ontario, Canada	Analyze financial data and provide insights for RBC clients.
3	3	Systems Design Engineer	AMD	3	John Daniels	28.5	40	26/09/25	Markham, Ontario, Canada	Responsible for designing, integrating, and validating complex hardware and software systems.
4	4	Software Developer	Samsung	4	Jack Jones	25	42	28/09/25	Vancouver, British Columbia, Canada	Develop and maintain applications for Samsung.
5	5	Mobile Developer	Apple Canada	1	Jane Doe	31.5	36.25	22/09/25	Toronto, Ontario, Canada	Develop and maintain Apple mobile products and services.
6	6	Janitor	Apple Canada	1	Jane Doe	31.5	36.25	04/10/25	Toronto, Ontario, Canada	Mop the floors and clean the bathrooms.

ActiveApplicants - applicants who have applied to at least one job and have at least one resume, with corresponding app and resume count

```sql
-- view of "active applicants" who have applied to at least one job and have at least one resume uploaded
CREATE or REPLACE VIEW ActiveApplicants ("Applicant ID", "First Name", "Last Name", "Address", "Email", "Phone", "Application Count", "Resume Count") AS
    SELECT JobApplicant.applicantID, JobApplicant.first_name, JobApplicant.last_name, JobApplicant.address, JobApplicant.email, JobApplicant.phone,
        COUNT(DISTINCT JobApplication.jobAppID) AS "Application Count",
        COUNT(DISTINCT Resume.resumeID) AS "Resume Count"
    FROM JobApplication JOIN JobApplicant ON JobApplicant.applicantID = JobApplication.applicantID
    JOIN Resume ON Resume.applicantID = JobApplicant.applicantID
    GROUP BY JobApplicant.applicantID, JobApplicant.first_name, JobApplicant.last_name, JobApplicant.address, JobApplicant.email, JobApplicant.phone
    HAVING COUNT(DISTINCT JobApplication.jobAppID) >= 1 AND COUNT(DISTINCT Resume.resumeID) >= 1
    ORDER BY JobApplicant.applicantID;
-- this will not include Sam Inactive, and will also show Michael Jordan having two resumes, and Joe Random having two applications
```

Output:

	Applicant ID	First Name	Last Name	Address	Email	Phone	Application Count	Resume Count
1	1	Alice	Bob	123 Bay Street, Toronto, Ontario, Canada, M1B 2K3	alice.bob@gmail.com	416-123-455	1	1
2	2	Jake	Blake	456 Main Street, Markham, Ontario, Canada, L6H 1F3	jake.blake@hotmail.com	647-444-1947	1	1
3	3	Griffin	Walker	789 Bond Avenue, Ajax, Ontario, Canada, L0H 1H9	griffin.walker@outlook.com	905-289-9876	1	1
4	4	Ed	Stephens	145 Bloor Avenue, Toronto, Ontario, Canada, M5B 3K9	ed.stephens@outlook.com	905-444-2121	1	1
5	5	Joe	Random	7622 Markham Road, Markham, Ontario, Canada, L6H 9A3	joe.random@gmail.com	647-543-2211	1	1
6	6	Michael	Jordan	116 Bond Street, Hamilton, Ontario, Canada, L0P 1B9	michael.jordan@gmail.com	416-989-7777	1	2

RecruiterData - view of recruiters with more info applied such as apps received, interviews, etc.

```sql
-- view of all recruiters with additional data (such as job posting count, application count, interview count, etc.)
-- "interviews scheduled" refers to the number of interviews scheduled for applications to jobs posted by that recruiter
CREATE or REPLACE VIEW RecruiterData ("Recruiter ID", "First Name", "Last Name", "Company ID", "Company Name", "Email", "Phone", "Jobs Posted", "Applications Received", "Interviews Scheduled") AS
    SELECT Recruiter.recruiterID, Recruiter.first_name, Recruiter.last_name, Company.companyID, Company.name, Recruiter.email, Recruiter.phone,
        COUNT(DISTINCT Job.jobID),
        COUNT(DISTINCT JobApplication.jobAppID),
        COUNT(DISTINCT Interview.interviewID)
        FROM Recruiter LEFT JOIN Job ON Recruiter.recruiterID = Job.recruiterID
            LEFT JOIN JobApplication on JobApplication.jobID = Job.jobID
            LEFT JOIN Interview on Interview.jobAppID = JobApplication.jobAppID
            JOIN Company on Recruiter.companyID = Company.companyID
        GROUP BY Recruiter.recruiterID, Recruiter.first_name, Recruiter.last_name, Company.companyID, Company.name, Recruiter.email, Recruiter.phone
        ORDER BY Recruiter.recruiterID;
```

Output:

Columns Data Grants Dependencies Details Triggers SQL Errors

+ Insert Export Delete Selected Commit Undo All

	Recruiter ID	First Name	Last Name	Company ID	Company Name	Email	Phone	Jobs Posted	Applications Received	Interviews Scheduled
1	1	Jane	Doe	1	Apple Canada	jane.doe@apple.com	647-222-3333	3	2	1
2	2	Bob	William	2	Royal Bank of Canada (RBC)	bob.william@rbc.com	416-111-1111	1	1	0
3	3	John	Daniels	3	AMD	john.daniels@amd.com	905-423-5678	1	2	1
4	4	Jack	Jones	4	SAMSUNG	jack.jones@samsung.com	647-333-4444	1	2	1

ApplicantInterview: View all applicants who have interviews and list the company they have the interview with and the date

```sql
CREATE VIEW ApplicantInterview AS
SELECT JobApplicant.applicantID, first_name, last_name, Company.name AS companyName, Interview.dateTime
FROM JobApplicant, JobApplication, Interview, Company, Job
WHERE JobApplicant.applicantID = JobApplication.applicantID AND JobApplication.jobAppID = Interview.jobAppID AND JobApplication.jobID = Job.jobID AND Job.companyID = Company.companyID;

SELECT * FROM ApplicantInterview;
```

Output:

	APPLICANTID	FIRST_NAME	LAST_NAME	COMPANYNAME	DATETIME
1	4	Ed	Stephens	Apple Canada	25-10-01
2	5	Joe	Random	AMD	25-10-02
3	6	Michael	Jordan	SAMSUNG	25-10-03
4	5	Joe	Random	SAMSUNG	25-10-06

JobApplicationDetails: View all applicants and the company they applied to, job position, and application status

```
CREATE VIEW JobApplicationDetails AS
SELECT JobApplication.jobAppID, JobApplicant.last_name, JobApplicant.first_name, Job.title, Company.name as companyName, JobApplication.status
FROM JobApplication, JobApplicant, Job, Company
WHERE JobApplication.applicantID = JobApplicant.applicantID AND JobApplication.jobID = Job.jobID AND Job.companyID = Company.companyID;

SELECT * FROM JobApplicationDetails;
```

Output:

	JOBAPPID	LAST_NAME	FIRST_NAME	TITLE	NAME	STATUS
1	1	Black	Jack	Software Engineer	Apple Canada	Rejected
2	2	Doe	John	Financial Analyst	Royal Bank of Canada (RBC)	Under Review
3	3	Walker	Griffin	Systems Design Engineer	AMD	Submitted
4	4	Stephens	Ed	Software Engineer	Apple Canada	Interview Pending
5	7	Random	Joe	Software Developer	SAMSUNG	Interview Pending
6	5	Random	Joe	Systems Design Engineer	AMD	Interview Pending
7	6	Jordan	Michael	Software Developer	SAMSUNG	Interview Pending

RecruiterInterview: View each recruiter, the company, and the interviews scheduled for jobs they manage

```
CREATE VIEW RecruiterInterview AS
SELECT Recruiter.recruiterID, Recruiter.last_name, Recruiter.first_name, Company.name AS companyName, Job.title, Interview.dateTime
FROM Recruiter, Company, Interview, JobApplication, Job
WHERE Recruiter.companyID = Company.companyID AND Job.recruiterID = Recruiter.recruiterID AND Job.jobID = JobApplication.jobID AND JobApplication.jobAppID = Interview.jobAppID;

SELECT * FROM RecruiterInterview;
```

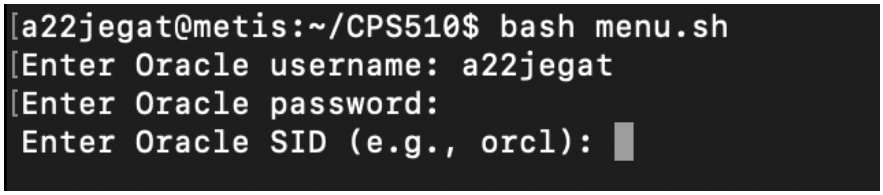
Output:

	RECRUITERID	LAST_NAME	FIRST_NAME	COMPANYNAME	TITLE	DATETIME
1	1	Doe	Jane	Apple Canada	Software Engineer	25-10-01
2	3	Daniels	John	AMD	Systems Design Engineer	25-10-02
3	4	Jones	Jack	SAMSUNG	Software Developer	25-10-03
4	4	Jones	Jack	SAMSUNG	Software Developer	25-10-06

Unix Shell Menu

We have included the code in our Unix Shell menu below, along with the screenshots of the results in the Unix Shell menu after running the commands and queries.

When you enter *bash menu.sh* to run the shell menu, you will be prompted to enter your username, password, and SID to use the database:



Once you have entered your username, password, and SID correctly, you will be directed to the shell menu.

menu.sh:

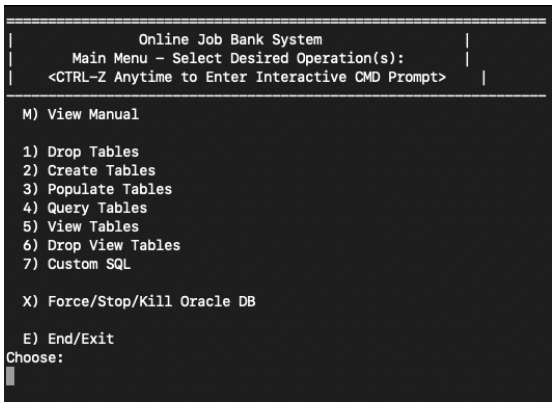
```
#!/bin/sh
while true; do
  read -p "Enter Oracle username: " USERNAME
  if [[ -n "${USERNAME//[[:space:]]/}" ]]; then
    break
  fi
  echo "Username cannot be empty. Please try again."
done

while true; do
  read -s -p "Enter Oracle password: " PASSWORD
  echo
  if [[ -n "${PASSWORD//[[:space:]]/}" ]]; then
    break
  fi
  echo "Password cannot be empty. Please try again."
done

while true; do
  read -p "Enter Oracle SID (e.g., orcl): " SID
  if [[ -n "${SID//[[:space:]]/}" ]]; then
    break
  fi
  echo "SID cannot be empty. Please try again."
done

# Build and export connection string
STR="${USERNAME}/${PASSWORD}@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.cs.torontomu.ca)
(Port=1521))(CONNECT_DATA=(SID=${SID})))"
export STR
MainMenu()
{
        while [ "$CHOICE" != "START" ]
        do
                clear
                echo "================================================================="
                echo "|               Online Job Bank System                          |"
                echo "|        Main Menu - Select Desired Operation(s):       |"
                echo "|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>    |"
```

```
        echo "------------------------------------------------------------------"
echo " $IS_SELECTEDM M) View Manual"
echo " "
echo " $IS_SELECTED1 1) Drop Tables"
echo " $IS_SELECTED2 2) Create Tables"
echo " $IS_SELECTED3 3) Populate Tables"
echo " $IS_SELECTED4 4) Query Tables"
 echo " $IS_SELECTED5 5) View Tables"
echo " $IS_SELECTED6 6) Drop View Tables"
echo " $IS_SELECTED7 7) Custom SQL"
echo " "
echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
echo " "
echo " $IS_SELECTEDE E) End/Exit"
echo "Choose: "

read CHOICE

if [ "$CHOICE" == "0" ]
then
        echo "Nothing Here"

elif [ "$CHOICE" == "1" ]
then
        bash drop_tables.sh
        echo "Press any key to continue"
          while true; do
                  read -rsn1 key  # Read a single character silently
                  if [[ -n "$key" ]]; then
                          echo "Key pressed. Continuing..."
                          break  # Exit the loop if a key is pressed
                  fi
          done

elif [ "$CHOICE" == "2" ]
then
        bash create_tables.sh
        echo "Press any key to continue"
          while true; do
                  read -rsn1 key  # Read a single character silently
                  if [[ -n "$key" ]]; then
                          echo "Key pressed. Continuing..."
                          break  # Exit the loop if a key is pressed
                  fi
          done

elif [ "$CHOICE" == "3" ]
then
        bash populate_tables.sh
        echo "Press any key to continue"
          while true; do
                  read -rsn1 key  # Read a single character silently
                  if [[ -n "$key" ]]; then
                          echo "Key pressed. Continuing..."
                          break  # Exit the loop if a key is pressed
                  fi
          done

elif [ "$CHOICE" == "4" ]
then
        bash queries.sh
        echo "Press any key to continue"
```

```
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done

        elif [ "$CHOICE" == "5" ]
        then
                bash views.sh
                echo "Press any key to continue"
                 while true; do
                         read -rsn1 key  # Read a single character silently
                         if [[ -n "$key" ]]; then
                                 echo "Key pressed. Continuing..."
                                 break  # Exit the loop if a key is pressed
                         fi
                 done

        elif [ "$CHOICE" == "6" ]
        then
                bash drop_views.sh
                echo "Press any key to continue"
                 while true; do
                         read -rsn1 key  # Read a single character silently
                         if [[ -n "$key" ]]; then
                                 echo "Key pressed. Continuing..."
                                 break  # Exit the loop if a key is pressed
                         fi
                 done

        elif [ "$CHOICE" == "7" ]
        then
                echo "Enter SQL statement. Press ENTER to run it."
                echo "To exit, enter M, then press ENTER."
                while true; do
                        # Prompting user for SQL statement
                        read -p "SQL> " sql_query

                        # Check if user entered M to exit
                        if [ "$sql_query" == "M" ]; then
                                echo "Returning to main menu..."
                                break
                        fi
                        #Execute SQL query
sqlplus64 -s "$STR"<<EOF
SET LINESIZE 200;
SET PAGESIZE 50;
SET FEEDBACK ON;
SET TRIMSPOOL ON;
SET WRAP OFF;
SET TAB OFF;
SET COLSEP ' | ';
$sql_query;
EXIT;
EOF

                done

        elif [ "$CHOICE" == "E" ]
        then
```

```
                exit
        fi
        done
}

#--COMMENTS BLOCK--
# Main Program
#--COMMENTS BLOCK--
ProgramStart()
{
        StartMessage
        while [ 1 ]
        do
                MainMenu
        done
}

ProgramStart
```

Output after running the command *bash menu.sh* and entering your credentials*:*

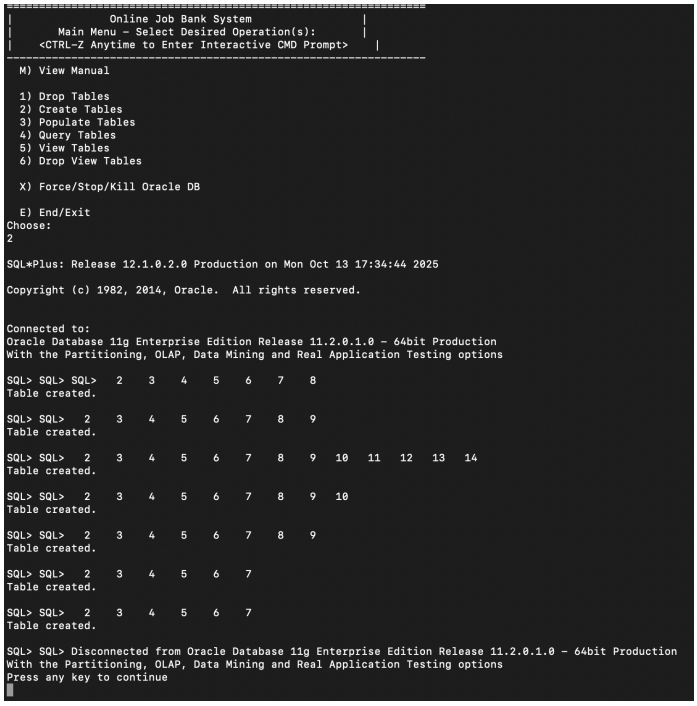


create_tables.sh:

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF

CREATE TABLE Company (
    companyID      INTEGER PRIMARY KEY,
    name           VARCHAR(30) NOT NULL UNIQUE,
    industry       VARCHAR(100),
    location       VARCHAR(200),
    email          VARCHAR(100),
    phone          VARCHAR(20)
);

CREATE TABLE Recruiter (
    recruiterID    INTEGER PRIMARY KEY,
    companyID      INTEGER NOT NULL,
    first_name     VARCHAR(30),
    last_name      VARCHAR(30),
```

```
    email          VARCHAR(100) NOT NULL UNIQUE,
    phone          VARCHAR(20),
    FOREIGN KEY (companyID) REFERENCES Company (companyID)
);

CREATE TABLE Job (
    jobID          INTEGER PRIMARY KEY,
    companyID      INTEGER NOT NULL,
    recruiterID    INTEGER NOT NULL,
    employer       VARCHAR(30) NOT NULL,
    salary         FLOAT,
    workingHours   FLOAT,
    datePosted     DATE,
    location       VARCHAR(200),
    title          VARCHAR(100) NOT NULL,
    description    VARCHAR(500) NOT NULL,
    FOREIGN KEY (companyID) REFERENCES Company (companyID),
    FOREIGN KEY (recruiterID) REFERENCES Recruiter (recruiterID)
);

CREATE TABLE JobApplicant (
    applicantID    INTEGER PRIMARY KEY,
    first_name     VARCHAR(30) NOT NULL,
    last_name      VARCHAR(30),
    industry       VARCHAR(100),
    birthdate      DATE,
    address        VARCHAR(200),
    email          VARCHAR(100) NOT NULL UNIQUE,
    phone          VARCHAR(20)
);

CREATE TABLE JobApplication (
    jobAppID       INTEGER PRIMARY KEY,
    jobID          INTEGER NOT NULL,
    applicantID    INTEGER NOT NULL,
    dateTime       DATE,
    status         VARCHAR(20),
    FOREIGN KEY (jobID) REFERENCES Job (jobID),
    FOREIGN KEY (applicantID) REFERENCES JobApplicant (applicantID)
);

CREATE TABLE Resume (
    resumeID       INTEGER PRIMARY KEY,
    applicantID    INTEGER NOT NULL,
    upload_file    BLOB NOT NULL,
    uploadDate     DATE DEFAULT SYSDATE NOT NULL,
    FOREIGN KEY (applicantID) REFERENCES JobApplicant (applicantID)
);

CREATE TABLE Interview (
    interviewID    INTEGER PRIMARY KEY,
    jobAppID       INTEGER NOT NULL,
    dateTime       DATE DEFAULT SYSDATE NOT NULL,
    location       VARCHAR(100) NOT NULL,
    FOREIGN KEY (jobAppID) REFERENCES JobApplication (jobAppID)
);

exit;
EOF
```

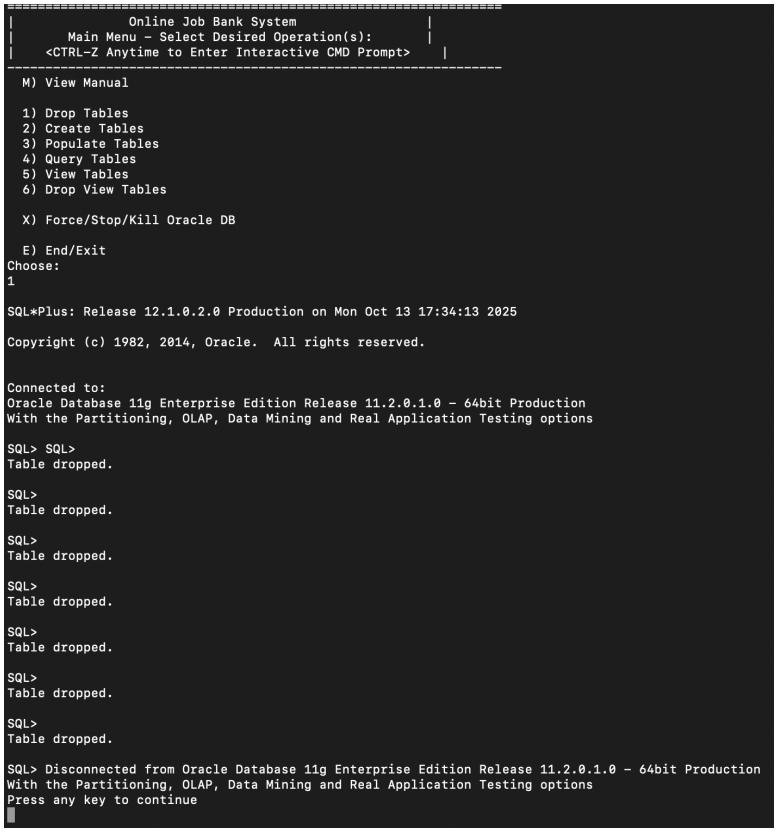
Output in menu.sh after running create_tables.sh command:



drop_tables.sh:

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF

DROP TABLE Interview CASCADE CONSTRAINTS;
DROP TABLE Resume CASCADE CONSTRAINTS;
DROP TABLE JobApplication CASCADE CONSTRAINTS;
DROP TABLE JobApplicant CASCADE CONSTRAINTS;
DROP TABLE Job CASCADE CONSTRAINTS;
DROP TABLE Recruiter CASCADE CONSTRAINTS;
DROP TABLE Company CASCADE CONSTRAINTS;
exit;
EOF
```

Output in menu.sh after running drop_tables.sh command:



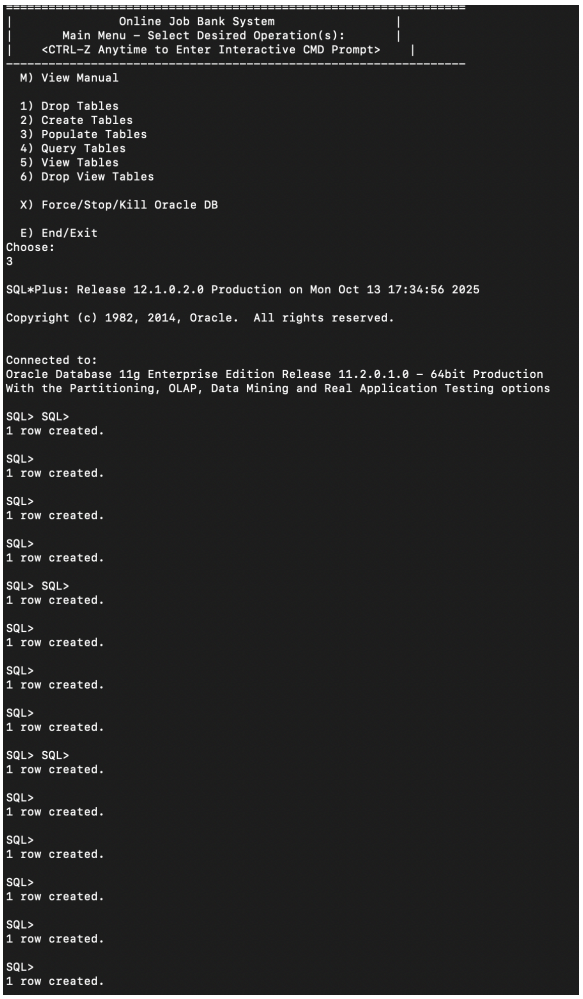
populate_tables.sh:

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF

INSERT INTO Company VALUES (1, 'Apple Canada', 'Software', '120 Bremner Boulevard Suite 1600,
Toronto, Ontario, M5J 0A8', '', '647-943-4400');
INSERT INTO Company VALUES (2, 'Royal Bank of Canada (RBC)', 'Banking', 'Toronto, Ontario,
Canada', 'recruitment@rbc.com', '1-800-769-2511');
INSERT INTO Company VALUES (3, 'AMD', 'Technology', 'Markham, Ontario, Canada', '',
'905-882-2600');
INSERT INTO Company VALUES (4, 'SAMSUNG', 'Hardware', 'Vancouver, British Columbia',
'recruitment@samsung.com', '416-230-8121');

INSERT INTO Recruiter VALUES (1, 1, 'Jane', 'Doe', 'jane.doe@apple.com', '647-222-3333');
INSERT INTO Recruiter VALUES (2, 2, 'Bob', 'William', 'bob.william@rbc.com', '416-111-1111');
INSERT INTO Recruiter VALUES (3, 3, 'John', 'Daniels', 'john.daniels@amd.com',
'905-423-5678');
INSERT INTO Recruiter VALUES (4, 4, 'Jack', 'Jones', 'jack.jones@samsung.com',
'647-333-4444');

INSERT INTO Job VALUES (1, 1, 1, 'Apple Canada', 31.50, 36.25, TO_DATE('2025-09-22',
'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Software Engineer', 'Develop and maintain Apple
software products.');
```

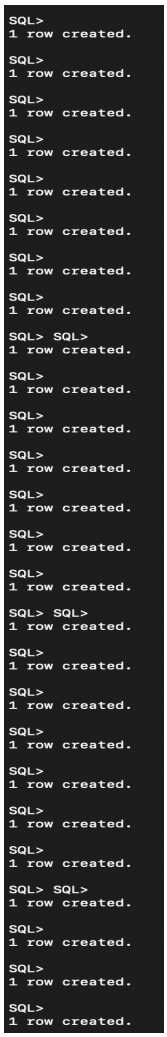
```
INSERT INTO Job VALUES (2, 2, 2, 'Royal Bank of Canada (RBC)', 24.50, 35.00,
TO_DATE('2025-09-24', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Financial Analyst', 'Analyze
financial data and provide insights for RBC clients.');
INSERT INTO Job VALUES (3, 3, 3, 'AMD', 28.50, 40.00, TO_DATE('2025-09-26', 'YYYY-MM-DD'),
'Markham, Ontario, Canada', 'Systems Design Engineer', 'Responsible for designing,
integrating, and validating complex hardware and software systems.');
INSERT INTO Job VALUES (4, 4, 4, 'Samsung', 25, 42, DATE '2025-09-28', 'Vancouver, British
Columbia, Canada', 'Software Developer', 'Develop and maintain applications for Samsung.');
INSERT INTO Job VALUES (5, 1, 1, 'Apple Canada', 31.50, 36.25, TO_DATE('2025-09-22',
'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Mobile Developer', 'Develop and maintain Apple
mobile products and services.');
INSERT INTO Job VALUES (6, 1, 1, 'Apple Canada', 31.50, 36.25, TO_DATE('2025-10-04',
'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Janitor', 'Mop the floors and clean the
bathrooms.');
INSERT INTO Job VALUES (7, 1, 1, 'Apple Canada', 40.75, 40.00, TO_DATE('2025-10-10',
'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Data Engineer', 'Design and maintain Apple's data
pipelines.');
INSERT INTO Job VALUES (8, 1, 1, 'Apple Canada', 29.00, 38.00, TO_DATE('2025-10-11',
'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'IT Support Specialist', 'Provide internal tech
support for Apple employees.');
INSERT INTO Job VALUES (9, 2, 2, 'Royal Bank of Canada (RBC)', 22.00, 35.00,
TO_DATE('2025-10-08', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Bank Teller', 'Assist
clients with daily transactions.');
INSERT INTO Job VALUES (10, 2, 2, 'Royal Bank of Canada (RBC)', 31.50, 37.50,
TO_DATE('2025-10-09', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Data Analyst', 'Analyze
customer trends to improve banking performance.');
INSERT INTO Job VALUES (11, 3, 3, 'AMD', 45.00, 40.00, TO_DATE('2025-10-06', 'YYYY-MM-DD'),
'Markham, Ontario, Canada', 'Hardware Engineer', 'Develop and test AMD hardware components.');
INSERT INTO Job VALUES (12, 3, 3, 'AMD', 35.00, 40.00, TO_DATE('2025-10-07', 'YYYY-MM-DD'),
'Markham, Ontario, Canada', 'Software Engineer', 'Develop internal tools and automation
frameworks for AMD.');
INSERT INTO Job VALUES (13, 4, 4, 'Samsung', 26.50, 40.00, TO_DATE('2025-10-05',
'YYYY-MM-DD'), 'Vancouver, British Columbia, Canada', 'QA Tester', 'Perform software and
hardware quality assurance tests.');
INSERT INTO Job VALUES (14, 4, 4, 'Samsung', 30.00, 40.00, TO_DATE('2025-10-06',
'YYYY-MM-DD'), 'Vancouver, British Columbia, Canada', 'UI/UX Designer', 'Design user
interfaces for Samsung applications.');

INSERT INTO JobApplicant VALUES (1, 'Alice', 'Bob', 'Technology', TO_DATE('2002-04-13',
'YYYY-MM-DD'), '123 Bay Street, Toronto, Ontario, Canada, M1B 2K3', 'alice.bob@gmail.com',
'416-123-455');
INSERT INTO JobApplicant VALUES (2, 'Jake', 'Blake', 'Technology', TO_DATE('2004-05-06',
'YYYY-MM-DD'), '456 Main Street, Markham, Ontario, Canada, L6H 1F3', 'jake.blake@hotmail.com',
'647-444-1947');
INSERT INTO JobApplicant VALUES (3, 'Griffin', 'Walker', 'Banking', TO_DATE('2003-12-25',
'YYYY-MM-DD'), '789 Bond Avenue, Ajax, Ontario, Canada, L0H 1H9',
'griffin.walker@outlook.com', '905-289-9876');
INSERT INTO JobApplicant VALUES (4, 'Ed', 'Stephens', 'Software', TO_DATE('2004-01-25',
'YYYY-MM-DD'), '145 Bloor Avenue, Toronto, Ontario, Canada, M5B 3K9',
'ed.stephens@outlook.com', '905-444-2121');
INSERT INTO JobApplicant VALUES (5, 'Joe', 'Random', 'Technology', TO_DATE('2001-07-29',
'YYYY-MM-DD'), '7622 Markham Road, Markham, Ontario, Canada, L6H 9A3', 'joe.random@gmail.com',
'647-543-2211');
INSERT INTO JobApplicant VALUES (6, 'Michael', 'Jordan', 'Hardware', TO_DATE('2002-11-09',
'YYYY-MM-DD'), '116 Bond Street, Hamilton, Ontario, Canada, LOP 1B9',
'michael.jordan@gmail.com', '416-989-7777');
INSERT INTO JobApplicant VALUES (7, 'Sam', 'Inactive', 'Technology', TO_DATE('2002-04-13',
'YYYY-MM-DD'), '123 Bay Street, Toronto, Ontario, Canada, M1B 2K3', 'sam.inactive@gmail.com',
'416-123-8293');

INSERT INTO JobApplication VALUES (1, 1, 1, TO_DATE('2025-09-28', 'YYYY-MM-DD'), 'Rejected');
```

```
INSERT INTO JobApplication VALUES (2, 2, 2, TO_DATE('2025-09-29', 'YYYY-MM-DD'), 'Under
Review');
INSERT INTO JobApplication VALUES (3, 3, 3, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Submitted');
INSERT INTO JobApplication VALUES (4, 1, 4, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview
Pending');
INSERT INTO JobApplication VALUES (5, 3, 5, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview
Pending');
INSERT INTO JobApplication VALUES (6, 4, 6, TO_DATE('2025-09-30', 'YYYY-MM-DD'), 'Interview
Pending');
INSERT INTO JobApplication VALUES (7, 4, 5, TO_DATE('2025-10-04', 'YYYY-MM-DD'), 'Interview
Pending');

INSERT INTO Resume VALUES (1, 1, UTL_RAW.CAST_TO_RAW('Alice Bob'), TO_DATE('2002-04-13',
'YYYY-MM-DD'));
INSERT INTO Resume VALUES (2, 2, UTL_RAW.CAST_TO_RAW('Jake Blake Resume'),
TO_DATE('2025-09-28', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (3, 3, UTL_RAW.CAST_TO_RAW('Griffin Walker Resume'),
TO_DATE('2025-09-29', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (4, 4, UTL_RAW.CAST_TO_RAW('Ed Stephens Resume'),
TO_DATE('2025-09-28', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (5, 5, UTL_RAW.CAST_TO_RAW('Joe Random Resume'),
TO_DATE('2025-09-29', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (6, 6, UTL_RAW.CAST_TO_RAW('Michael Jordan Resume'),
TO_DATE('2025-09-30', 'YYYY-MM-DD'));
INSERT INTO Resume VALUES (7, 6, UTL_RAW.CAST_TO_RAW('Michael Jordan Resume 2'),
TO_DATE('2025-10-04', 'YYYY-MM-DD'));

INSERT INTO Interview VALUES (1, 4, TO_DATE('2025-10-1 10:00 AM', 'YYYY-MM-DD HH:MI AM'),
'Toronto, Ontario, Canada');
INSERT INTO Interview VALUES (2, 5, TO_DATE('2025-10-2 11:00 AM', 'YYYY-MM-DD HH:MI AM'),
'Markham, Ontario, Canada');
INSERT INTO Interview VALUES (3, 6, TO_DATE('2025-10-3 1:00 PM', 'YYYY-MM-DD HH:MI PM'),
'Vancouver, British Columbia, Canada');

UPDATE Company
SET location = 'Toronto, Ontario, Canada'
WHERE companyID = 1;

UPDATE Company
SET industry = 'Technology'
WHERE companyID = 1;

UPDATE Resume
SET upload_file = UTL_RAW.CAST_TO_RAW('Alice Bob Resume')
WHERE resumeID = 1;

UPDATE Resume
SET uploadDate = TO_DATE('2025-09-27', 'YYYY-MM-DD')
WHERE resumeID = 1;

exit;
EOF
```

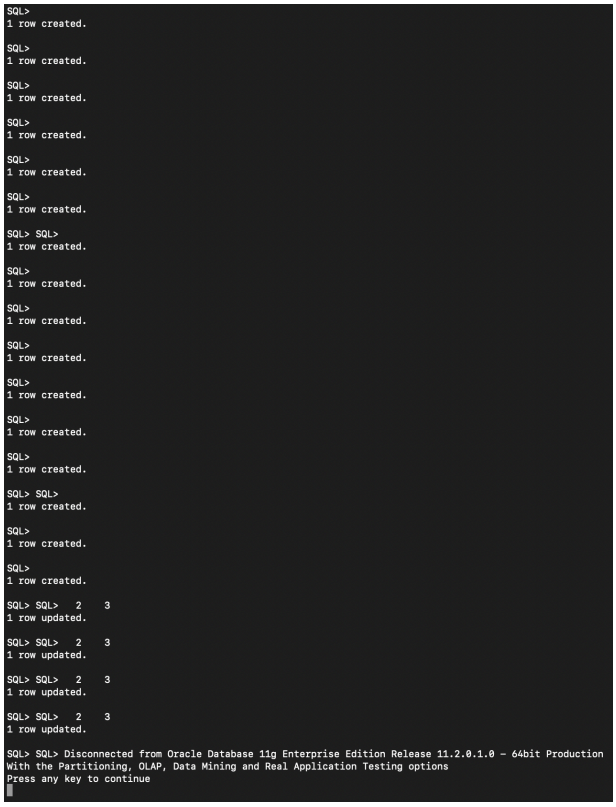
Output in menu.sh after running populate_tables.sh command:



```
================================================================
|              Online Job Bank System                 |
|       Main Menu - Select Desired Operation(s):      |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>    |
----------------------------------------------------------------
  M) View Manual

  1) Drop Tables
  2) Create Tables
  3) Populate Tables
  4) Query Tables
  5) View Tables
  6) Drop View Tables

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
3

SQL*Plus: Release 12.1.0.2.0 Production on Mon Oct 13 17:34:56 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.
```



```
SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.
```



queries.sh (contains queries from a4 and a5):

```
#!/bin/sh
MainMenu()
{
        while [ "$CHOICE" != "START" ]
        do
                clear
                echo "================================================================="
                echo "|                 Online Job Bank System                 |"
                echo "|         Query Menu - Select Desired Query(ies):        |"
                echo "|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>    |"
                echo "-----------------------------------------------------------------"
        echo " $IS_SELECTEDM M) View Manual"
        echo " "
        echo " $IS_SELECTED1 1) All Industries of Companies (a4_1)"
        echo " $IS_SELECTED2 2) All Emails of Recrutiers (a4_1)"
        echo " $IS_SELECTED3 3) Number of Jobs Posted by Each Company (a4_1)"
        echo " $IS_SELECTED4 4) Find Youngest Job Applicant (a4_1)"
        echo " $IS_SELECTED5 5) All Job Applicants That Have Uploaded a Resume (a4_1)"
        echo " $IS_SELECTED6 6) Number of Job Applications Submitted to a Job Posting (a4_1)"
        echo " $IS_SELECTED7 7) Scheduled Interviews (a4_1)"
        echo " $IS_SELECTED8 8) Job Applicants That Applied to Jobs Posted by Recruiter Bob
William (a4_2)"
        echo " $IS_SELECTED9 9) Job Applicants That Applied to the Software Engineer Job at
Apple Canada (a4_2)"
        echo " $IS_SELECTED10 10) Job Applicants With Interviews For Jobs Posted by Each
Recrutier (a4_2)"
        echo " $IS_SELECTED11 11) Job Applicants Who Applied to Apple Canada or AMD (a5)"
        echo " $IS_SELECTED12 12) Total Job Applications Per Company (a5)"
        echo " $IS_SELECTED13 13) Job Applicants With Interviews For Jobs Posted by Each
Recrutier (a5)"
```

```
        echo " $IS_SELECTED14 14) Job Applicants Who Applied to Apple Canada or AMD (a5)"
        echo " $IS_SELECTED15 15) Total Job Applications Per Company (a5)"
        echo " "
        echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
        echo " "
        echo " $IS_SELECTEDE E) End/Exit"
        echo "Choose: "

        read CHOICE

        if [ "$CHOICE" == "0" ]
        then
                echo "Nothing Here"

        elif [ "$CHOICE" == "1" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT DISTINCT industry
FROM Company
WHERE industry IS NOT NULL
ORDER BY industry;
EOF
               while true; do
                      read -rsn1 key  # Read a single character silently
                      if [[ -n "$key" ]]; then
                             echo "Key pressed. Continuing..."
                             break  # Exit the loop if a key is pressed
                      fi
               done

        elif [ "$CHOICE" == "2" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT DISTINCT email
FROM Recruiter
ORDER BY email;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done


        elif [ "$CHOICE" == "3" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
```

```
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT companyID, COUNT(jobID) AS job_count, employer
FROM Job
GROUP BY companyID, employer
ORDER BY job_count DESC, companyID ASC;
EOF
               while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done


        elif [ "$CHOICE" == "4" ]
        then
                echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT DISTINCT first_name, last_name, birthdate
FROM JobApplicant
WHERE last_name IS NOT NULL AND birthdate IS NOT NULL
ORDER BY birthdate DESC;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done


        elif [ "$CHOICE" == "5" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT jobID, COUNT(jobAppID) AS application_count
FROM JobApplication
GROUP BY jobID
ORDER BY application_count DESC, jobID ASC;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
```

```
                done

        elif [ "$CHOICE" == "6" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT DISTINCT applicantID, uploadDate
FROM Resume
ORDER BY uploadDate DESC, applicantID ASC;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done

        elif [ "$CHOICE" == "7" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT jobAppID, COUNT(interviewID) AS interview_count
FROM Interview
GROUP BY jobAppID
ORDER BY interview_count;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done

        elif [ "$CHOICE" == "8" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT a.applicantID, a.first_name || ' ' || a.last_name AS applicant_name
FROM JobApplicant a, JobApplication ja, Job j, Recruiter r
WHERE r.first_name = 'Bob'
 AND r.last_name = 'William'
 AND r.recruiterID = j.recruiterID
 AND j.jobID = ja.jobID
 AND ja.applicantID = a.applicantID
ORDER BY a.applicantID asc;
```

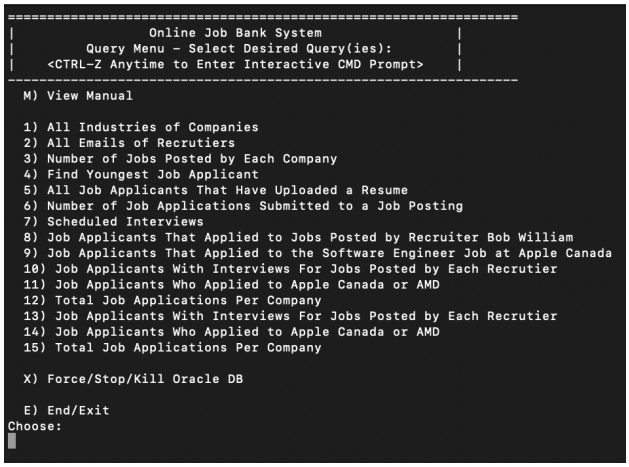
```
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done

        elif [ "$CHOICE" == "9" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT a.applicantID, a.first_name || ' ' || a.last_name AS applicant_name
FROM JobApplicant a, JobApplication ja, Job j, Company c
WHERE j.title = 'Software Engineer'
 AND c.name = 'Apple Canada'
 AND j.companyID = c.companyID
 AND ja.jobID = j.jobID
 AND ja.applicantID = a.applicantID
ORDER BY a.applicantID asc;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done


        elif [ "$CHOICE" == "10" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT r.first_name || ' ' || r.last_name AS recruiter_name, COUNT(DISTINCT a.applicantID) AS
applicant_count
FROM Interview i, JobApplication ja, Job j, Recruiter r, JobApplicant a
WHERE r.recruiterID = j.recruiterID
 AND j.jobID = ja.jobID
 AND ja.jobAppID = i.jobAppID
 AND ja.applicantID = a.applicantID
GROUP BY r.first_name, r.last_name
ORDER BY applicant_count DESC;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done
```

```
        elif [ "$CHOICE" == "11" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Job j ON ja.jobID = j.jobID
JOIN Company c ON j.companyID = c.companyID
WHERE c.name = 'Apple Canada'
UNION
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Job j ON ja.jobID = j.jobID
JOIN Company c ON j.companyID = c.companyID
WHERE c.name = 'AMD';
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done

        elif [ "$CHOICE" == "12" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT c.name AS CompanyName, COUNT(ja.jobAppID) AS TotalJobApplications
FROM Company c
JOIN Job j ON c.companyID = j.companyID
JOIN JobApplication ja ON j.jobID = ja.jobID
GROUP BY c.name
HAVING COUNT(ja.jobAppID) > (
 SELECT AVG(job_app_count)
 FROM (
  SELECT COUNT(ja2.jobAppID) AS job_app_count
  FROM JobApplication ja2
  JOIN Job j2 ON ja2.jobID = j2.jobID
  GROUP BY j2.companyID
 )
)
ORDER BY TotalJobApplications DESC;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
```

```
                        fi
                done

        elif [ "$CHOICE" == "13" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT DISTINCT c.name AS CompanyName
FROM Company c
WHERE EXISTS (
 SELECT 1
 FROM Job j
 JOIN JobApplication ja ON j.jobID = ja.jobID
 WHERE j.companyID = c.companyID
);
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done

        elif [ "$CHOICE" == "14" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
MINUS
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Interview i ON ja.jobAppID = i.jobAppID;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done

        elif [ "$CHOICE" == "15" ]
        then
               echo "Creating query..."
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi
```

```
sqlplus64 -s "$STR" <<EOF
SELECT
 c.name AS CompanyName,
 AVG(j.salary) AS AvgSalary,
 MIN(j.salary) AS MinSalary,
 MAX(j.salary) AS MaxSalary
FROM Company c
JOIN Job j ON c.companyID = j.companyID
GROUP BY c.name
ORDER BY AvgSalary DESC;
EOF
                while true; do
                        read -rsn1 key  # Read a single character silently
                        if [[ -n "$key" ]]; then
                                echo "Key pressed. Continuing..."
                                break  # Exit the loop if a key is pressed
                        fi
                done

        elif [ "$CHOICE" == "E" ]
        then
                exit
        fi
        done
}
MainMenu
```

Output in menu.sh after running queries.sh command (takes you to another menu to run queries):



Below are the outputs after running queries from a4_2 and a5. Note that the query numbers correspond to the query numbers listed on the shell menu.

Query 8 (a4_2):

```
============================================================
|                Online Job Bank System                    |
|         Query Menu - Select Desired Query(ies):          |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>       |
------------------------------------------------------------
  M) View Manual

  1) All Industries of Companies (a4_1)
  2) All Emails of Recrutiers (a4_1)
  3) Number of Jobs Posted by Each Company (a4_1)
  4) Find Youngest Job Applicant (a4_1)
  5) All Job Applicants That Have Uploaded a Resume (a4_1)
  6) Number of Job Applications Submitted to a Job Posting (a4_1)
  7) Scheduled Interviews (a4_1)
  8) Job Applicants That Applied to Jobs Posted by Recruiter Bob William (a4_2)
  9) Job Applicants That Applied to the Software Engineer Job at Apple Canada (a4_2)
  10) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a4_2)
  11) Job Applicants Who Applied to Apple Canada or AMD (a5)
  12) Total Job Applications Per Company (a5)
  13) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a5)
  14) Job Applicants Who Applied to Apple Canada or AMD (a5)
  15) Total Job Applications Per Company (a5)

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
8
Creating query...

APPLICANTID APPLICANT_NAME
----------- ------------------------------------------------------------
          2 Jake Blake
```

Query 9 (a4_2):

```
============================================================
|                Online Job Bank System                    |
|         Query Menu - Select Desired Query(ies):          |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>       |
------------------------------------------------------------
  M) View Manual

  1) All Industries of Companies (a4_1)
  2) All Emails of Recrutiers (a4_1)
  3) Number of Jobs Posted by Each Company (a4_1)
  4) Find Youngest Job Applicant (a4_1)
  5) All Job Applicants That Have Uploaded a Resume (a4_1)
  6) Number of Job Applications Submitted to a Job Posting (a4_1)
  7) Scheduled Interviews (a4_1)
  8) Job Applicants That Applied to Jobs Posted by Recruiter Bob William (a4_2)
  9) Job Applicants That Applied to the Software Engineer Job at Apple Canada (a4_2)
  10) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a4_2)
  11) Job Applicants Who Applied to Apple Canada or AMD (a5)
  12) Total Job Applications Per Company (a5)
  13) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a5)
  14) Job Applicants Who Applied to Apple Canada or AMD (a5)
  15) Total Job Applications Per Company (a5)

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
9
Creating query...

APPLICANTID APPLICANT_NAME
----------- ------------------------------------------------------------
          1 Alice Bob
          4 Ed Stephens
```

Query 10 (a4_2):

```
================================================================
|               Online Job Bank System                 |
|         Query Menu - Select Desired Query(ies):       |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>    |
----------------------------------------------------------------
  M) View Manual

  1) All Industries of Companies (a4_1)
  2) All Emails of Recrutiers (a4_1)
  3) Number of Jobs Posted by Each Company (a4_1)
  4) Find Youngest Job Applicant (a4_1)
  5) All Job Applicants That Have Uploaded a Resume (a4_1)
  6) Number of Job Applications Submitted to a Job Posting (a4_1)
  7) Scheduled Interviews (a4_1)
  8) Job Applicants That Applied to Jobs Posted by Recruiter Bob William (a4_2)
  9) Job Applicants That Applied to the Software Engineer Job at Apple Canada (a4_2)
  10) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a4_2)
  11) Job Applicants Who Applied to Apple Canada or AMD (a5)
  12) Total Job Applications Per Company (a5)
  13) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a5)
  14) Job Applicants Who Applied to Apple Canada or AMD (a5)
  15) Total Job Applications Per Company (a5)

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
10
Creating query...

RECRUITER_NAME                                             APPLICANT_COUNT
---------------------------------------------------------- ---------------
Jane Doe                                                                 1
John Daniels                                                             1
Jack Jones                                                               1
```

Query 11 (a5):

```
================================================================
|               Online Job Bank System                 |
|         Query Menu - Select Desired Query(ies):       |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>    |
----------------------------------------------------------------
  M) View Manual

  1) All Industries of Companies (a4_1)
  2) All Emails of Recrutiers (a4_1)
  3) Number of Jobs Posted by Each Company (a4_1)
  4) Find Youngest Job Applicant (a4_1)
  5) All Job Applicants That Have Uploaded a Resume (a4_1)
  6) Number of Job Applications Submitted to a Job Posting (a4_1)
  7) Scheduled Interviews (a4_1)
  8) Job Applicants That Applied to Jobs Posted by Recruiter Bob William (a4_2)
  9) Job Applicants That Applied to the Software Engineer Job at Apple Canada (a4_2)
  10) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a4_2)
  11) Job Applicants Who Applied to Apple Canada or AMD (a5)
  12) Total Job Applications Per Company (a5)
  13) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a5)
  14) Job Applicants Who Applied to Apple Canada or AMD (a5)
  15) Total Job Applications Per Company (a5)

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
11
Creating query...

APPLICANTID FIRST_NAME                     LAST_NAME
----------- ------------------------------ ------------------------------
          1 Alice                          Bob
          3 Griffin                        Walker
          4 Ed                             Stephens
          5 Joe                            Random
```

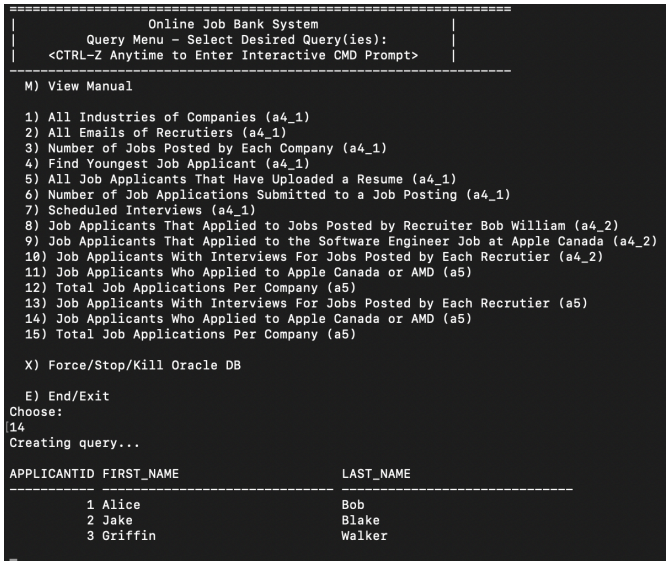
Query 12 (a5):

```
================================================================
|                Online Job Bank System                        |
|         Query Menu - Select Desired Query(ies):              |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>           |
----------------------------------------------------------------
  M) View Manual

  1) All Industries of Companies (a4_1)
  2) All Emails of Recrutiers (a4_1)
  3) Number of Jobs Posted by Each Company (a4_1)
  4) Find Youngest Job Applicant (a4_1)
  5) All Job Applicants That Have Uploaded a Resume (a4_1)
  6) Number of Job Applications Submitted to a Job Posting (a4_1)
  7) Scheduled Interviews (a4_1)
  8) Job Applicants That Applied to Jobs Posted by Recruiter Bob William (a4_2)
  9) Job Applicants That Applied to the Software Engineer Job at Apple Canada (a4_2)
  10) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a4_2)
  11) Job Applicants Who Applied to Apple Canada or AMD (a5)
  12) Total Job Applications Per Company (a5)
  13) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a5)
  14) Job Applicants Who Applied to Apple Canada or AMD (a5)
  15) Total Job Applications Per Company (a5)

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
12
Creating query...

COMPANYNAME                    TOTALJOBAPPLICATIONS
------------------------------ --------------------
Apple Canada                                      2
SAMSUNG                                           2
AMD                                               2
```

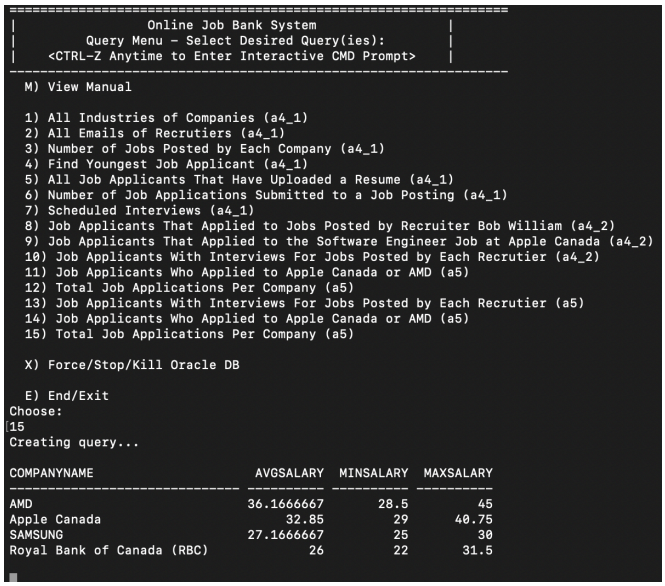
Query 13 (a5):

```
================================================================
|                Online Job Bank System                        |
|         Query Menu - Select Desired Query(ies):              |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>           |
----------------------------------------------------------------
  M) View Manual

  1) All Industries of Companies (a4_1)
  2) All Emails of Recrutiers (a4_1)
  3) Number of Jobs Posted by Each Company (a4_1)
  4) Find Youngest Job Applicant (a4_1)
  5) All Job Applicants That Have Uploaded a Resume (a4_1)
  6) Number of Job Applications Submitted to a Job Posting (a4_1)
  7) Scheduled Interviews (a4_1)
  8) Job Applicants That Applied to Jobs Posted by Recruiter Bob William (a4_2)
  9) Job Applicants That Applied to the Software Engineer Job at Apple Canada (a4_2)
  10) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a4_2)
  11) Job Applicants Who Applied to Apple Canada or AMD (a5)
  12) Total Job Applications Per Company (a5)
  13) Job Applicants With Interviews For Jobs Posted by Each Recrutier (a5)
  14) Job Applicants Who Applied to Apple Canada or AMD (a5)
  15) Total Job Applications Per Company (a5)

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
13
Creating query...

COMPANYNAME
------------------------------
Apple Canada
Royal Bank of Canada (RBC)
SAMSUNG
AMD
```

Query 14 (a5):



Query 15 (a5):

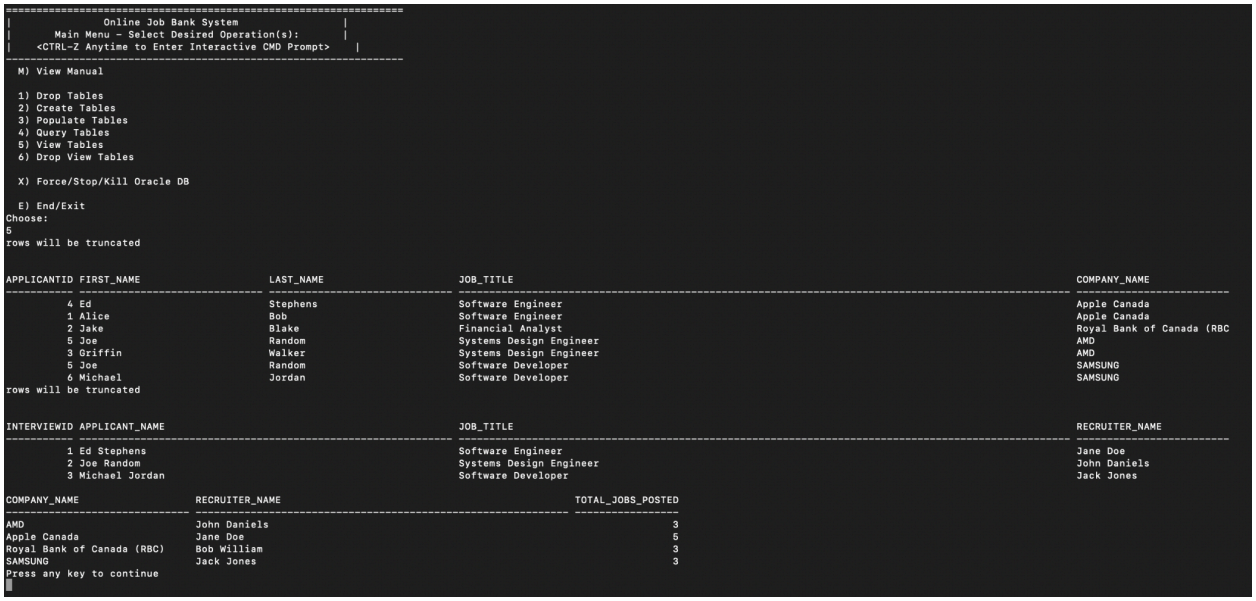


views.sh:

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF

SET PAGESIZE 100
SET LINESIZE 200
SET FEEDBACK OFF
SET HEADING ON
```

```
SET ECHO OFF
SET WRAP OFF
SET TRIMSPOOL ON

CREATE VIEW ApplicantJobInfo AS
SELECT
    a.applicantID,
    a.first_name,
    a.last_name,
    j.title AS job_title,
    c.name AS company_name,
    r.first_name || ' ' || r.last_name AS recruiter_name
FROM JobApplicant a, JobApplication ja, Job j, Company c, Recruiter r
WHERE a.applicantID = ja.applicantID
    AND ja.jobID = j.jobID
    AND j.companyID = c.companyID
    AND j.recruiterID = r.recruiterID;
SELECT * FROM ApplicantJobInfo;

CREATE VIEW InterviewSchedules AS
SELECT
    i.interviewID,
    a.first_name || ' ' || a.last_name AS applicant_name,
    j.title AS job_title,
    r.first_name || ' ' || r.last_name AS recruiter_name,
    i.location
FROM Interview i, JobApplication ja, Job j, Recruiter r, JobApplicant a
WHERE i.jobAppID = ja.jobAppID
    AND ja.jobID = j.jobID
    AND ja.applicantID = a.applicantID
    AND j.recruiterID = r.recruiterID;
SELECT * FROM InterviewSchedules;

CREATE VIEW CompanyJobSummary AS
SELECT
    c.name AS company_name,
    r.first_name || ' ' || r.last_name AS recruiter_name,
    COUNT(j.jobID) AS total_jobs_posted
FROM Company c, Job j, Recruiter r
WHERE c.companyID = j.companyID
    AND j.recruiterID = r.recruiterID
GROUP BY c.name, r.first_name, r.last_name
ORDER BY c.name ASC;
SELECT * FROM CompanyJobSummary;

exit;
EOF
```

Output in menu.sh after running views.sh command:

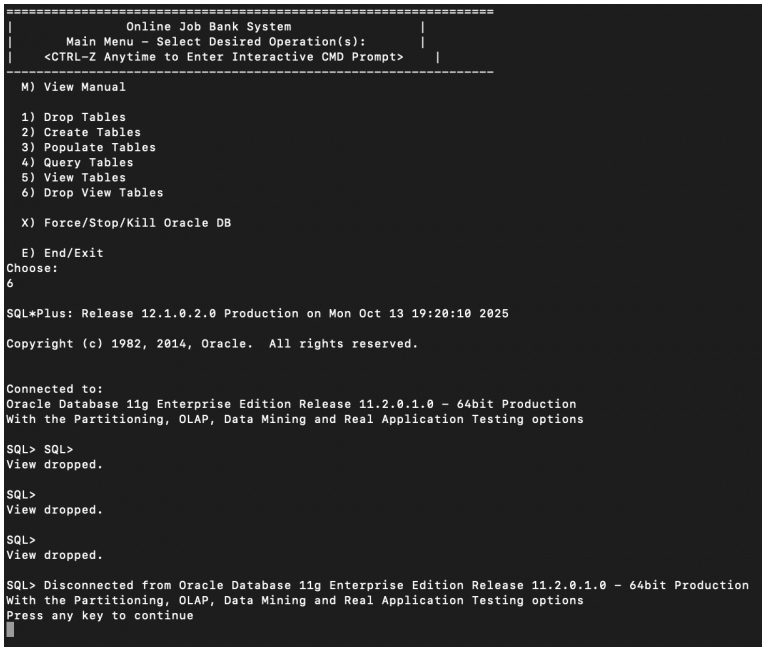


```
================================================================
|              Online Job Bank System                          |
|       Main Menu - Select Desired Operation(s):               |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>           |
----------------------------------------------------------------
  M) View Manual

  1) Drop Tables
  2) Create Tables
  3) Populate Tables
  4) Query Tables
  5) View Tables
  6) Drop View Tables

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
5
rows will be truncated

APPLICANTID FIRST_NAME                     LAST_NAME                     JOB_TITLE                                                                                                    COMPANY_NAME
----------- ------------------------------ ----------------------------- ------------------------------------------------------------------------------------------------------------ --------------------
          4 Ed                             Stephens                      Software Engineer                                                                                            Apple Canada
          1 Alice                          Bob                           Software Engineer                                                                                            Apple Canada
          2 Jake                           Blake                         Financial Analyst                                                                                            Royal Bank of Canada (RBC
          5 Joe                            Random                        Systems Design Engineer                                                                                      AMD
          3 Griffin                        Walker                        Systems Design Engineer                                                                                      AMD
          5 Joe                            Random                        Software Developer                                                                                           SAMSUNG
          6 Michael                        Jordan                        Software Developer                                                                                           SAMSUNG
rows will be truncated


INTERVIEWID APPLICANT_NAME                                               JOB_TITLE                                                                                                    RECRUITER_NAME
----------- ---------------------------------------------------------- ------------------------------------------------------------------------------------------------------------ --------------------
          1 Ed Stephens                                                  Software Engineer                                                                                            Jane Doe
          2 Joe Random                                                   Systems Design Engineer                                                                                      John Daniels
          3 Michael Jordan                                               Software Developer                                                                                           Jack Jones

COMPANY_NAME               RECRUITER_NAME                                                   TOTAL_JOBS_POSTED
-------------------------- ---------------------------------------------------------------- -----------------
AMD                        John Daniels                                                                     3
Apple Canada               Jane Doe                                                                         5
Royal Bank of Canada (RBC) Bob William                                                                      3
SAMSUNG                    Jack Jones                                                                       3
Press any key to continue
```

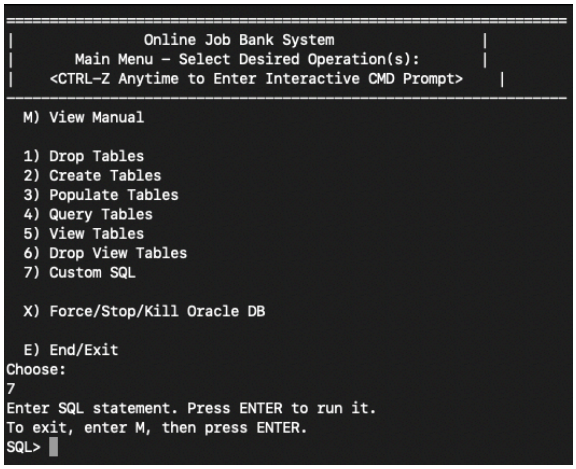
drop_views.sh:

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
if [ -z "$STR" ]; then
  echo "ERROR: STR not set. Run from menu.sh or set STR before calling this script."
  exit 1
fi

sqlplus64 -s "$STR" <<EOF

DROP VIEW ApplicantJobInfo;
DROP VIEW InterviewSchedules;
DROP VIEW CompanyJobSummary;
exit;
EOF
```

Output in menu.sh after running drop_views.sh command:



Running Custom SQL Queries:



E.g. Output after running SELECT * FROM Job query:

```
============================================================
|              Online Job Bank System                      |
|       Main Menu - Select Desired Operation(s):           |
|    <CTRL-Z Anytime to Enter Interactive CMD Prompt>     |
------------------------------------------------------------
  M) View Manual

  1) Drop Tables
  2) Create Tables
  3) Populate Tables
  4) Query Tables
  5) View Tables
  6) Drop View Tables
  7) Custom SQL

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
7
Enter SQL statement. Press ENTER to run it.
To exit, enter M, then press ENTER.
SQL> SELECT * FROM Job
rows will be truncated

rows will be truncated


     JOBID |  COMPANYID | RECRUITERID | EMPLOYER                       |     SALARY | WORKINGHOURS | DATEPOSTE | LOCATION
---------- | ---------- | ----------- | ------------------------------ | ---------- | ------------ | --------- | ----------------------------------------
         1 |          1 |           1 | Apple Canada                   |       31.5 |        36.25 | 22-SEP-25 | Toronto, Ontario, Canada
         2 |          2 |           2 | Royal Bank of Canada (RBC)     |       24.5 |           35 | 24-SEP-25 | Toronto, Ontario, Canada
         3 |          3 |           3 | AMD                            |       28.5 |           40 | 26-SEP-25 | Markham, Ontario, Canada
         4 |          4 |           4 | Samsung                        |         25 |           42 | 28-SEP-25 | Vancouver, British Columbia, Canada
         5 |          1 |           1 | Apple Canada                   |       31.5 |        36.25 | 22-SEP-25 | Toronto, Ontario, Canada
         6 |          1 |           1 | Apple Canada                   |       31.5 |        36.25 | 04-OCT-25 | Toronto, Ontario, Canada
         7 |          1 |           1 | Apple Canada                   |      40.75 |           40 | 10-OCT-25 | Toronto, Ontario, Canada
         8 |          1 |           1 | Apple Canada                   |         29 |           38 | 11-OCT-25 | Toronto, Ontario, Canada
         9 |          2 |           2 | Royal Bank of Canada (RBC)     |         22 |           35 | 08-OCT-25 | Toronto, Ontario, Canada
        10 |          2 |           2 | Royal Bank of Canada (RBC)     |       31.5 |         37.5 | 09-OCT-25 | Toronto, Ontario, Canada
        11 |          3 |           3 | AMD                            |         45 |           40 | 06-OCT-25 | Markham, Ontario, Canada
        12 |          3 |           3 | AMD                            |         35 |           40 | 07-OCT-25 | Markham, Ontario, Canada
        13 |          4 |           4 | Samsung                        |       26.5 |           40 | 05-OCT-25 | Vancouver, British Columbia, Canada
        14 |          4 |           4 | Samsung                        |         30 |           40 | 06-OCT-25 | Vancouver, British Columbia, Canada

14 rows selected.

SQL>
```

Advanced Queries

Note: for the next queries, the following lines were added to the database.

```
-- Insert Into Statements for Job Table
INSERT INTO Job VALUES (7, 1, 1, 'Apple Canada', 40.75, 40.00, TO_DATE('2025-10-10', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Data Engineer', 'Design and maintain Apple's data pipelines.');
INSERT INTO Job VALUES (8, 1, 1, 'Apple Canada', 29.00, 38.00, TO_DATE('2025-10-11', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'IT Support Specialist', 'Provide internal tech support for Apple employees.');
INSERT INTO Job VALUES (9, 2, 2, 'Royal Bank of Canada (RBC)', 22.00, 35.00, TO_DATE('2025-10-08', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Bank Teller', 'Assist clients with daily transactions.');
INSERT INTO Job VALUES (10, 2, 2, 'Royal Bank of Canada (RBC)', 31.50, 37.50, TO_DATE('2025-10-09', 'YYYY-MM-DD'), 'Toronto, Ontario, Canada', 'Data Analyst', 'Analyze customer trends to improve banking performance.');
INSERT INTO Job VALUES (11, 3, 3, 'AMD', 45.00, 40.00, TO_DATE('2025-10-06', 'YYYY-MM-DD'), 'Markham, Ontario, Canada', 'Hardware Engineer', 'Develop and test AMD hardware components.');
INSERT INTO Job VALUES (12, 3, 3, 'AMD', 35.00, 40.00, TO_DATE('2025-10-07', 'YYYY-MM-DD'), 'Markham, Ontario, Canada', 'Software Engineer', 'Develop internal tools and automation frameworks for AMD.');
INSERT INTO Job VALUES (13, 4, 4, 'Samsung', 26.50, 40.00, TO_DATE('2025-10-05', 'YYYY-MM-DD'), 'Vancouver, British Columbia, Canada', 'QA Tester', 'Perform software and hardware quality assurance tests.');
INSERT INTO Job VALUES (14, 4, 4, 'Samsung', 30.00, 40.00, TO_DATE('2025-10-06', 'YYYY-MM-DD'), 'Vancouver, British Columbia, Canada', 'UI/UX Designer', 'Design user interfaces for Samsung applications.');
```

Below are some screenshots showing our advanced queries and the results after running them.

Query that lists the job applicants who applied to either Apple Canada or AMD:

```
-- List of job applicants who applied to either Apple Canada or AMD - Combines two sets: Job applicants who applied to Apple Canada or AMD, uses UNION
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Job j ON ja.jobID = j.jobID
JOIN Company c ON j.companyID = c.companyID
WHERE c.name = 'Apple Canada'
UNION
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Job j ON ja.jobID = j.jobID
JOIN Company c ON j.companyID = c.companyID
WHERE c.name = 'AMD';
```

Output:

	APPLICANTID	FIRST_NAME	LAST_NAME
1	1	Alice	Bob
2	3	Griffin	Walker
3	4	Ed	Stephens
4	5	Joe	Random

Query that counts the total number of job applications per company and shows only those above average:

```
-- Count total job applications per company and show only those above average - uses GROUP BY, aggregate COUNT, and HAVING with a subquery
SELECT c.name AS CompanyName, COUNT(ja.jobAppID) AS TotalJobApplications
FROM Company c
JOIN Job j ON c.companyID = j.companyID
JOIN JobApplication ja ON j.jobID = ja.jobID
GROUP BY c.name
HAVING COUNT(ja.jobAppID) > (
    SELECT AVG(job_app_count)
    FROM (
        SELECT COUNT(ja2.jobAppID) AS job_app_count
        From JobApplication ja2
        JOIN Job j2 ON ja2.jobID = j2.jobID
        GROUP BY j2.companyID
    )
)
ORDER BY TotalJobApplications DESC;
```

Output:

	COMPANYNAME	TOTALJOBAPPLICATIONS
1	Apple Canada	2
2	SAMSUNG	2
3	AMD	2

Query that lists all companies that have at least one job posting where at least one job applicant has sent a job application:

```sql
-- Lists all companies that have at least one job posting where at least one job applicant has applied (sent a job application) - uses EXISTS
SELECT DISTINCT c.name AS CompanyName
FROM Company c
WHERE EXISTS (
    SELECT 1
    FROM Job j
    JOIN JobApplication ja ON j.jobID = ja.jobID
    WHERE j.companyID = c.companyID
);
```

Output:

	COMPANYNAME
1	Apple Canada
2	Royal Bank of Canada (RBC)
3	SAMSUNG
4	AMD

Query that lists all the job applicants who have sent a job application for at least one job, but have never been interviewed:

```sql
-- Lists all the job applicants that have applied (sent a job application) for at least one job, but have never been interviewed - uses MINUS
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
MINUS
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Interview i ON ja.jobAppID = i.jobAppID;
```

Output:

	APPLICANTID	FIRST_NAME	LAST_NAME
1	1	Alice	Bob
2	2	Jake	Blake
3	3	Griffin	Walker

Query that lists the average salary of all jobs posted, including the minimum and maximum salaries of the jobs posted by the company:

```sql
-- Lists the average salary of all jobs posted, including the minimum and maximum salaries of the jobs, by the company - uses aggregates AVG, MIN, and MAX
SELECT
    c.name AS CompanyName,
    AVG(j.salary) AS AvgSalary,
    MIN(j.salary) AS MinSalary,
    MAX(j.salary) AS MaxSalary
FROM Company c
JOIN Job j ON c.companyID = j.companyID
GROUP BY c.name
ORDER BY AvgSalary DESC;
```

Output:

	COMPANYNAME	AVGSALARY	MINSALARY	MAXSALARY
1	AMD	36.166666666666666666666666666666666666667	28.5	45
2	Apple Canada	32.85	29	40.75
3	SAMSUNG	27.166666666666666666666666666666666666667	25	30
4	Royal Bank of Canada (RBC)	26	22	31.5

Query that lists all companies who have posted jobs, but have no interviews scheduled yet

```
SELECT c.name as CompanyName
FROM Company c, Job j
WHERE c.companyID = j.companyID
MINUS
SELECT c.name as CompanyName
FROM Company c, Job j, JobApplication japp, Interview i
WHERE c.companyID = j.companyID AND j.jobID = japp.jobID AND japp.jobAppID = i.jobAppID;
```

Output

	COMPANYNAME
1	Royal Bank of Canada (RBC)

Query that lists companies that have above average working hours for jobs and list their average working hours and the company number of jobs the company has posted

```
SELECT c.name as CompanyName, AVG(j.workingHours) AS AverageWorkingHours, COUNT(j.jobID) as TotalJobs
FROM Company c, Job j
WHERE c.companyID = j.companyID
GROUP BY c.name
HAVING AVG(j.workingHours) > (
    SELECT AVG(workingHours)
    FROM Job);
```

Output

	COMPANYNAME	AVERAGEWORKINGHOURS	TOTALJOBS
1	AMD	40	3
2	SAMSUNG	40.666666666666666666666666666666666666667	3

Query that lists all recruiters and their email that work with apple and applicants with their email that applied for positions at apple

```
SELECT r.last_name, r.first_name, r.email
FROM RECRUITER r, Company c
WHERE r.companyID = c.companyID AND c.name = 'Apple Canada'
UNION
SELECT ja.last_name, ja.first_name, ja.email
FROM JobApplicant ja, JobApplication japp, Job j, Company c
WHERE c.name ='Apple Canada' and ja.applicantID = japp.applicantID and japp.jobID = j.jobID AND j.companyID = c.companyID;
```

Output

	LAST_NAME	FIRST_NAME	EMAIL
1	Bob	Alice	alice.bob@gmail.com
2	Doe	Jane	jane.doe@apple.com
3	Stephens	Ed	ed.stephens@outlook.com

Query that lists all companies, that have received at least one job application still under review

```
--List all companies that have received atleast one job application still under review
SELECT c.name AS CompanyName
FROM Company c
WHERE EXISTS (
    SELECT *
    FROM Job j, JobApplication japp
    WHERE j.companyID = c.companyID AND japp.jobID = j.jobID AND japp.status = 'Under Review');
```

Output

	COMPANYNAME
1	Royal Bank of Canada (RBC)

Query that list all applicants and their email who have never been interviewed for a job

```
SELECT ja.last_name, ja.first_name, ja.email
FROM JobApplicant ja
WHERE NOT EXISTS (
    SELECT *
    FROM JobApplication japp, Interview i
    WHERE japp.applicantID = ja.applicantID AND i.jobAppID = japp.jobAppID);
```

Output

	LAST_NAME	FIRST_NAME	EMAIL
1	Bob	Alice	alice.bob@gmail.com
2	Blake	Jake	jake.blake@hotmail.com
3	Walker	Griffin	griffin.walker@outlook.com
4	Inactive	Sam	sam.inactive@gmail.com

For the next queries, the following lines were added to the database:

```
-- quick new insertion here:
INSERT INTO Company VALUES (5, 'Empty Company', 'Nothing', 'Ellesmere Island, Nunavut', 'nothing@nil.com', '000-000-0000');

-- another one, though
INSERT INTO Company VALUES (6, 'Less Empty Company', 'A few things', 'Baffin Island, Nunavut', 'nothing@something.com', '000-000-0001');
INSERT INTO Recruiter VALUES (5, 6, 'Satsuki', 'Uruma', 'satsuki.uruma@something.com', '647-333-4444');
INSERT INTO Job VALUES (15, 6, 5, 'Less Empty Company', 30.00, 40.00, TO_DATE('2025-11-06', 'YYYY-MM-DD'), 'Yellowknife, Northwest Territories, Canada', 'Security Guard', 'Secure the main Less Empty Company building against intruders.');
```

Query that gets the names, emails, and phones of all users (people) on the platform who are active (recruiter posted at least one job, applier applied to at least one job)

```sql
-- get the names, emails, and phones of all users on the platform who are "active" (posted or applied to at least one job)
SELECT first_name, last_name, email, phone, role AS user_role FROM
    (SELECT first_name, last_name, email, phone, 'JobApplicant' AS role
        FROM JobApplicant
        WHERE EXISTS
            (SELECT 1 FROM JobApplication WHERE JobApplication.applicantID = JobApplicant.applicantID))
    UNION
    (SELECT first_name, last_name, email, phone, 'Recruiter' AS role
        FROM Recruiter
        WHERE EXISTS
            (SELECT 1 FROM Job WHERE Job.recruiterID = Recruiter.recruiterID));
```

Output:

	FIRST_NAME	LAST_NAME	EMAIL	PHONE	USER_ROLE
1	Alice	Bob	alice.bob@gmail.com	416-123-455	JobApplicant
2	Bob	William	bob.william@rbc.com	416-111-1111	Recruiter
3	Ed	Stephens	ed.stephens@outlook.com	905-444-2121	JobApplicant
4	Griffin	Walker	griffin.walker@outlook.com	905-289-9876	JobApplicant
5	Jack	Jones	jack.jones@samsung.com	647-333-4444	Recruiter
6	Jake	Blake	jake.blake@hotmail.com	647-444-1947	JobApplicant
7	Jane	Doe	jane.doe@apple.com	647-222-3333	Recruiter
8	Joe	Random	joe.random@gmail.com	647-543-2211	JobApplicant
9	John	Daniels	john.daniels@amd.com	905-423-5678	Recruiter
10	Michael	Jordan	michael.jordan@gmail.com	416-989-7777	JobApplicant
11	Satsuki	Uruma	satsuki.uruma@something.com	647-333-4444	Recruiter

Query that retrieves the average job application count per job posting per company for companies that have posted at least one job

```sql
-- get the average job application count per job posting per company for companies that have posted at least one job
-- this will not list Empty Company as it has not posted any jobs, but it will list Less Empty Company
SELECT Job.employer, COUNT(JobApplication.jobID) / COUNT(Job.jobID) AS avg_application_count
FROM Job FULL JOIN JobApplication ON Job.jobID = JobApplication.jobID
GROUP BY Job.employer
HAVING COUNT(Job.jobID) > 0
ORDER BY avg_application_count DESC;
```

Output:

All rows fetched: 5 in 0.060 seconds

	EMPLOYER	AVG_APPLICATION_COUNT
1	Samsung	0.5
2	AMD	0.5
3	Royal Bank of Canada (RBC)	0.3333333333333333333333333333333333333333
4	Apple Canada	0.3333333333333333333333333333333333333333
5	Less Empty Company	0

Query that retrieves job applicants who have an interview scheduled or pending, and have applied to more than the average number of jobs applied to per applicant

```sql
-- get the above-average job applicants -- those who have an interview schedule or pending, and have applied to more than the average number of jobs
SELECT first_name, last_name, email, phone, count(jobAppID) FROM JobApplicant JOIN JobApplication ON JobApplicant.applicantID = JobApplication.applicantID
WHERE EXISTS
    (SELECT 1 FROM Interview JOIN JobApplication On Interview.jobAppID = JobApplication.jobAppID JOIN JobApplicant ON JobApplication.applicantID = JobApplicant.applicantID)
HAVING COUNT(jobAppID) > (SELECT AVG(application_count) FROM
    (SELECT applicantID, COUNT(jobAppID) AS application_count FROM JobApplication GROUP BY applicantID) subquery)
GROUP BY JobApplicant.applicantID, first_name, last_name, email, phone;
```

Output:

All rows fetched: 1 in 0.043 seconds

	FIRST_NAME	LAST_NAME	EMAIL	PHONE	COUNT(JOBAPPID)
1	Joe	Random	joe.random@gmail.com	647-543-2211	2

(only one entry because average job application is 1.167ish)

Query retrieving recruiters and the number of jobs they have posted with zero applications

```sql
-- get recruiters and the number of jobs they have posted with zero applications
SELECT DISTINCT Recruiter.recruiterID, Recruiter.first_name, Recruiter.last_name, Recruiter.email, Recruiter.phone, COUNT(zero_app_jobs.jobID) AS jobs_with_zero_applications
FROM
    (SELECT DISTINCT Job.jobID, Job.title, Job.employer, Job.datePosted, Job.recruiterID
        FROM Job
    MINUS
    SELECT DISTINCT Job.jobID, Job.title, Job.employer, Job.datePosted, Job.recruiterID
        FROM Job JOIN JobApplication ON Job.jobID = JobApplication.jobID)
    zero_app_jobs
LEFT JOIN Recruiter ON zero_app_jobs.recruiterID = Recruiter.recruiterID
GROUP BY Recruiter.recruiterID, Recruiter.first_name, Recruiter.last_name, Recruiter.email, Recruiter.phone;
```

Output:

	RECRUITERID	FIRST_NAME	LAST_NAME	EMAIL	PHONE	JOBS_WITH_ZERO_APPLICATIONS
1	2	Bob	William	bob.william@rbc.com	416-111-1111	2
2	3	John	Daniels	john.daniels@amd.com	905-423-5678	2
3	5	Satsuki	Uruma	satsuki.uruma@something.com	647-333-4444	1
4	4	Jack	Jones	jack.jones@samsung.com	647-333-4444	2
5	1	Jane	Doe	jane.doe@apple.com	647-222-3333	4

Query getting job applicants by their job working hour preferences (showing the max, min, and average working hours of the jobs they have applied to)

```sql
-- get applicants by what their working hour preferences are (the maximum working hours, minimum working hours, and average working hours of the jobs they have applied to)
SELECT JobApplicant.applicantID, JobApplicant.first_name, JobApplicant.last_name, JobApplicant.email,
    MAX(Job.workingHours) AS max_working_hours,
    MIN(Job.workingHours) AS min_working_hours,
    AVG(Job.workingHours) AS avg_working_hours
    FROM JobApplicant JOIN JobApplication ON JobApplicant.applicantID = JobApplication.applicantID JOIN Job on Job.jobID = JobApplication.jobID
GROUP BY JobApplicant.applicantID, JobApplicant.first_name, JobApplicant.last_name, JobApplicant.email;
```

Output:

	APPLICANTID	FIRST_NAME	LAST_NAME	EMAIL	MAX_WORKING_HOURS	MIN_WORKING_HOURS	AVG_WORKING_HOURS
1	1	Alice	Bob	alice.bob@gmail.com	36.25	36.25	36.25
2	6	Michael	Jordan	michael.jordan@gmail.com	42	42	42
3	3	Griffin	Walker	griffin.walker@outlook.com	40	40	40
4	4	Ed	Stephens	ed.stephens@outlook.com	36.25	36.25	36.25
5	2	Jake	Blake	jake.blake@hotmail.com	35	35	35
6	5	Joe	Random	joe.random@gmail.com	42	40	41

(lots of MAX = MIN = AVG because alot of these applicants have only applied to one job, but Joe Random shows the efficacy of the query)

Normalization: 1NF, 2NF, 3NF, and BCNF

Functional Dependencies

Below are the functional dependencies for each table, which are denoted by 'FD'. We based these on the table definitions given in the previous assignment files.

COMPANY

companyID (primary key)	name	industry	location	email	phone

FD: companyID → name, industry, location, email, phone
FD: name → companyID, industry, location, email, phone

*Since name and companyID are necessarily unique per row, the relation from name/companyID onto industry/location/phone is always a function because the latter values **always** relate to one and only one of the former two values; the two are always unique per row, after all. This functional dependency is actually built-in due to the table constraints.*

RECRUITER

recruiterID (primary key)	companyID (references **COMPANY**.companyID)	first_name	last_name	email	phone

FD: recruiterID → companyID, first_name, last_name, email, phone
FD: email → recruiterID, companyID, first_name, last_name, phone

Same reasoning as above. companyID and email are always unique per row, so you cannot have one first_name/last_name/phone/recruiterID/email map onto multiple emails/recruiterIDs.

JOB

jobID (primary key)	companyID (references **COMPANY**.companyID)	recruiterID (references **RECRUITER.** recruiterID)	employer	salary	workingHours	datePosted	location	title	description

FD: jobID → companyID, recruiterID, employer, salary, workingHours, datePosted, location, title, description

JOB_APPLICANT

applicantID (primary key)	first_name	last_name	industry	birthdate	address	email	phone

FD: applicantID → first_name, last_name, industry, birthdate, address, email, phone
FD: email → applicantID, first_name, last_name, industry, birthdate, address, phone

email is unique and not null on each row, so there must be an FD from it onto every other attribute, as per the reasoning described above.

JOB_APPLICATION

jobAppID (primary key)	jobID **(**references **JOB.**jobID**)**	applicantID (references **JOB_APPLICANT**.applicantID)	dateTime	status

FD: jobAppID → jobID, applicantID, dateTime, status

RESUME

resumeID (primary key)	applicantID (references **JOB_APPLICANT.**applicantID**)**	upload_file	uploadDate

FD: resumeID → applicantID, upload_file, uploadDate

INTERVIEW

interviewID (primary key)	jobAppID (references **JOB_APPLICATION.**jobAppID**)**	dateTime	location

FD: interviewID → jobAppID, dateTime, location

Database Normalization: 2NF and 3NF

COMPANY

companyID (primary key)	name	industry	location	email	phone

1NF: All column values are atomic.
2NF: All candidate keys are non-composite.

- If the primary key for was instead **{companyID, name}**, the primary key FD would be:
  - **companyID, name** → industry, location, email, phone
  - However, both are also true:
    - companyID → industry, location, email, phone
    - name → industry, location, email, phone
  - As such, Industry, location, email, phone **would not depend on ALL attributes of the primary key**
    - That is, our primary key is not a minimal subset of the superkey
  - To make into 2NF, the primary key can either be JUST {companyID} and {name}, OR {companyID, name} can be reworked s.t. companyID and name are not always not null unique, but the pair is always not null unique
- Hence, companyID as the primary key ensures this table is in 2NF.

3NF: There exist no functional dependencies between non-candidate-key attributes

- To verify:
  - All other attributes besides name and companyID have no constraints
    - This means that no functional dependencies exist upon them, as all columns are able to take on null values and relate to other attributes through that null value – meaning it is not a functional dependency
    - For example: email value *null* may map to two different phone numbers and vice versa

RECRUITER

recruiterID (primary key)	companyID (references **COMPANY**.companyID)	first_name	last_name	email	phone

1NF: All column values are atomic.
2NF: All candidate keys are non-composite.
3NF: There exist no functional dependencies between non-candidate-key attributes (attributes besides email and recruiterID)

- To verify:
  - CompanyID can be the same across multiple recruiters with different first name, last name, email, and phone (which can be null and also not unique (which is possible, technically!))

JOB

jobID (primary key)	companyID (references **COMPANY**.companyID)	recruiterID (references **RECRUITER.**recruiterID)	employer	salary	workingHours	datePosted	location	title	description

1NF: All column values are atomic.
2NF: All candidate keys (minimal subsets of super keys) are non-composite.

- In this case, the only real candidate key here is jobID

3NF: There exists an extraneous functional dependency between non-key attributes:

- companyID → employer
- To remedy this, we can decompose the table as such:

jobID (primary key)	companyID (references **COMPANY**.companyID)	recruiterID (references **RECRUITER.**recruiterID)	salary	workingHours	datePosted	location	title	description

companyID (references **COMPANY**.companyID)	employer	… (other values relating to company)

- Fortunately, there exists an equivalent field to “employer” in the company’s table called *name*, so practically all we need to do is remove the employer field from the Job table
    - If we need the employer of the job, we can get the name from the company’s table referencing companyID

JOB_APPLICANT

applicantID (primary key)	first_name	last_name	industry	birthdate	address	email	phone

1NF: All column values are atomic.
2NF: All candidate keys (minimal subsets of super keys) are non-composite.

- In this case, the only real candidate keys here are applicantID and email. If the primary key were a composite of both, then every other non-prime attribute would depend on it, but also depend on the attributes individually, so it would not be 2NF

3NF: There exist no functional dependencies between non-candidate-key attributes.

- There are no constraints on other attributes besides email and applicantID, so the null argument still applies – it is also possible for users to have the same first name, last name, industry, birthdate, address (if they live in the same house!), and phone number (technically)

JOB_APPLICATION

jobAppID (primary key)	jobID **(**references **JOB.**jobID**)**	applicantID (references **JOB_APPLICANT**.applicantID)	dateTime	status

1NF: All column values are atomic.

2NF: All candidate keys (minimal subsets of super keys) are non-composite.

- Another possible candidate key is {jobID, applicantID}, but because you can only send one job application per job posting, this both works as a primary key and is also a minimal superkey
    - Removing any set from that attribute changes the uniqueness of the jobID, applicantID pairing – this applicant AND this job is the essential info

3NF: There are no extraneous FDs between non-key attributes

- As established before, the candidate-key pair of {jobID, applicantID} count as key attributes, and so any FDs from that are also candidate-key FDs and exempt from 3NF's restriction upon it

RESUME

resumeID (primary key)	applicantID (references **JOB_APPLICANT.**applicantID**)**	upload_file	uploadDate

1NF: All column values are atomic.

2NF: All candidate keys (minimal subsets of super keys) are non-composite.

- Only resumeID is a possible candidate key here

3NF: There are no extraneous FDs between non-key attributes

INTERVIEW

interviewID (primary key)	jobAppID (references **JOB_APPLICATION.**jobAppID**)**	dateTime	location

1NF: All column values are atomic.

2NF: All candidate keys (minimal subsets of super keys) are non-composite.

- One jobAppID may have multiple interviews as part of a multi-stage process

3NF: There are no extraneous FDs between non-key attributes

- The "location" field specifies interview_location, which may be different from the job of the jobApp's location (we should probably rename that field)

Database Normalization: Bernstein's Algorithm for 3NF and BCNF Algorithm

COMPANY

companyID (primary key)	name	industry	location	email	phone

There are no partial dependencies since all candidate keys are single attributes. There are no transitive dependencies because every non-key attribute is non-transitively dependent on the primary key.

Bernstein's Algorithm

Step 1: Determine the functional dependencies

- companyID → name, industry, location, email, phone
- name → companyID, industry, location, email, phone
    - name is a unique attribute

Step 2: Break RHS and find redundancies

- companyID → name, companyID$^+$ = {companyID, industry, location, email, phone} we do not get name so not redundant
- companyID → industry, companyID$^+$ = {companyID, name, location, email, phone} we do not get industry so not redundant
- companyID → location, companyID$^+$ = {companyID, name, industry, email, phone} we do not get location so not redundant
- companyID → email, companyID$^+$ = {companyID, name, industry, location, phone} we do not get email so not redundant
- companyID → phone companyID$^+$ = {companyID, name, industry, location, email} we do not get phone so not redundant
- name → companyID name$^+$ = {name, industry, location, email, phone} we do not get companyID so not redundant
- name → industry name$^+$ = {name, companyID, location, email, phone} we do not get industry so not redundant
- name → email name$^+$ = {name, companyID, industry, location, phone} we do not get email so not redundant
- name → phone name$^+$ = {name, companyID, industry, location, email} we do not get phone so not redundant

Step 2b: Find and remove Partial dependencies

- No partial dependencies because COMPANY has no non-key attributes that depend on part of a composite key and all non key attributes depend on the candidate keys

Step 3: Find keys

- companyID$^+$ = {companyID, name, industry, location, email, phone}
- name$^+$ = {name, companyID, industry, location, email, phone}
- companyID and name both determine all the attributes so companyID and name are both candidate keys

Step 4: Make tables

Create a relation for each functional dependency

- companyID → name, industry, location, email, phone: R1(companyID, name, industry, location, email, phone)
- name → companyID, industry, location, email, phone: R2(companyID, name, industry, location, email, phone)

We can remove R2 because it has the same attributes as R1
Final schema is R1(companyID, name, industry, location, email, phone)

<u>BCNF Algorithm</u>
Functional Dependencies (FD):

- companyID → name, industry, location, email, phone
- name → companyID, industry, location, email, phone
    - name is a unique attribute

RHS-split:

- companyID → name, companyID → industry, companyID → location, companyID → email, companyID → phone, name → companyID, name → industry, name → location, name → email, name → phone

Keys (closures):

- companyID$^+$ = {companyID, name, industry, location, email, phone} → companyID is a key
- name$^+$ = {name, companyID, industry, location, email, phone} → name is a key
- So, the candidate keys are: {companyID, name}

Every FD has a key attribute, either companyID or name, both of which are superkeys.

Both name and companyID are candidate keys. Since both of their keys (closures) reveal identical attribute sets, no decomposition is required. If we introduce a new relation for name, then we would have duplicate attributes. Thus, we use companyID as the primary key and use name as a unique attribute to preserve the alternate key.

Decomposition:

- Initialize D := {R1}
    - D = { R1(companyID, name, industry, location, email, phone) }
- Check each FD
    - Both key attributes (companyID and name) are candidate keys, so no FD violates BCNF
- No schema in D is decomposed

Therefore, D = { R1(companyID, name, industry, location, email, phone) } is already in BCNF.

RECRUITER

recruiterID (primary key)	companyID (references	first_name	last_name	email	phone

	COMPANY.co mpanyID)				

There are no partial dependencies since there are no composite candidate keys. There are no transitive dependencies because every non-key attribute is non-transitively dependent on the primary key.

Bernstein's Algorithm

Step 1: Determine the functional dependencies

- recruiterID → companyID, first_name, last_name, email, phone
- email → recruiterID, companyID, first_name, last_name, phone

Step 2: Break RHS and find redundancies

- recruiterID → companyID recruiterID$^+$ = {recruiterID, first_name, last_name, email, phone} we do not get companyID so not redundant
- recruiterID → first_name recruiterID$^+$ = {recruiterID, companyID, last_name, email, phone} we do not get first_name so not redundant
- recruiterID → last_name recruiterID$^+$ = {recruiterID, companyID, first_name, email, phone} we do not get last_name so not redundant
- recruiterID → email recruiterID$^+$ = {recruiterID, companyID, first_name, last_name, phone} we do not get last_name so not redundant
- recruiterID → phone recruiterID$^+$ = {recruiterID, companyID, first_name, last_name, email} we do not get phone so not redundant
- email → recruiterID email$^+$ = {email, companyID, first_name, last_name, phone} we do not get recruiterID so not redundant
- email → companyID email$^+$ = {email, recruiterID, first_name, last_name, phone} we do not get companyID so not redundant
- email → first_name email$^+$ = {email, recruiterID, companyID, last_name, phone} we do not get companyID so not redundant
- email → last_name email$^+$ = {email, recruiterID, companyID, first_name, phone} we do not get last_name so not redundant
- email → phone email$^+$ = {email, recruiterID, companyID, first_name, last_name} we do not get phone so not redundant

Step 2b: Find and remove any partial dependencies

- No partial dependencies because RECRUITER has no non-key attributes that depend on part of a composite key and all non key attributes depend on the candidate keys

Step 3: Find Keys

- recruiterID$^+$ = {recruiterID, companyID, first_name, last_name, email, phone}
- email$^+$ = {email, recruiterID, companyID, first_name, last_name, phone}
- recruiterID and email both determine all the attributes, so recruiterID and email are both candidate keys

Step 4: Make tables
Create a relation for each functional dependency

- recruiterID → companyID, first_name, last_name, email, phone: R1(recruiterID, companyID, first_name, last_name, email, phone)
- email → recruiterID, companyID, first_name, last_name, phone: R2(email, recruiterID, companyID, first_name, last_name, phone)

We can remove R2 because it has the same attributes as R1
Final schema is R1(recruiterID, companyID, first_name, last_name, email, phone)

<u>BCNF Algorithm</u>
Functional Dependencies (FD):

- recruiterID → companyID, first_name, last_name, email, phone
- email → recruiterID, companyID, first_name, last_name, phone
    - email is a unique attribute

RHS-split:

- recruiterID → companyID, recruiterID → first_name, recruiterID → last_name, recruiterID → email, recruiterID → phone, email → recruiterID, email → companyID, email → first_name, email → last_name, email → phone

Keys (closures):

- $recruiterID^+$ = {recruiterID, companyID, first_name, last_name, email, phone} → recruiterID is a key
- $email^+$ = {email, recruiterID, companyID, first_name, last_name, phone} → email is a key
- So, the candidate keys are: {recruiterID, email}

Every FD has a key attribute, either recruiterID or email, both of which are superkeys.

Both email and recruiterID are candidate keys. Since both of their keys (closures) reveal identical attribute sets, no decomposition is required. If we introduce a new relation for email, then we would have duplicate attributes. Thus, we use recruiterID as the primary key and use email as a unique attribute to preserve the alternate key.

Decomposition:

- Initialize D := {R1}
    - D = { R1(recruiterID, companyID, first_name, last_name, email, phone) }
- Check each FD
    - Both key attributes (recruiterID and email) are candidate keys, so no FD violates BCNF
- No schema in D is decomposed

Therefore, D = { R1(recruiterID, companyID, first_name, last_name, email, phone) } is already in BCNF.

JOB

jobID (primary key)	companyID (references **COMPANY**.companyID)	recruiterID (references **RECRUITER.**recruiterID)	salary	workingHours	datePosted	location	title	description

There are no partial dependencies since there are no composite keys. There are no transitive dependencies because every non-key attribute is non-transitively dependent on the primary key.

Note: If we kept employer as an attribute, we would have had companyID → employer, which means jobID → companyID → employer would be a transitive dependency, and it would violate 3NF and BCNF. Removing employer from this table in a7 (lab 7) resolves this issue.

Bernstein's algorithm

Step 1: Determine the functional dependencies

- jobID → companyID, recruiterID, salary, workingHours, datePosted, location, title, description

Step 2: Break RHS and find redundancies

- jobID → companyID $jobID^+$ = {jobID, recruiterID, salary, workingHours, datePosted, location, title, description} we do not get companyID so not redundant
- jobID → recruiterID $jobID^+$ = {jobID, companyID, salary, workingHours, datePosted, location, title, description} we do not get recruiterID so not redundant
- jobID → salary $jobID^+$ = {jobID, companyID, recruiterID, workingHours, datePosted, location, title, description} we not get salary so not redundant
- jobID → workingHours $jobID^+$ = {jobID, companyID, recruiterID, salary, datePosted, location, title, description} we do not get workingHours so not redundant
- jobID → datePosted $jobID^+$ = {jobID, companyID, recruiterID, salary, workingHours, location, title, description} we do not get datePosted so not redundant
- jobID → location $jobID^+$ = {jobID, companyID, recruiterID, salary, workingHours, datePosted, title, description} we do not get location so not redundant
- jobID → title $jobID^+$ = {jobID, companyID, recruiterID, salary, workingHours, datePosted, location, description} we do not get title so not redundant
- jobID → description $jobID^+$ = {jobID, companyID, recruiterID, salary, workingHours, datePosted, location, title} we do not get description so not redundant

Step 3: Find keys

- $jobID^+$ = {jobID, companyID, recruiterID, salary, workingHours, datePosted, location, title, description}
- All the attributes are determined by jobID so jobID is the only key for this table

Step 4: Make tables

Create a relation for each functional dependency

- jobID → companyID, recruiterID, salary, workingHours, datePosted, location, title, description: R1(jobID, companyID, recruiterID, salary, workingHours, datePosted, location, title, description)

Final schema is R1(recruiterID, companyID, first_name, last_name, email, phone).

<u>BCNF Algorithm</u>

Functional Dependencies (FD):

- jobID → companyID, recruiterID, salary, workingHours, datePosted, location, title, description

RHS-split:

- jobID → companyID, jobID → recruiterID, jobID → salary, jobID → workingHours, jobID → datePosted, jobID → location, jobID → title, jobID → description

Keys (closures):

- $jobID^+$ = {jobID, companyID, recruiterID, salary, workingHours, datePosted, location, title, description} → jobID is a key
- jobID is the only key (nothing else determines jobID).

All FDs have a key attribute (jobID), which is a superkey. Thus, there is no BCNF violation.

Decomposition:

- Initialize D := {R1}
  - D = { R1(jobID, companyID, recruiterID, salary, workingHours, datePosted, location, title, description) }
- Check each FD
  - jobID is the key (superkey), so no FD violates BCNF.
- No schema in D is decomposed since there are no violating FDs.

Therefore, D = { R1(jobID, companyID, recruiterID, salary, workingHours, datePosted, location, title, description) } is already in BCNF.

JOB_APPLICANT

applicantID (primary key)	first_name	last_name	industry	birthdate	address	email	phone

There are no partial dependencies since there are no composite candidate keys. There are no transitive dependencies because every non-key attribute is non-transitively dependent on the primary key (any apparent chain, such as applicantID → email → phone, passes through a key attribute, which does not violate 3NF or BCNF).

<u>Bernstein's algorithm</u>

Step 1: Determine the functional dependencies

- applicantID → first_name, last_name, industry, birthdate, address, email, phone
- email → applicantID, first_name, last_name, industry, birthdate, address, phone

Step 2: Break RHS and remove redundancies

- applicantID → first_name applicantID$^+$ = {applicantID, last_name, industry, birthdate, address, email, phone} we do not get first_name so not redundant
- applicantID → last_name applicantID$^+$ = {applicantID, first_name, industry, birthdate, address, email, phone} we do not get last_name so not redundant
- applicantID → industry applicantID$^+$ = {applicantID, first_name, last_name, birthdate, address, email, phone} we do not get industry so not redundant
- applicantID → birthdate applicantID$^+$ = {applicantID, first_name, last_name, industry, address, email, phone} we do not get birthdate so not redundant
- applicantID → address applicantID$^+$ = {applicantID, first_name, last_name, industry, birthdate, email, phone} we do not get address so not redundant
- applicantID → email applicantID$^+$ = {applicantID, first_name, last_name, industry, birthdate, address, phone} we do not get email so not redundant
- applicantID → phone applicantID$^+$ = {applicantID, first_name, last_name, industry, birthdate, address, email} we do not get phone so not redundant
- email→ applicantID email$^+$ = {email, first_name, last_name, industry, birthdate, address, phone} we do not get applicantID so not redundant
- email→ first_name email$^+$ = {email, applicantID, last_name, industry, birthdate, address, phone} we do not get first_name so not redundant
- email→ last_name email$^+$ = {email, applicantID, first_name, industry, birthdate, address, phone} we do not get last_name so not redundant
- email→ industry email$^+$ = {email, applicantID, first_name, last_name, birthdate, address, phone} we do not get industry so not redundant
- email→ birthdate email$^+$ = {email, applicantID, first_name, last_name, industry, address, phone} we do not get birthdate so not redundant
- email→ address email$^+$ = {email, applicantID, first_name, last_name, industry, birthdate, phone} we do not get address so not redundant
- email→ phone email$^+$ = {email, applicantID, first_name, last_name, industry, birthdate, address} we do not get phone so not redundant

Step 3: Find Keys

- applicantID$^+$ = {applicantID, first_name, last_name, industry, birthdate, address, email, phone}
- email$^+$ = {email, applicantID, first_name, last_name, industry, birthdate, address, phone}
- All the attributes are determined by applicantID and email so they are both the candidate keys

Step 4: Make tables

Create a relation for each of the functional dependencies

- applicantID → first_name, last_name, industry, birthdate, address, email, phone: R1(applicantID, first_name, last_name, industry, birthdate, address, email, phone)

- email → applicantID, first_name, last_name, industry, birthdate, address, phone: R2(email, applicantID, first_name, last_name, industry, birthdate, address, phone)

We can remove R2 because it has the same attributes as R1
The final schema is R1(applicantID, first_name, last_name, industry, birthdate, address, email, phone).

<u>BCNF Algorithm</u>
Functional Dependencies (FD):
- applicantID → first_name, last_name, industry, birthdate, address, email, phone
- email → applicantID, first_name, last_name, industry, birthdate, address, phone
    - email is a unique attribute

RHS-split:
- applicantID → first_name, applicantID → last_name, applicantID → industry, applicantID → birthdate, applicantID → address, applicantID → email, applicantID → phone, email → applicantID, email → first_name, email → last_name, email → industry, email → birthdate, email → address, email → phone

Keys (closures):
- $applicantID^+$ = {applicantID, first_name, last_name, industry, birthdate, address, email, phone} → applicantID is a key
- $email^+$ = {email, applicantID, first_name, last_name, industry, birthdate, address, phone} → email is a key
- So, the candidate keys are: {applicantID, email}

Every FD has a key attribute, either applicantID or email, both of which are superkeys.

Both email and applicantID are candidate keys. Since both of their keys (closures) reveal identical attribute sets, no decomposition is required. If we introduce a new relation for email, then we would have duplicate attributes. Thus, we use applicantID as the primary key and use email as a unique attribute to preserve the alternate key.

Decomposition:
- Initialize D := {R1}
    - D = { R1(applicantID, first_name, last_name, industry, birthdate, address, email, phone) }
- Check each FD
    - Both key attributes (applicantID and email) are candidate keys, so no FD violates BCNF
- No schema in D is decomposed

Therefore, D = { R1(applicantID, first_name, last_name, industry, birthdate, address, email, phone) } is already in BCNF.

JOB_APPLICATION

jobAppID (primary key)	jobID **(**references **JOB.**jobID**)**	applicantID (references **JOB_APPLICANT**.applicantID)	dateTime	status

There are no partial dependencies since there are no composite candidate keys. There are no transitive dependencies because every non-key attribute is non-transitively dependent on the primary key.

Bernstein's algorithm

Step 1: Determine the functional dependencies

- jobAppID → jobID, applicantID, dateTime, status

Step 2: Break RHS and remove redundancies

- jobAppID → jobID $jobAppID^+$ = {jobAppID, applicantID, dateTime, status} we do not have jobID so not redundant
- jobAppID → applicantID $jobAppID^+$ = {jobAppID, jobID, dateTime, status} we do not have applicantID so not redundant
- jobAppID → dateTime $jobAppID^+$ = {jobAppID, applicantID, jobID, status} we do not have dateTime so not redundant
- jobAppID → status $jobAppID^+$ = {jobAppID, applicantID, jobID, dateTime} we do not have status so not redundant

Step 3: Find keys

- $jobAppID^+$ = {jobAppID, jobID, applicantID, dateTime, status}
- All the attributes are determined by jobAppID so jobAppID is the only key for this table

Step 4: Make tables

Create a relation for each of the functional dependencies

- jobAppID → jobID, applicantID, dateTime, status: R1(jobAppID, jobID, applicantID, dateTime, status)

The final schema is R1(jobAppID, jobID, applicantID, dateTime, status)

BCNF Algorithm

Functional Dependencies (FD):

- jobAppID → jobID, applicantID, dateTime, status

RHS-split:

- jobAppID → jobID, jobAppID → applicantID, jobAppID → dateTime, jobAppID → status

Keys (closures):

- $jobAppID^+$ = {jobAppID, jobID, applicantID, dateTime, status} → jobID is the only key

All FDs have a key attribute (jobAppID), which is a superkey. Thus, there is no BCNF violation.

Decomposition:

- Initialize D := {R1}
    - D = { R1(jobAppID, jobID, applicantID, dateTime, status) }
- Check each FD
    - jobAppID is the key (superkey), so no FD violates BCNF.
- No schema in D is decomposed since there are no violating FDs.

Therefore, D = { R1(jobAppID, jobID, applicantID, dateTime, status) } is already in BCNF.

RESUME

resumeID (primary key)	applicantID (references **JOB_APPLICANT.**applicantID**)**	upload_file	uploadDate

There are no partial dependencies since RESUME has no non-key attributes that depend on part of a composite key, since resumeID is the primary key. There are no transitive dependencies because every non-key attribute is non-transitively dependent on the primary key.

Bernstein's Algorithm

Step 1: Determine the functional dependencies

- resumeID → applicantID, upload_file, uploadDate

Step 2: Break RHS and remove redundancies

- resumeID → applicantID resumeID$^+$ = {resumeID, upload_file, uploadDate}
  we do not have applicantID so not redundant
- resumeID → upload_file resumeID$^+$ = {resumeID, applicantID, uploadDate}
  we do not have upload_file so not redundant
- resumeID → uploadDate resumeID$^+$ = {resumeID, applicantID, upload_file}
  we do not have uploadDate so not redundant

Step 3: Find keys

- resumeID$^+$ = {resumeID, applicantID, upload_file, uploadDate}
- All the attributes are determined by resumeID, so resumeID is the only key for this table

Step 4: Make tables

Create a relation for each of the functional dependencies

- resumeID → applicantID, upload_file, uploadDate: R1(resumeID → applicantID, upload_file, uploadDate)

The final schema is R1(resumeID → applicantID, upload_file, uploadDate).

BCNF Algorithm

Functional Dependencies (FD):

- resumeID → applicantID, upload_file, uploadDate

RHS-split:

- resumeID → applicantID, resumeID → upload_file, resumeID → uploadDate

Keys (closures):

- $resumeID^+$ = {resumeID, applicantID, upload_file, uploadDate} → resumeID is the only key

All FDs have a key attribute (resumeID), which is a superkey. Thus, there is no BCNF violation.

Decomposition:

- Initialize D := {R1}
  - D = { R1(resumeID, applicantID, upload_file, uploadDate) }
- Check each FD
  - resumeID is the key (superkey), so no FD violates BCNF.
- No schema in D is decomposed since there are no violating FDs.

Therefore, D = { R1(resumeID, applicantID, upload_file, uploadDate) } is already in BCNF.

INTERVIEW

interviewID (primary key)	jobAppID (references **JOB_APPLICATION.**jobAppID**)**	dateTime	location

There are no partial dependencies since INTERVIEW has no non-key attributes that depend on part of a composite key, since interviewID is the primary key. There are no transitive dependencies because every non-key attribute is non-transitively dependent on the primary key.

<u>Bernstein's Algorithm</u>

Step 1: Determine the functional dependencies

- interviewID → jobAppID, dateTime, location

Step 2: Break RHS and remove redundancies

- interviewID → jobAppID $interviewID^+$ = {interviewID, dateTime, location} we do not have jobAppID so not redundant
- interviewID → dateTime $interviewID^+$ = {interviewID, jobAppID, location} we do not have dateTime so not redundant
- interviewID → location $interviewID^+$ = {interviewID, jobAppID, dateTime} we do not have location so not redundant

Step 3 Find keys

- $interviewID^+$ = {interviewID, jobAppID, dateTime, location}

- All the attributes are determined by interviewID so the only key in this table is interviewID

Step 4: Make tables
Create a relation for each of the functional dependencies
- interviewID → jobAppID, dateTime, location: R1(interviewID, jobAppID, dateTime, location)

The final schema is R1(interviewID, jobAppID, dateTime, location).

<u>BCNF Algorithm</u>
Functional Dependencies (FD):
- interviewID → jobAppID, dateTime, location

RHS-split:
- interviewID → jobAppID, interviewID → dateTime, interviewID → location

Keys (closures):
- $interviewID^+$ = {interviewID, jobAppID, dateTime, location} → interviewID is the only key

All FDs have a key attribute (interviewID), which is a superkey. Thus, there is no BCNF violation.

Decomposition:
- Initialize D := {R1}
    - D = { R1(interviewID, jobAppID, dateTime, location) }
- Check each FD
    - interviewID is the key (superkey), so no FD violates BCNF.
- No schema in D is decomposed since there are no violating FDs.

Therefore, D = { R1(interviewID, jobAppID, dateTime, location) } is already in BCNF.

Application User Guide

GitHub Link to the Project

https://github.com/arth-z/cps510-database-a9

User Guide to Run the Java GUI

Before running the program, please ensure that you have Java 8 installed. To check which version of Java you have, enter java -version command in the terminal window, and it should tell you which version of Java you have.

1. To obtain the project source code from the GitHub repository, you can do this in two ways:
    a. Using Git
        i. Create a new folder in your local system, which is where you are going to store the project
        ii. Go to the GitHub repository
        iii. Click on Code and copy the web URL of the project
        iv. Go into a new terminal window and change directories until you are in the newly created folder for the project
        v. Enter the following command in the terminal: git clone https://github.com/arth-z/cps510-database-a9.git
        vi. Once the project has been cloned to your local system, change the directory into the project folder, and you should see everything from the GitHub repository.
    b. Download the files from GitHub
        i. Create a new folder in your local system, which is where you are going to store the project
        ii. Go to the GitHub repository
        iii. Click on Code and then click on Download Zip
        iv. Once the file has been downloaded, store it in the newly created folder, and extract the files. You should see all the contents of the project folder.
2. To run the project, open your preferred IDE environment (e.g. Eclipse, Visual Studio Code, etc.) and open the project folder.
3. Make sure you are connected to the school VPN before running the program (needed to access the Oracle database).
4. To run the program, you would need to compile the ojdbc6.jar file in the lib folder of the project to access the Oracle database. To do this, you will need to open the terminal window, change the directory to the project folder with the src, bin, and lib files (cps510a9_java). Then you will enter the following commands:
    a. For Windows machines:
        i. Enter javac -cp "lib/ojdbc6.jar;src" -d bin src/*.java in the terminal
        ii. Enter java -cp "lib/ojdbc6.jar;bin" App in the terminal
    b. For Mac machines:
        i. Enter javac -cp "lib/ojdbc6.jar:src" -d bin src/*.java in the terminal

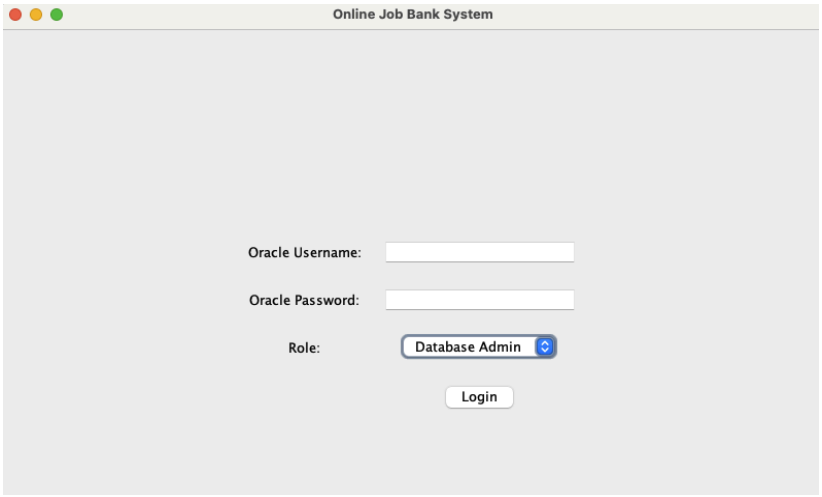
ii. Enter java -cp "lib/ojdbc6.jar:bin" App in the terminal

When you have entered these commands, the Oracle driver should be working, and the application will start running.

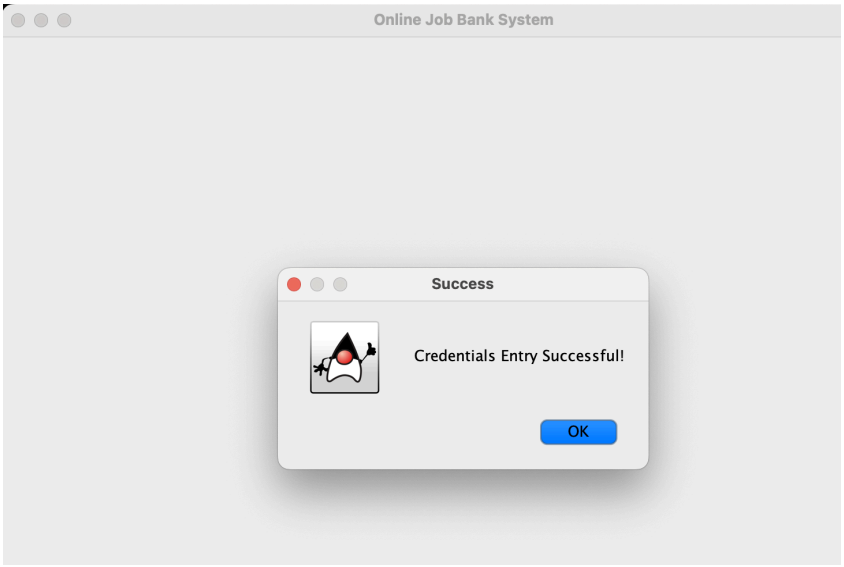
Below is the user guide of our application, which consists of various scenarios and steps for each feature and each role. Note that there are four different roles in the application: **Applicant, Company, Database Admin**, and **Recruiter**.

If User Selects "Database Admin" Role

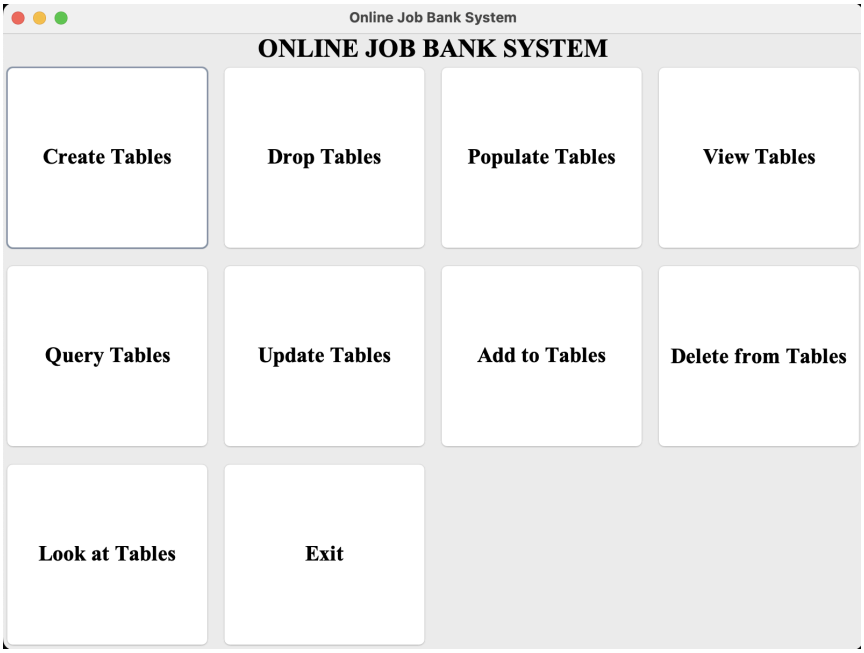
1. When the application launches, the login window appears. This is where you enter your Oracle credentials (username and password) and select the role as "**Database Admin**".



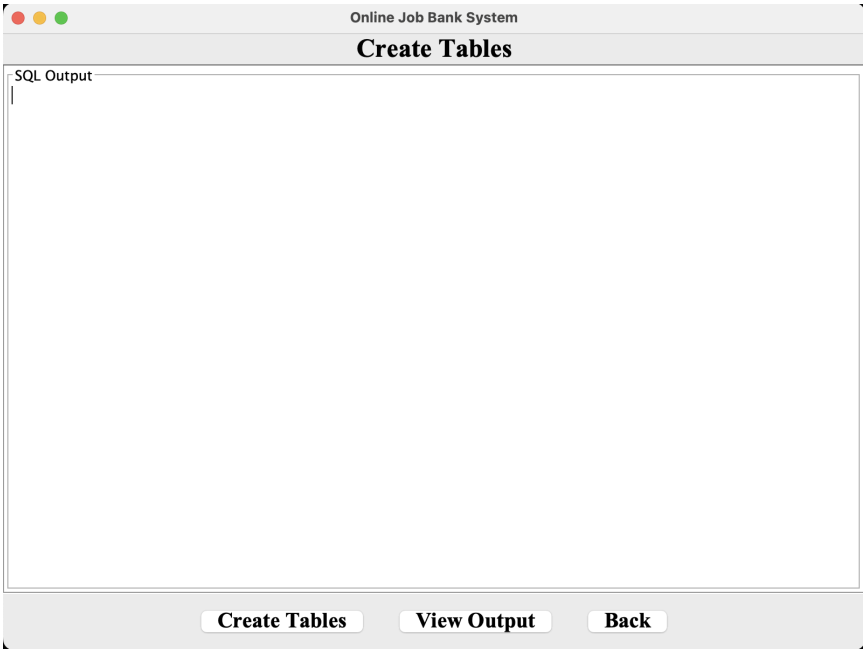
2. Once you have entered your Oracle username and password in the correct format, click the Login button.
    a. If the username and password are in the correct format, then a pop-up will appear telling you that the login is successful. When you click "Ok", you will be directed to the home screen.
    b. If the username and password are not in the correct format, then a pop-up will appear telling you that the login is unsuccessful with an error message. When you click "Ok", you will be prompted to re-enter your username and password.



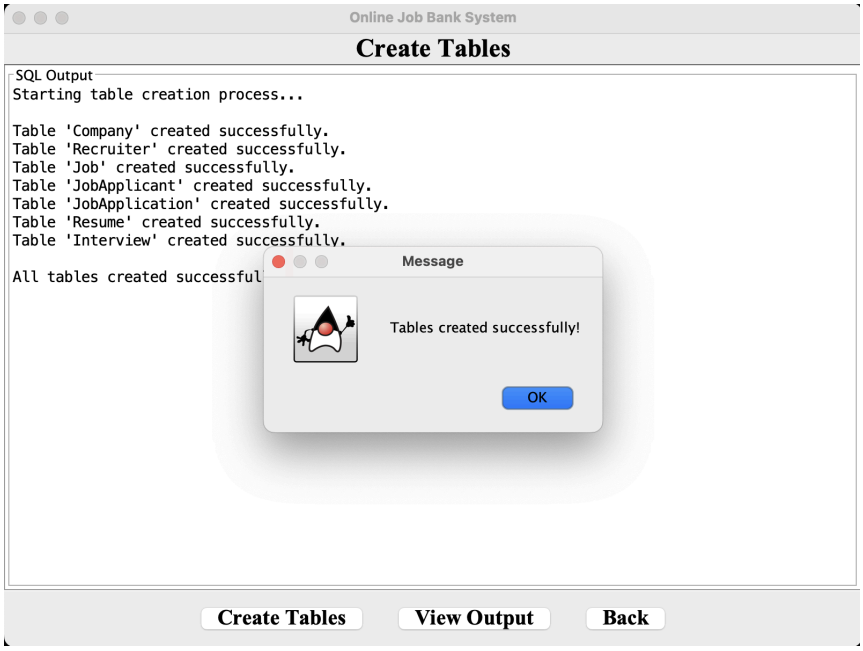
3. After logging in, you will be directed to the home screen, which consists of many features, such as Create Tables, Drop Tables, Populate Tables, View Tables, Query Tables, Update Tables, Add to Tables, Delete from Tables, Look at Tables, and Exit.



4. To create the tables, click “Create Tables”.
    a. You will be directed to the Create Tables page, which has 3 options: Create Tables, View Output, and Back, which takes you back to the home page.

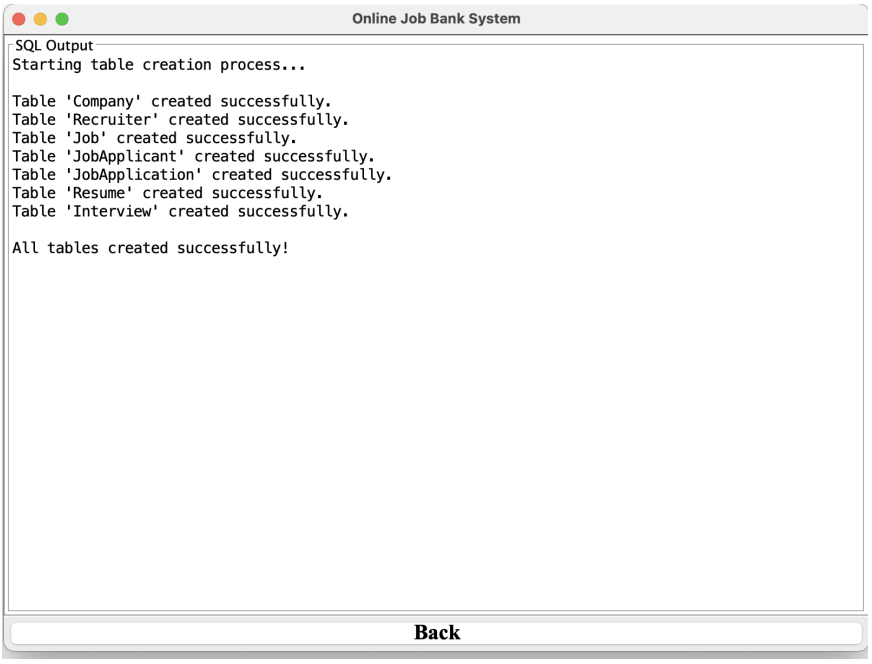


    b. Click Create Tables to create the tables for the database.
    c. If successful, then a pop-up will appear telling you that the tables were created successfully. It will also display the results of the tables being created, similar to what you see in SQLDeveloper.



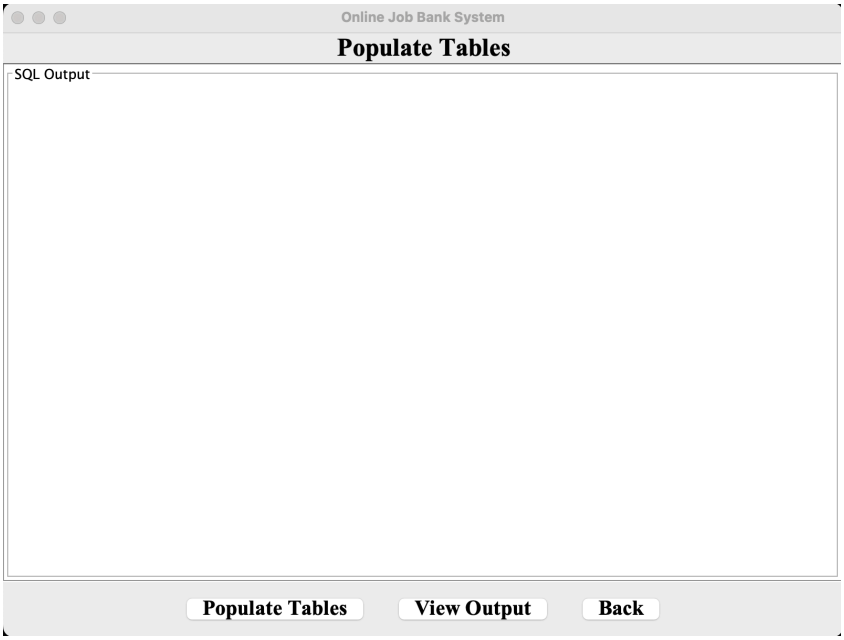
d. If unsuccessful, then a pop-up will appear telling you that the tables were not created due to an error, along with the error message.

e. If you click “View Output”, you will see a full screen of the results of the tables being created.

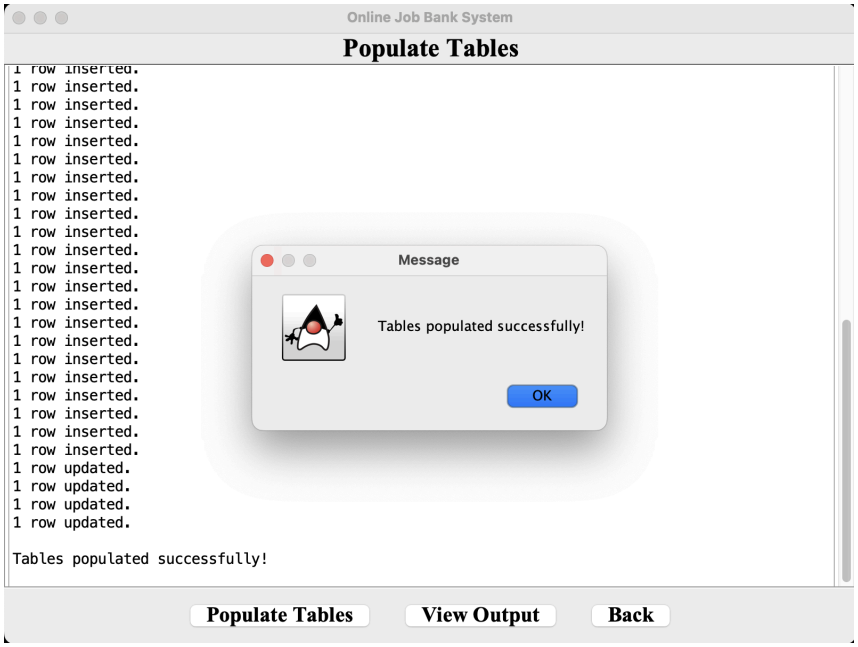


f. Click Back to go back to the home page/main menu.

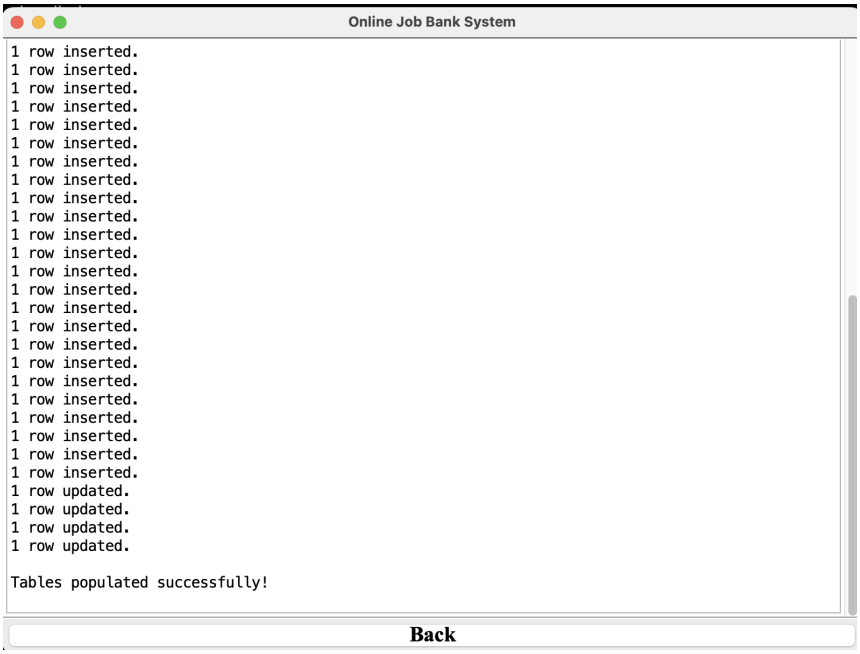
5. To populate the tables, click “Populate Tables”.
   a. You will be directed to the Populate Tables page, which has 3 options: Populate Tables, View Output, and Back, which takes you back to the home page.



b. Click Populate Tables to populate the tables.
c. If successful, then a pop-up will appear telling you that the tables were populated successfully. It will also display the results of the tables being populated, similar to what you see in SQLDeveloper.

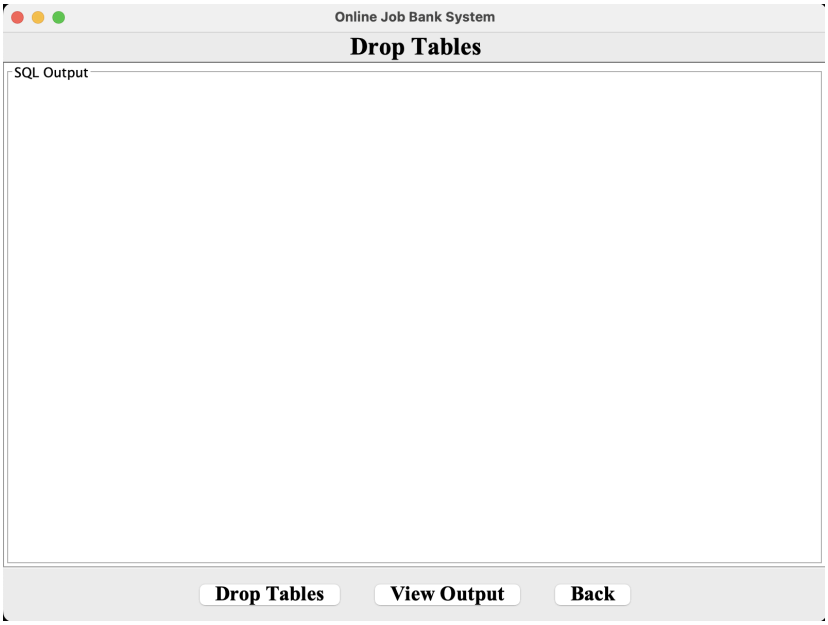


d. If unsuccessful, then a pop-up will appear telling you that the tables were not populated due to an error, along with the error message.
e. If you click “View Output”, you will see a full screen of the results of the tables being populated.

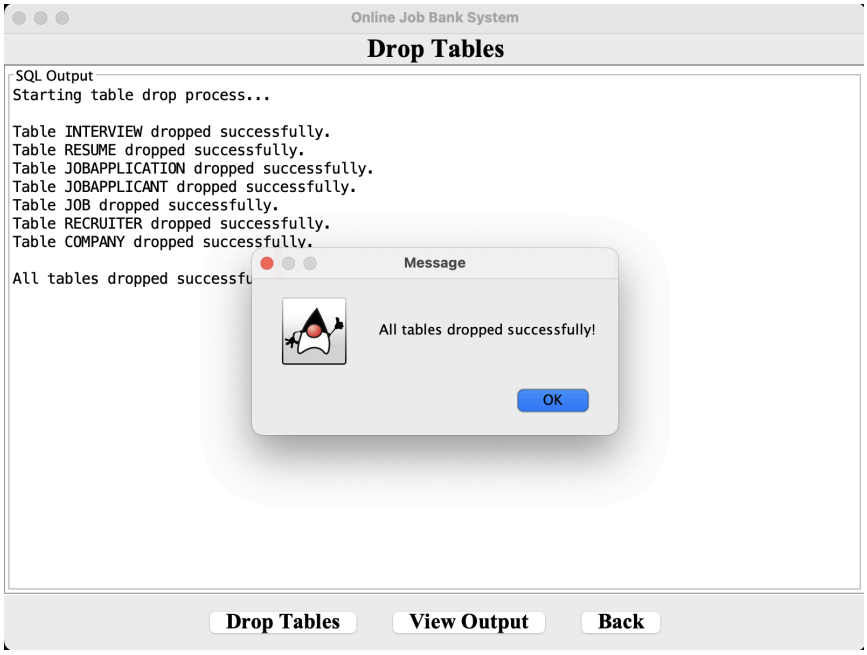


f. Click Back to go back to the home page/main menu.

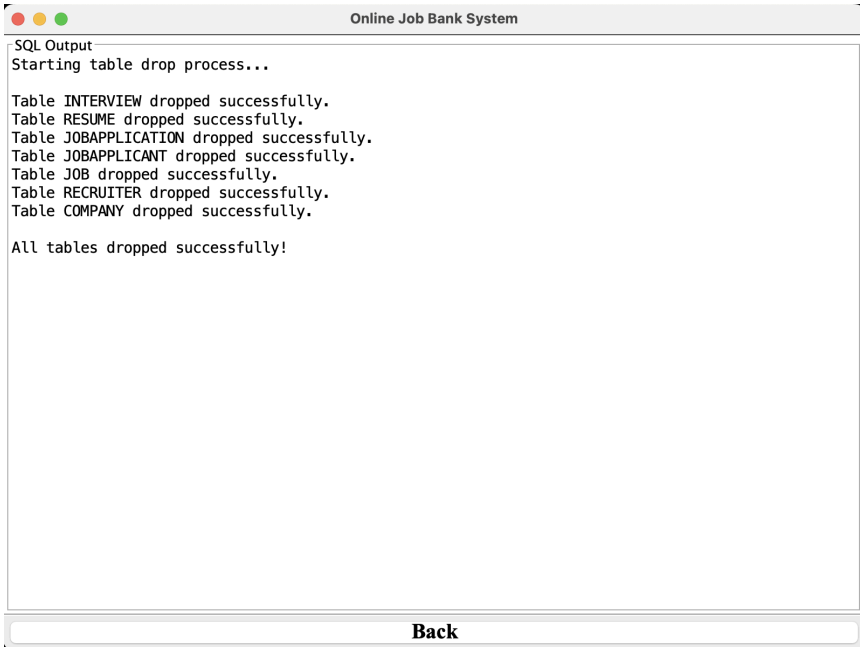
6. To drop the tables, click “Drop Tables”.
   a. You will be directed to the Drop Tables page, which has 3 options: Drop Tables, View Output, and Back, which takes you back to the home page.



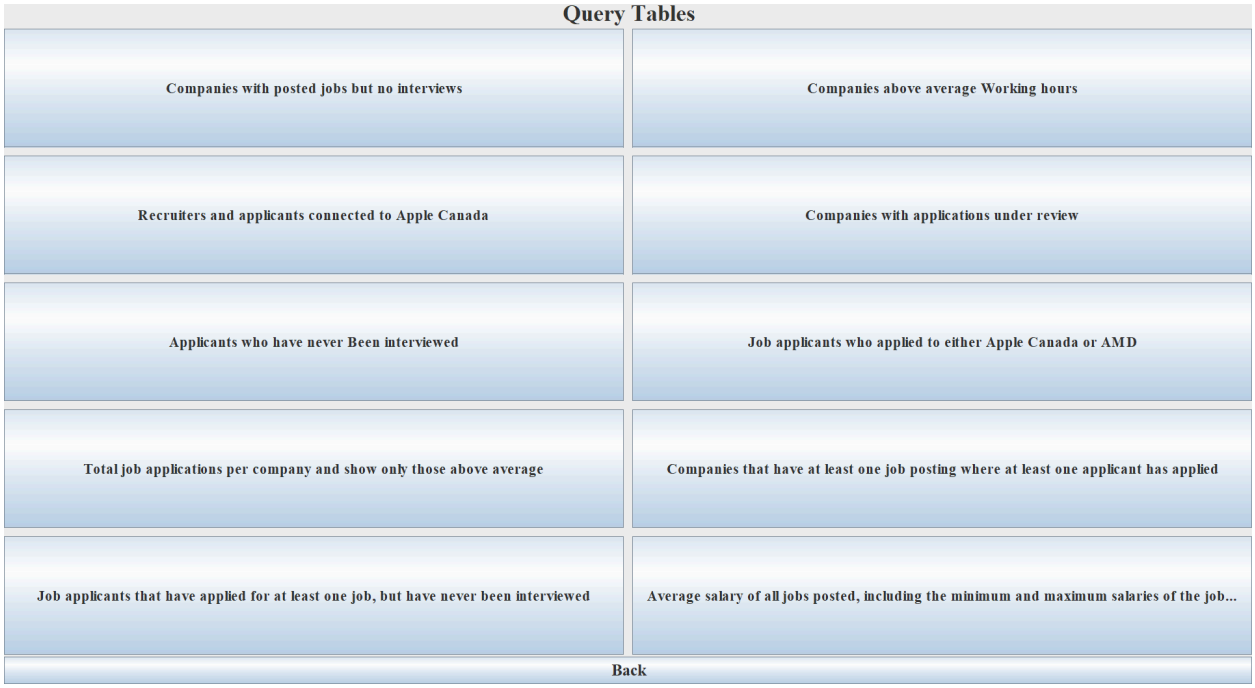
   b. Click Drop Tables to drop the tables.
   c. If successful, then a pop-up will appear telling you that the tables were dropped successfully. It will also display the results of the tables being dropped, similar to what you see in SQLDeveloper.



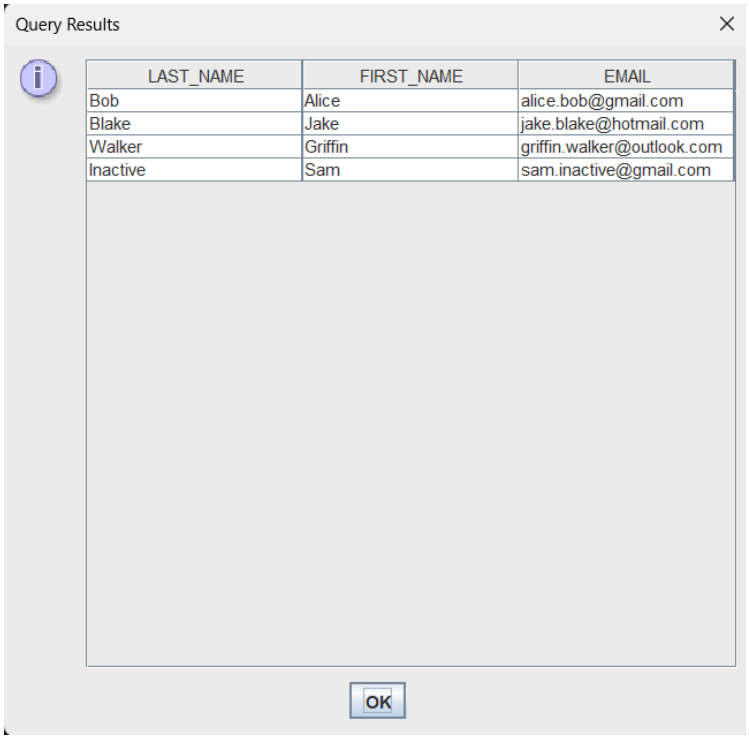
d. If unsuccessful, then a pop-up will appear telling you that the tables were not dropped due to an error, along with the error message.
e. If you click “View Output”, you will see a full screen of the results of the tables being dropped.



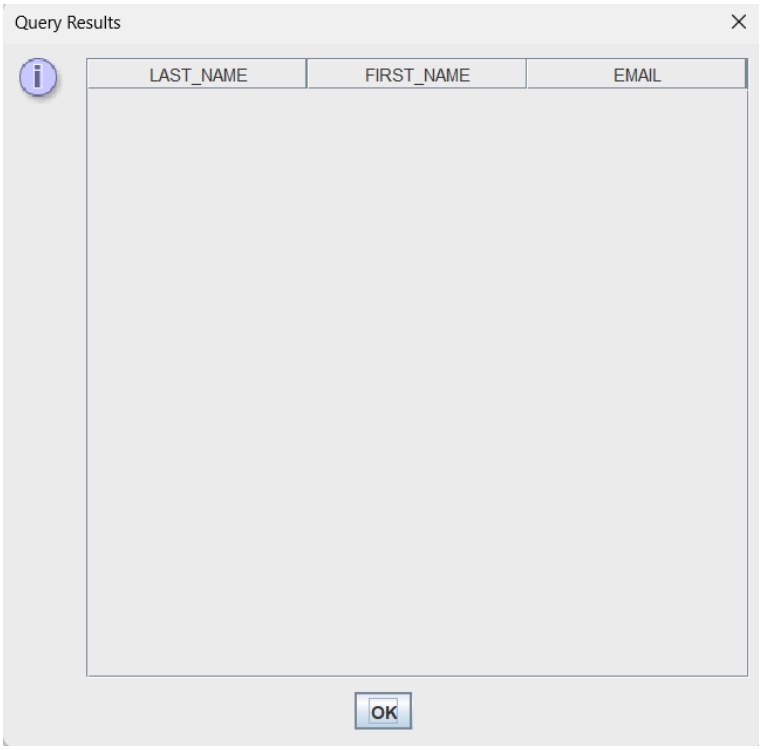
f. Click Back to go back to the home page/main menu.



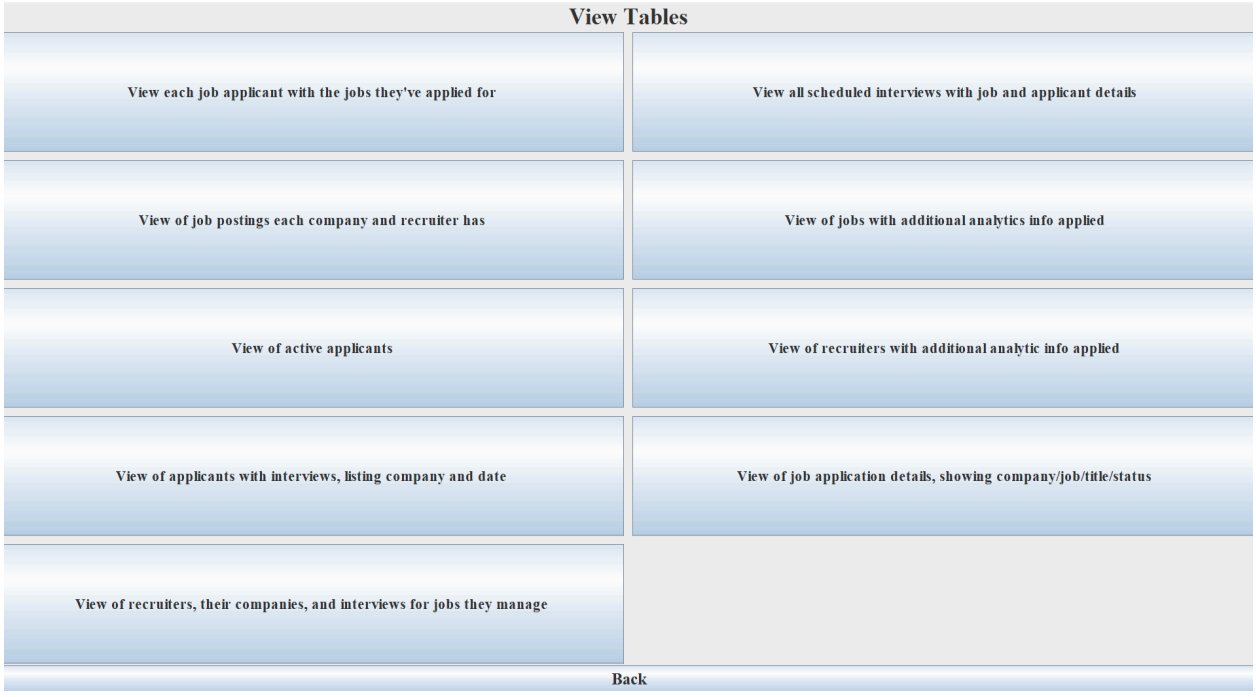
7. Click Query tables to query the tables
    a. After selecting a query, if successful, then a popup will appear which will show the queried information. After clicking ok, another popup will appear which will say “query executed successfully!”



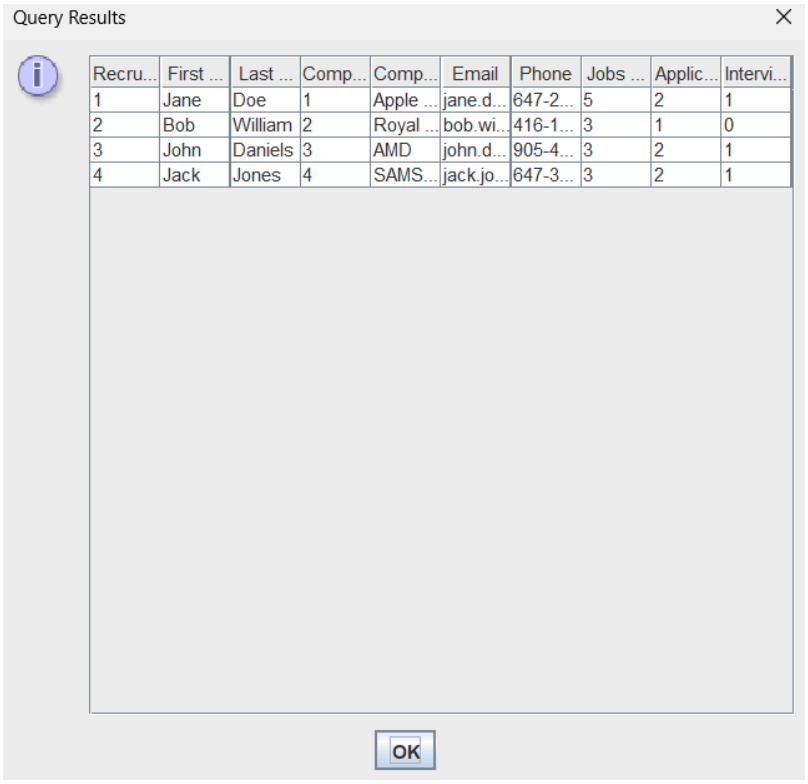
    b. If tables are not yet created, then error message will popup when attempting to query the database
    c. If tables are not yet populated, then query results will be blank.



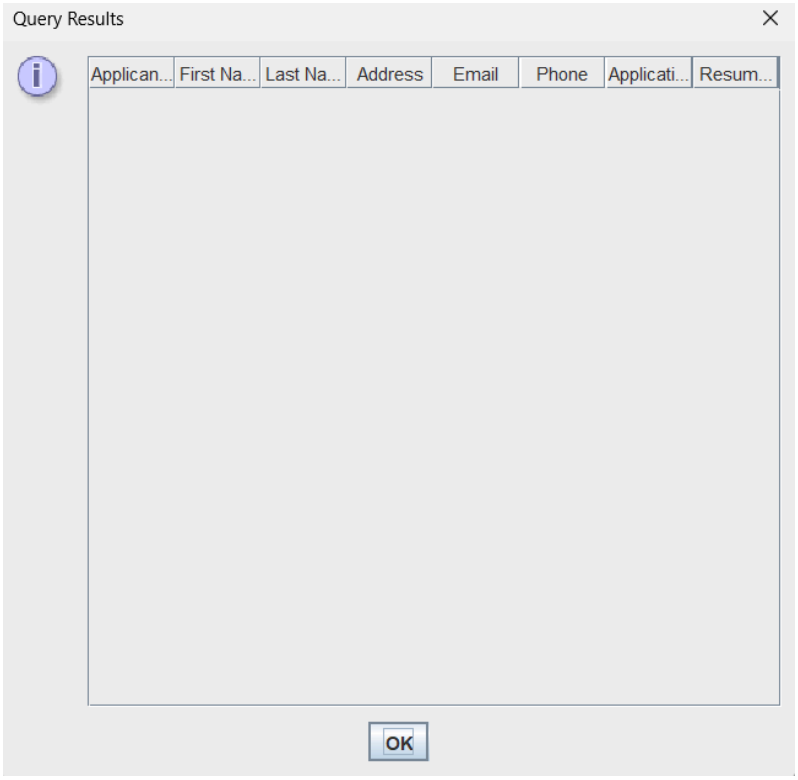
d. Click Back to go back to the home page/main menu.



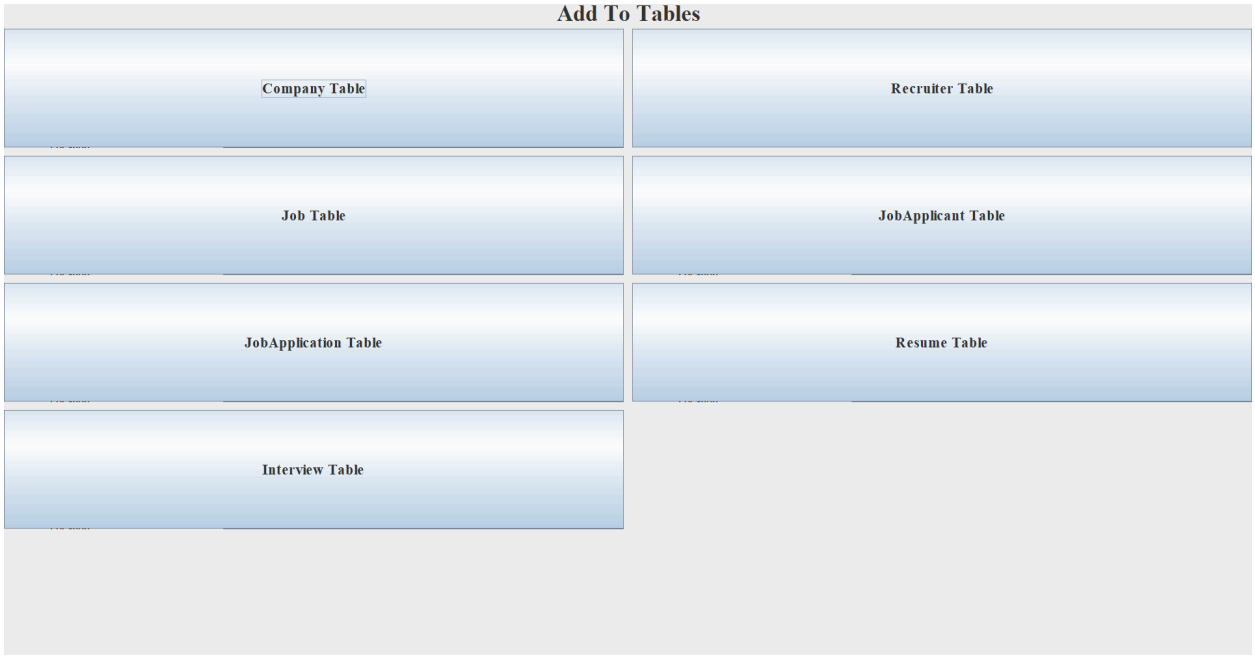
8. Click View tables to select a view for the tables
    a. After selecting a view, if successful, then a popup will appear which says “view created successfully, now showing view and then the information of the viewed query will be displayed. After clicking ok, another popup will appear which will say “view displayed successfully!”



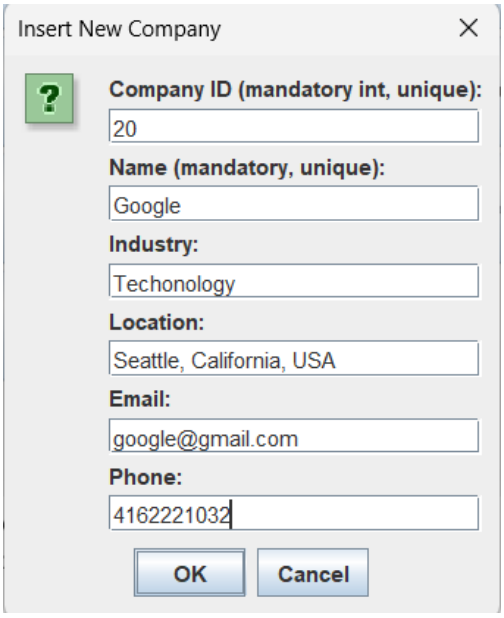
b. If tables are not yet created, then an error message will popup when attempting to query the database
c. If the tables are not yet populated, then the view results will be blank



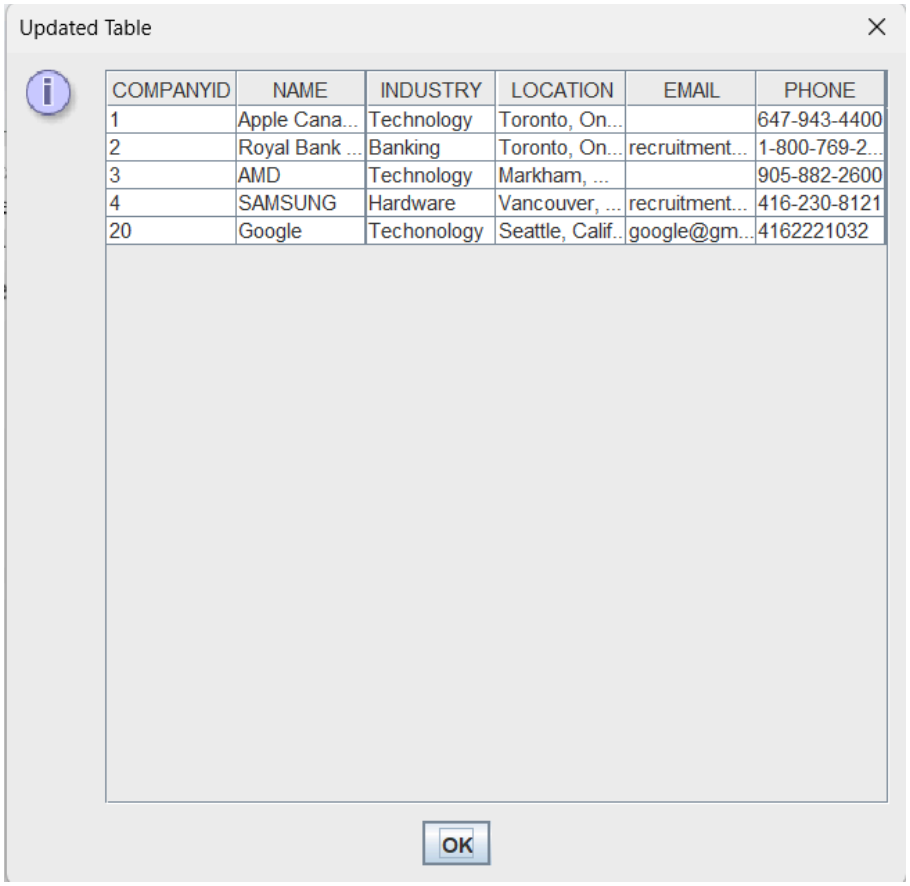
d. Click Back to go back to the home page/main menu.



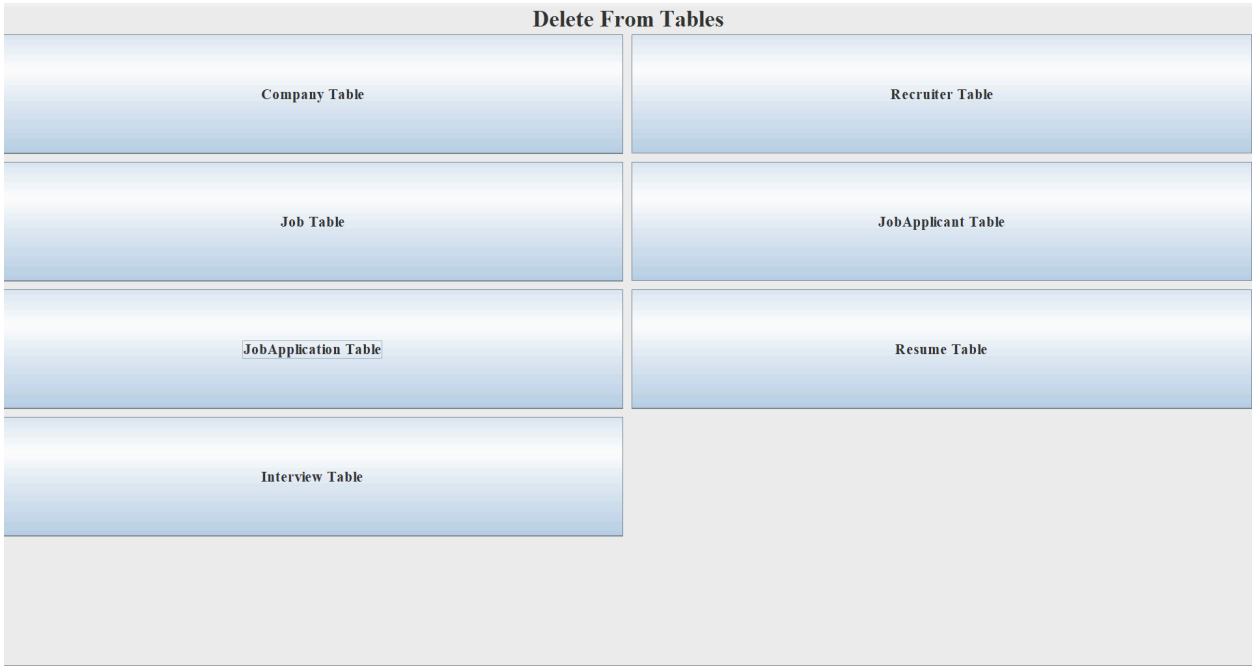
9. Add to tables to add records to a specific table
    a. Select a table and input the required information needed for the table. If successful, a message will pop up saying that the record was added successfully



    b. After clicking OK, the newly updated table will be displayed with all the records

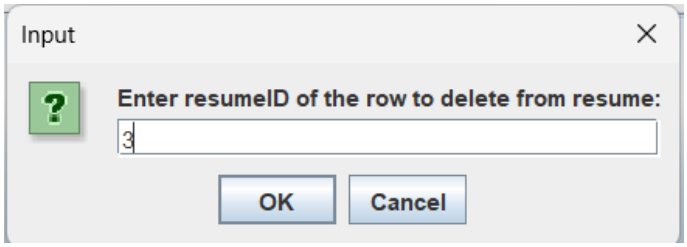


c. If no tables exist, error message will popup saying “the table does not exist”
d. Click Back to go back to the home page/main menu.

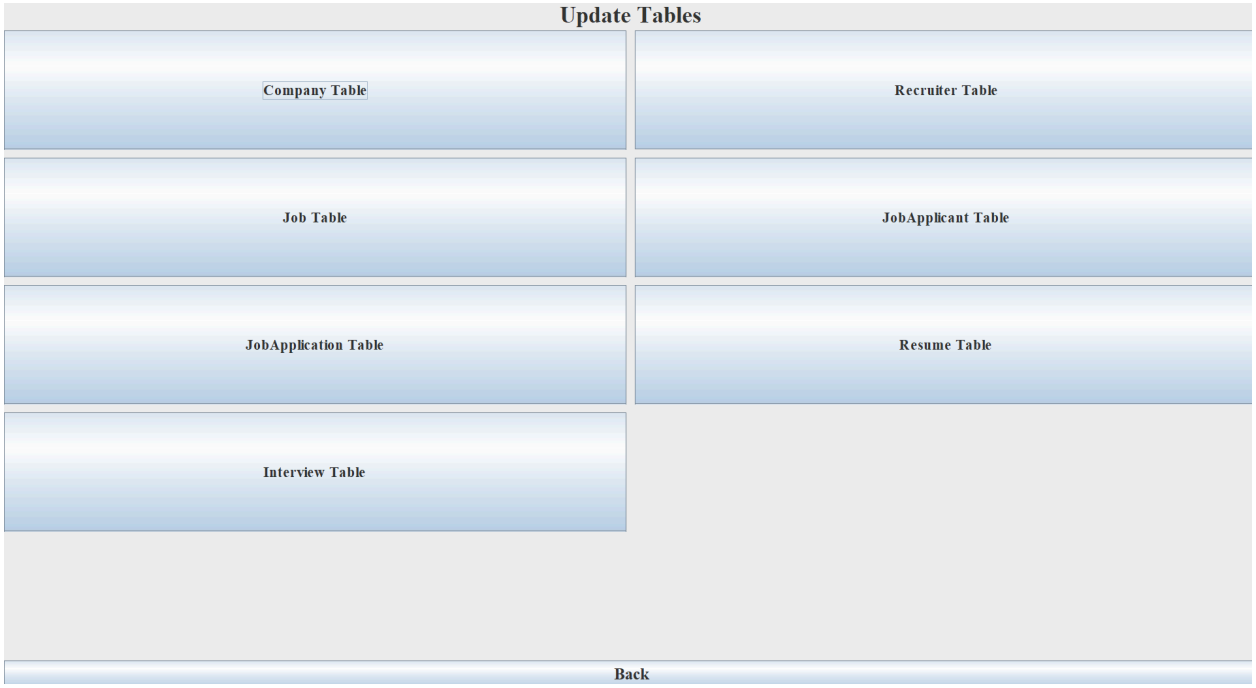


10. Delete from tables to delete a record from a specific table

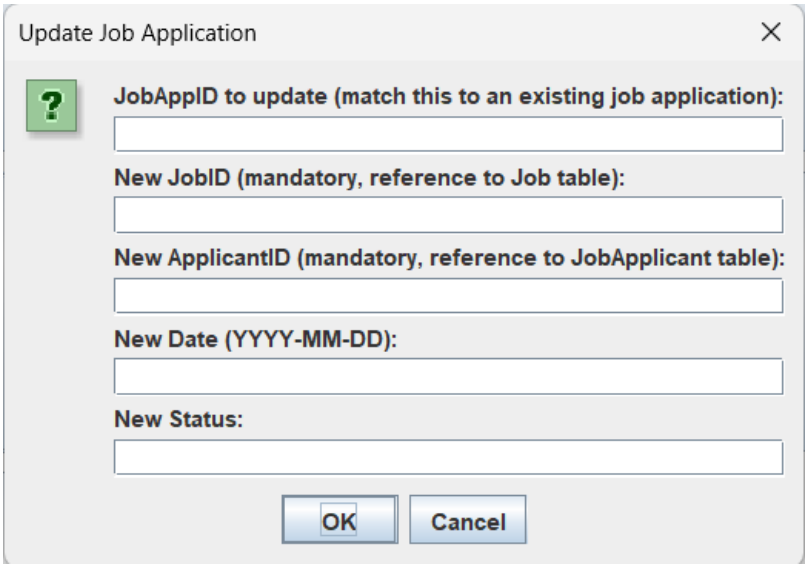
a. After selecting the table of the record you would like to delete, enter the id of the record to delete it from the database. After entering a valid id, a message will appear saying “record deleted successfully!” Once deleted, the application will display all records after deleting.



b. If an invalid identifier is entered, then an error will popup which says “no record found with the given id.” An error message will popup if deletion of the selected record id will cause an integrity constraint.
c. Click Back to go back to the home page/main menu.

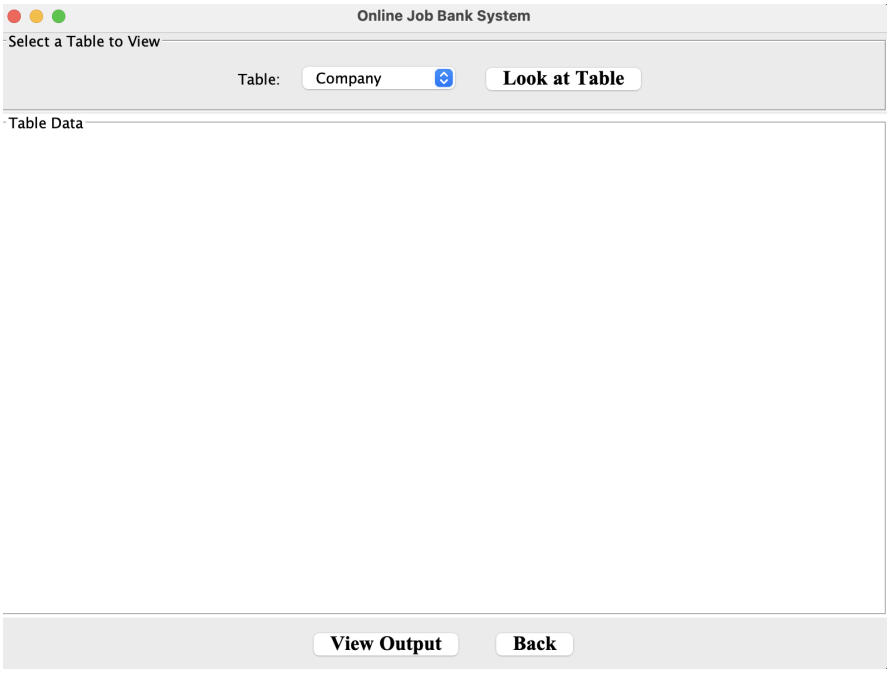


11. Update tables to update a record from a specific table
    a. After selecting the table of the record you would like to delete, enter the id of the record to update it. After entering a valid id, you can update the attributes of the record. A message will pop up saying the record was updated successfully. After updating the record, the application will display all the records after updating.

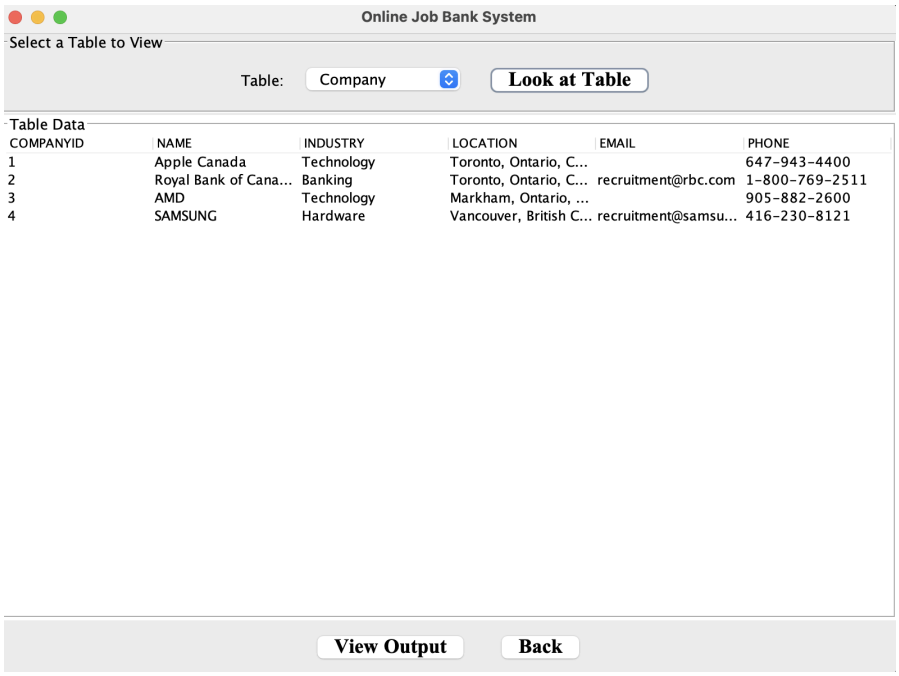


b. If any invalid identifiers or attribute fields are entered, then error messages will be displayed.
c. Click Back to go back to the home page/main menu.

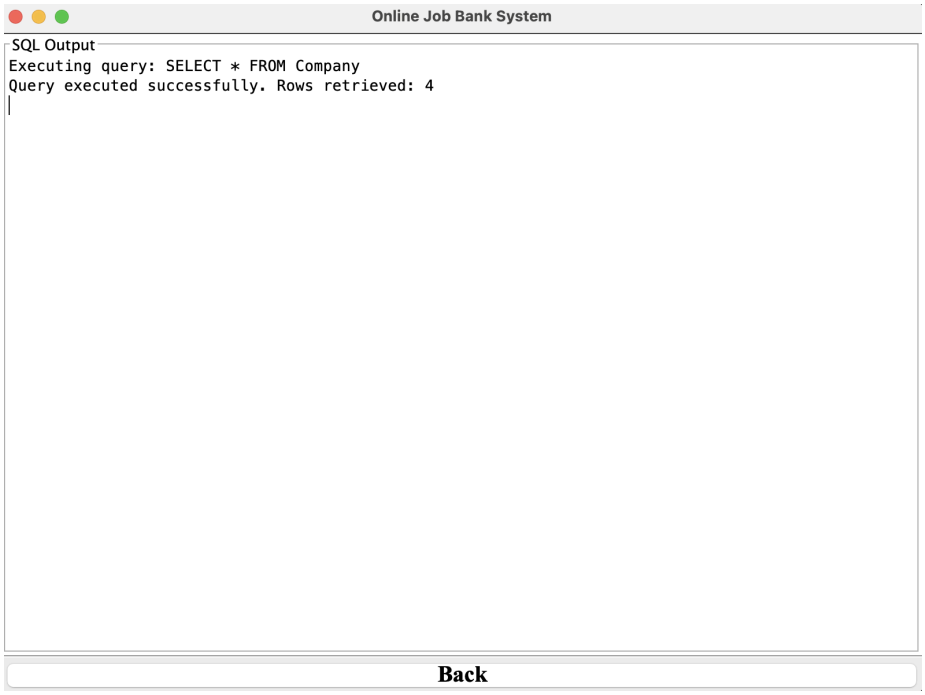
12. To read records in a table, click Look at Tables
    a. At the top of the screen, you will be given the option to select the table in the database that you want to look at.
    b. Once you have selected the table you want to look at, click the “Look at Table” button to view the table.



    c. Once you click the “Look at Table” button, you will see all the records that are in the table.



d. Click View Output to see the response from SQL after selecting the table you want to see.

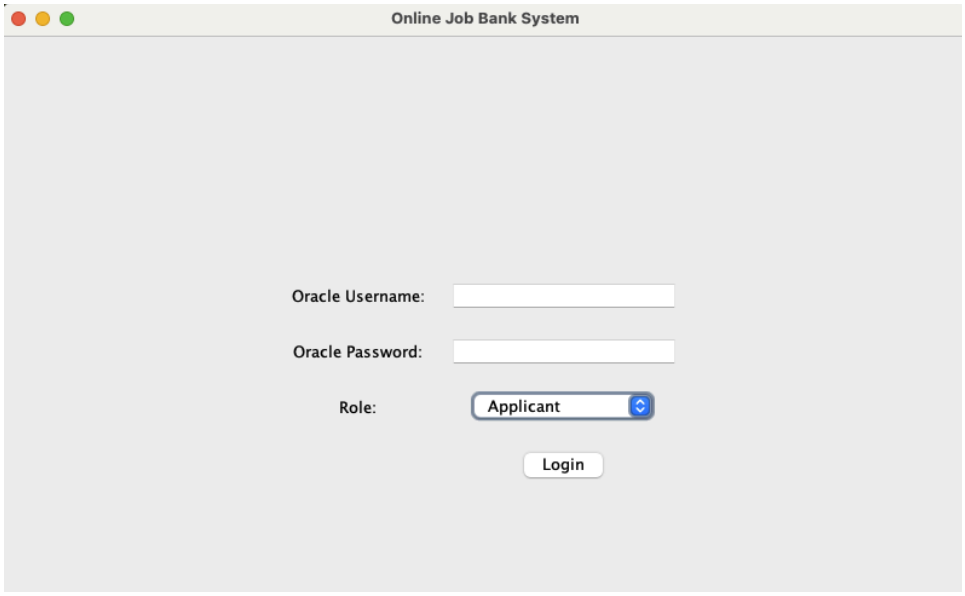


e. Click Back to go back to the home page/main menu.

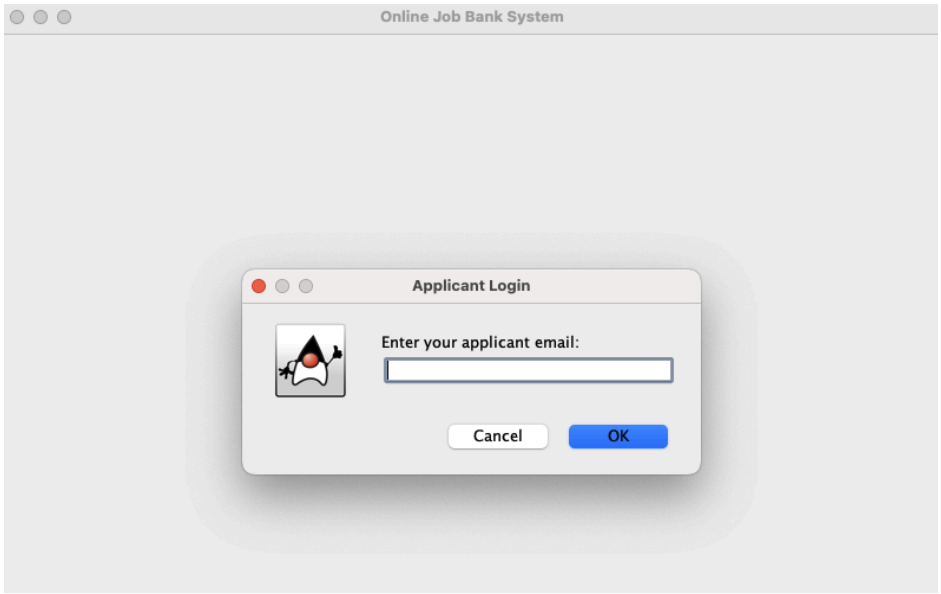
13. Click Exit to exit the application.

<u>If User Selects “Applicant” Role</u>

1. When the application launches, the login window appears. This is where you enter your Oracle credentials (username and password) and select the role as “**Applicant**”.

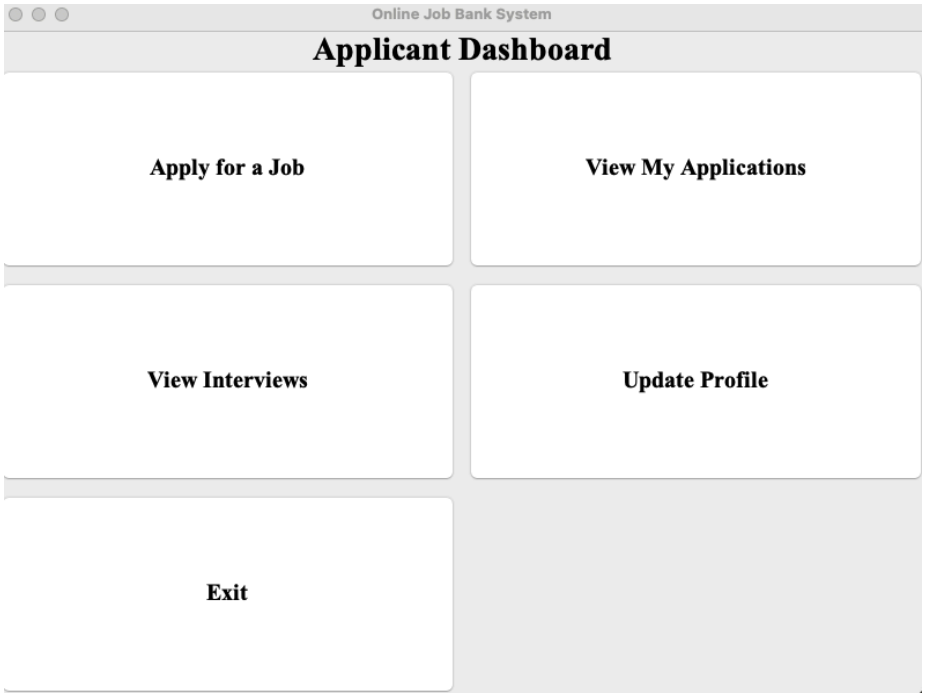


2. Once you have entered your Oracle username and password in the correct format, click the Login button.
    a. If the username and password are in the correct format, then a pop-up will appear telling you to enter the applicant's email address (e.g. alice.bob@gmail.com)

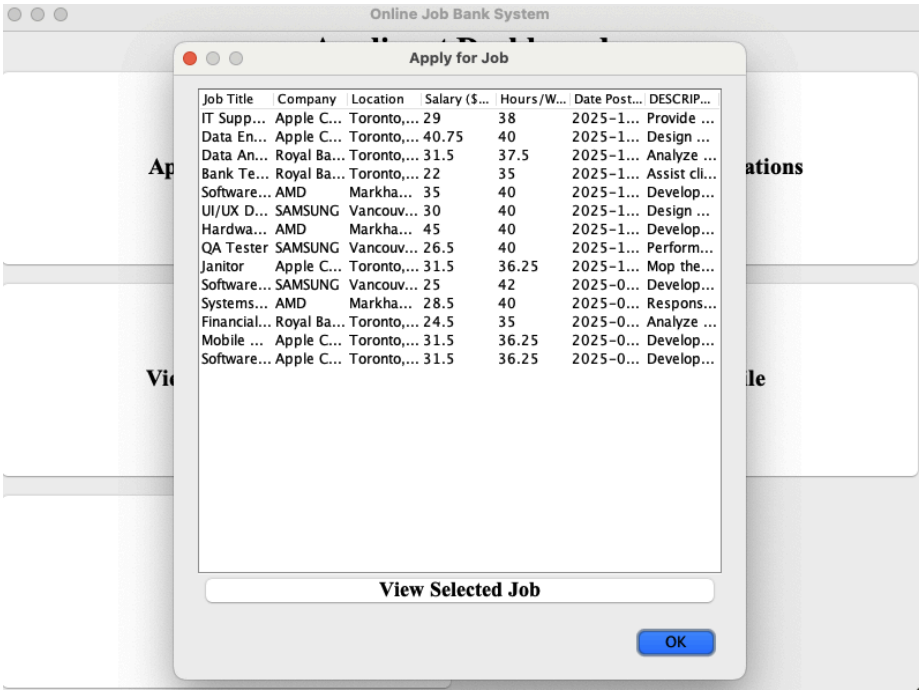


    i. If you enter an applicant email that is in the database, then the login is successful and you will be directed to the home page.
    ii. If you enter an applicant email that is not in the database, then you will get an error message saying that the applicant email is not in the database and you will have to login again.

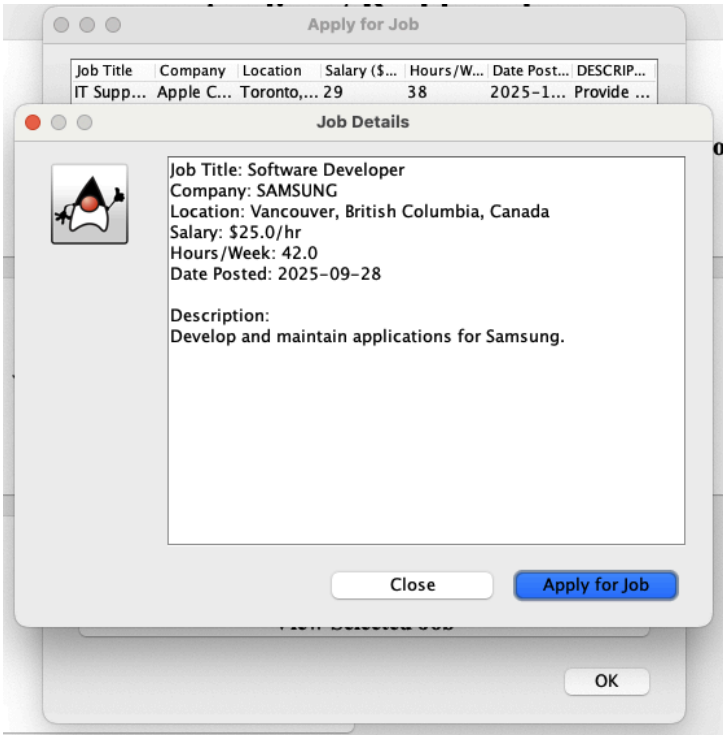
    b. If the username and password are not in the correct format, then a pop-up will appear telling you that the login is unsuccessful with an error message. When you click “Ok”, you will be prompted to re-enter your username and password.
3. After logging in, you will be directed to the home screen, which consists of many features, such as Apply for a Job, View My Applications, View Interviews, Update Profile, and Exit.



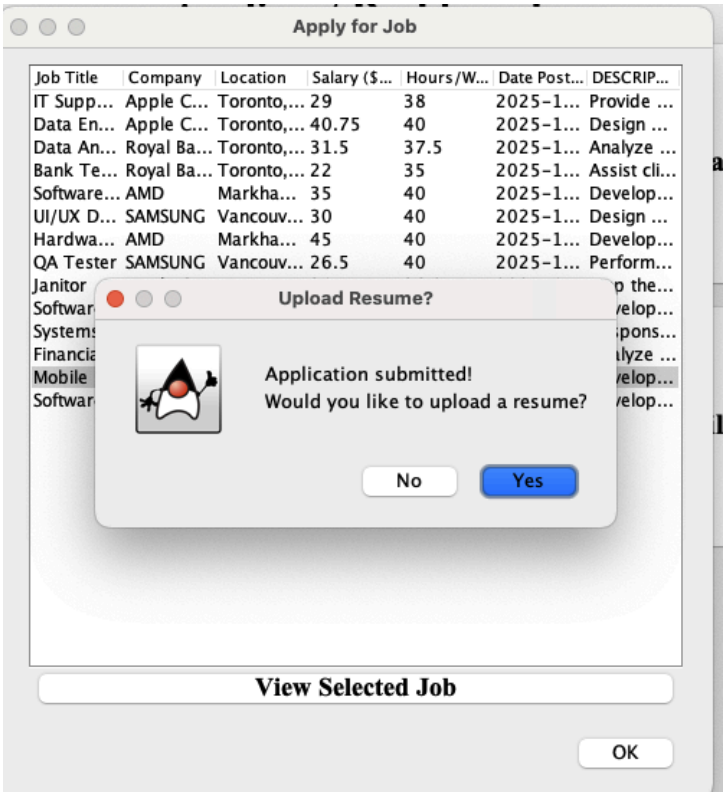
4. To apply for a job, click “Apply for a Job”.
    a. A list of the job postings at each company will be displayed, showing you each job, which consists of a job title, location (city) of the job, salary ($/hr), hours per week, when the job was posted, and the description of the job.



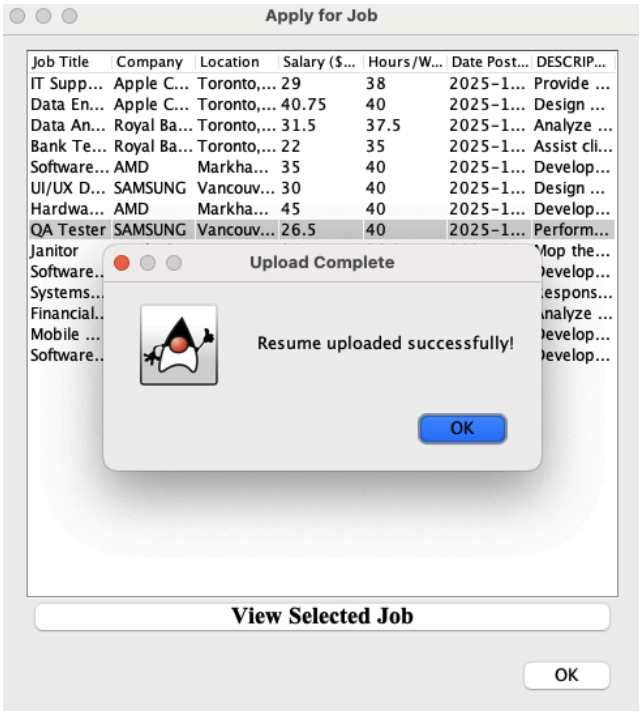
b. To view the job, click on a job listing and then click on the “View Selected Job” button, which will display the full job details.



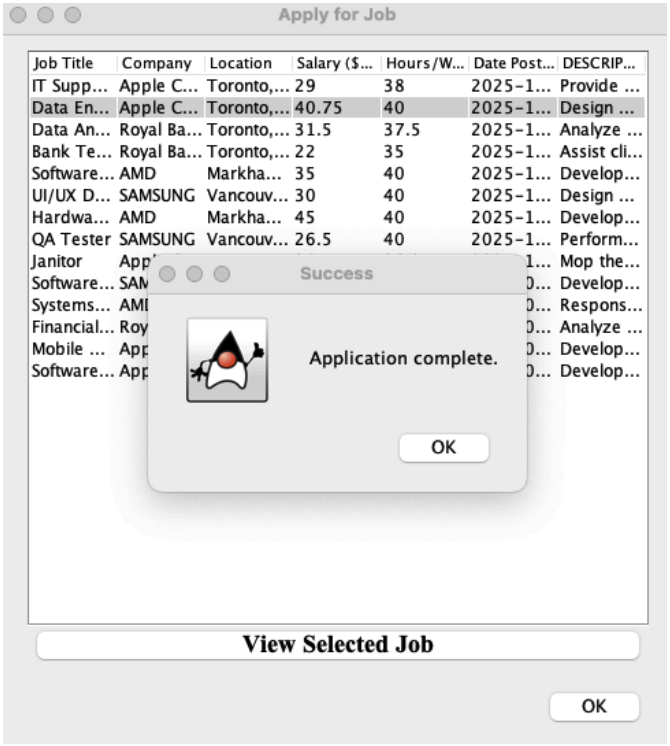
c. To apply for the job, click “Apply for Job”.
    i. A popup will show up saying that the application was submitted. It will ask you if you want to upload a resume.



1. If you want to upload a resume, click “Yes”, and then upload a file (resume) from your device.
    a. Once you have chosen a file to upload, click “Open”, and a popup will appear showing that the file upload was successful.



b. Another popup will show up saying that the application was complete.

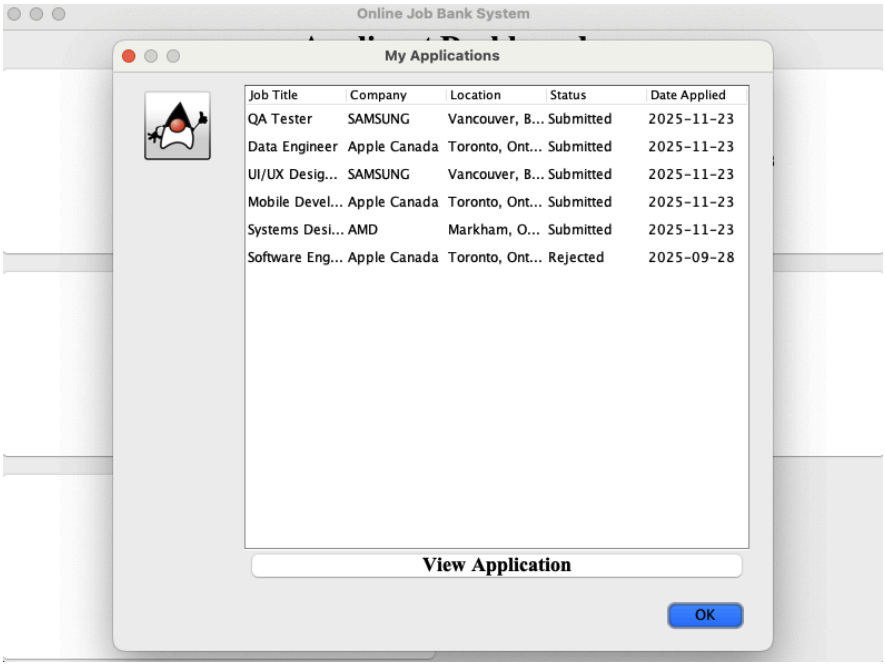


2. If you click “No”, then a popup will appear saying that the application was complete.

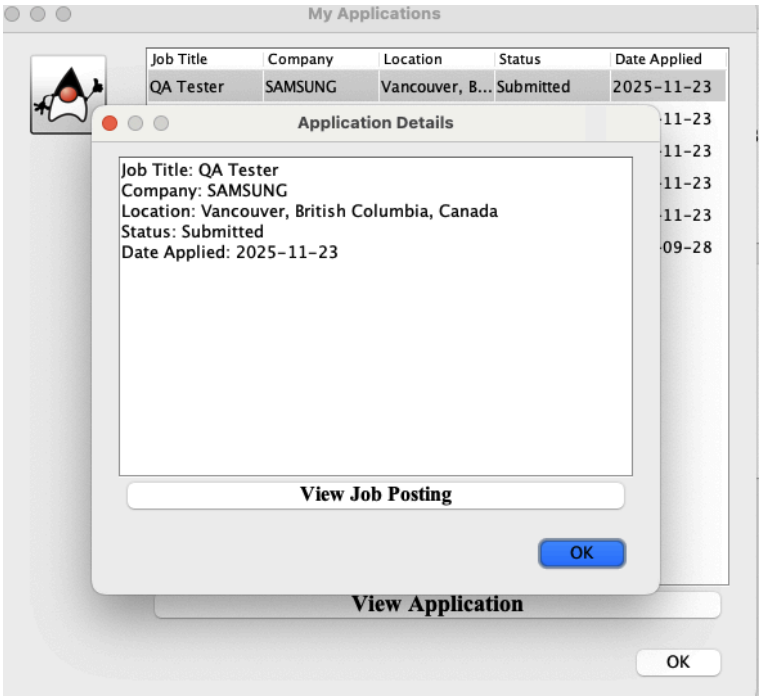
d. To go back to the job postings list, click “Close”.

e. Click “Ok” to go back to the home screen.

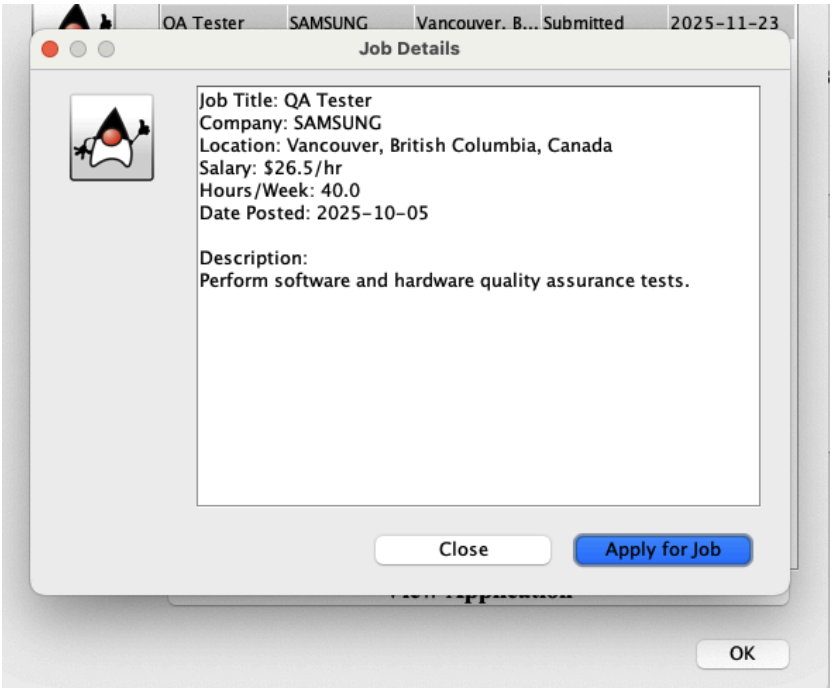
5. To view your submitted job applications, click “View My Applications”.
    a. A list of your submitted applications for each job will be displayed, showing you each application, which consists of the job title, company, location (city), application status, and the date applied.



b. To view the application, click on a job application and then click on the “View Application” button, which will display the full job application details.

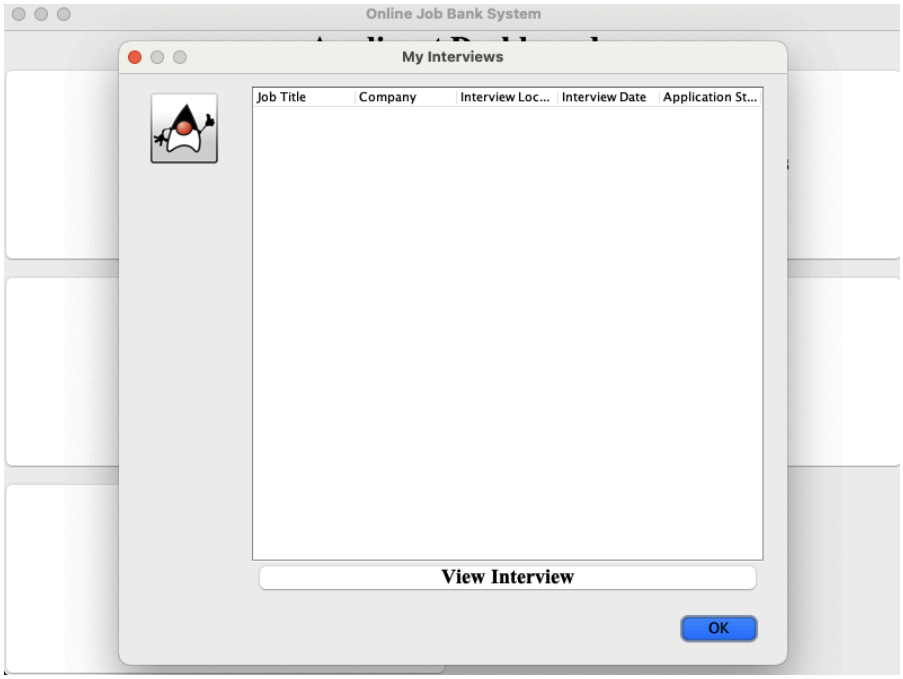


c. To view the job posting the application is for, click “View Job Posting” and the full job details of the job will be displayed.



d. Click “Close” and “Ok” to go back to the list of job applications.
e. Click “Ok” to go back to the main menu.

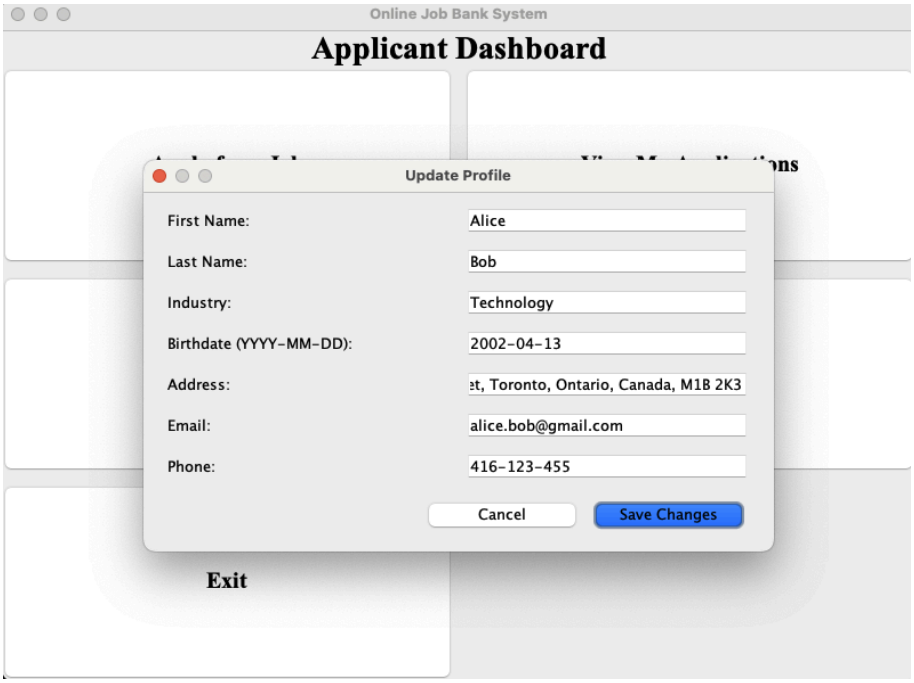
6. To view your upcoming job interviews, click “View Interviews”.
   a. A list of upcoming interviews (if any) will be displayed, showing you each interview, which consists of the job title that the interview is for, company, interview location, date of the interview, and the associated job application status.



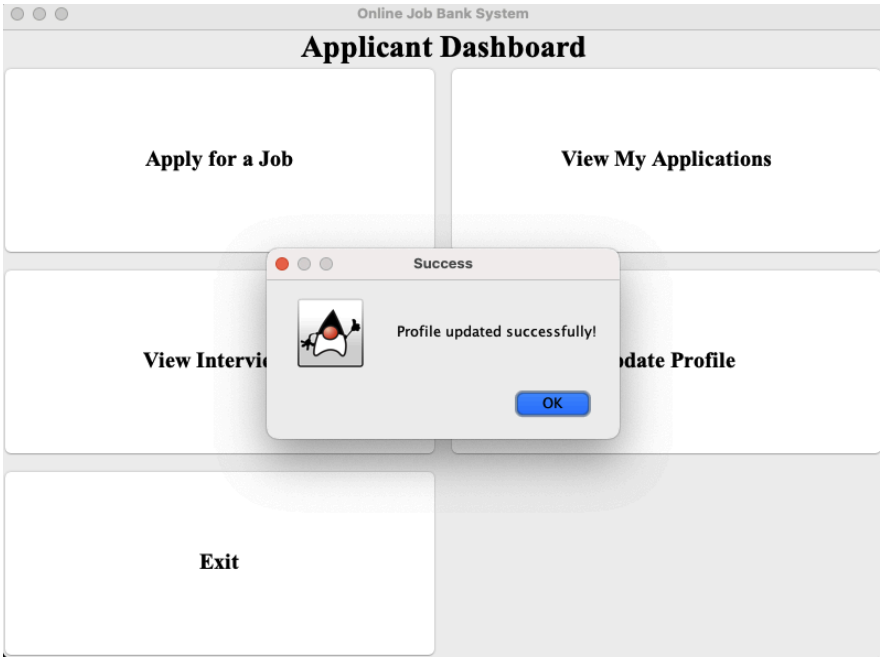
   b. To view the interview, click on an interview and then click on the “View Interview” button, which will display the full interview details.

c. To view the job posting the interview is for, click “View Job Posting” and the full job details of the job will be displayed.
d. Click “Close” and “Ok” to go back to the list of upcoming interviews.
e. Click “Ok” to go back to the main menu.

7. To update your information (applicant details), click “Update Profile”.
   a. A window will open containing of your (applicant) information, which consists of first name, last name, industry, birthdate, address, email, and phone number.



   b. Make any changes to the information if needed.
   c. Once you have made your changes, click “Save Changes”.



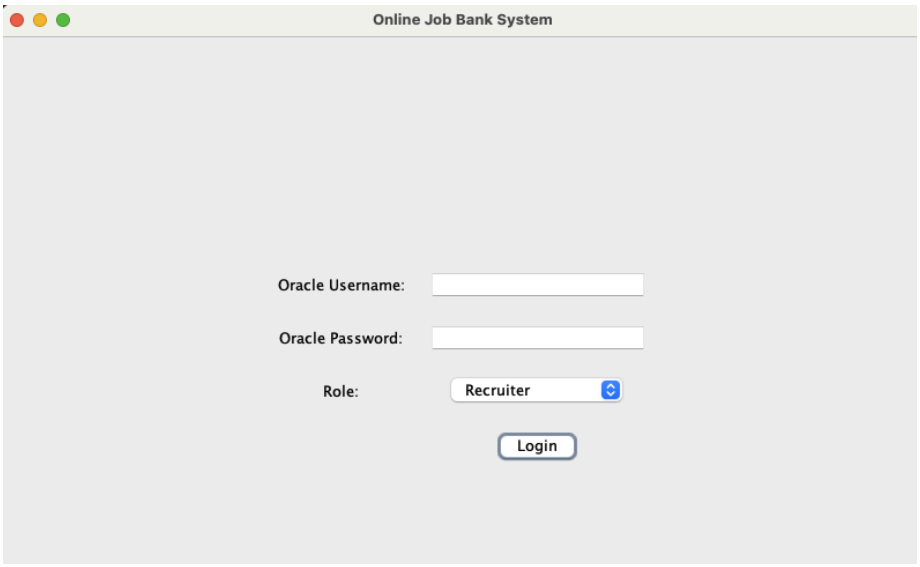
i. A popup will appear saying that the profile has been updated successfully. Click “Ok” to return to the main menu.

d. If you did not make any changes, then click “Cancel” and you will go back to the main menu.

8. Click Exit to exit the application.

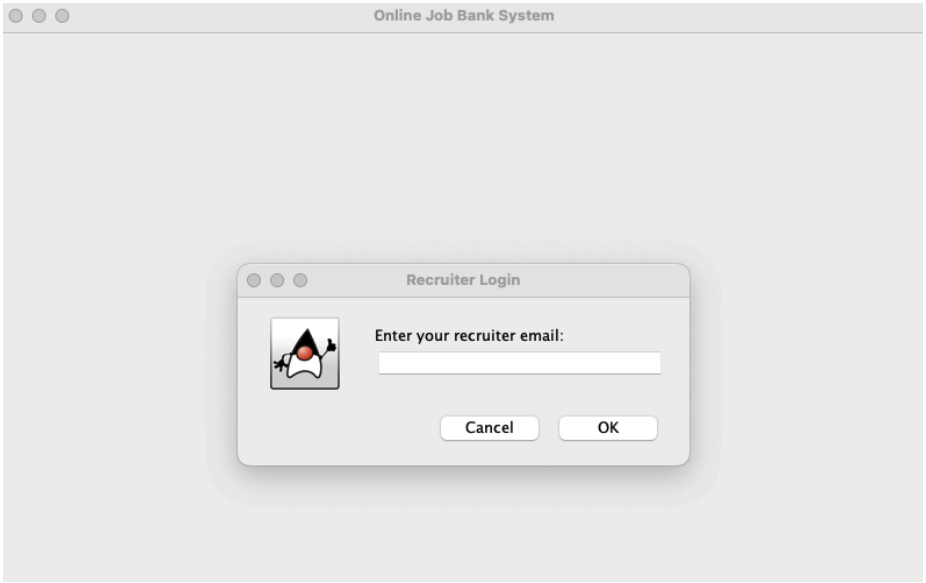
If User Selects “Recruiter” Role

1. When the application launches, the login window appears. This is where you enter your Oracle credentials (username and password) and select the role as “**Recruiter**”.



2. Once you have entered your Oracle username and password in the correct format, click the Login button.

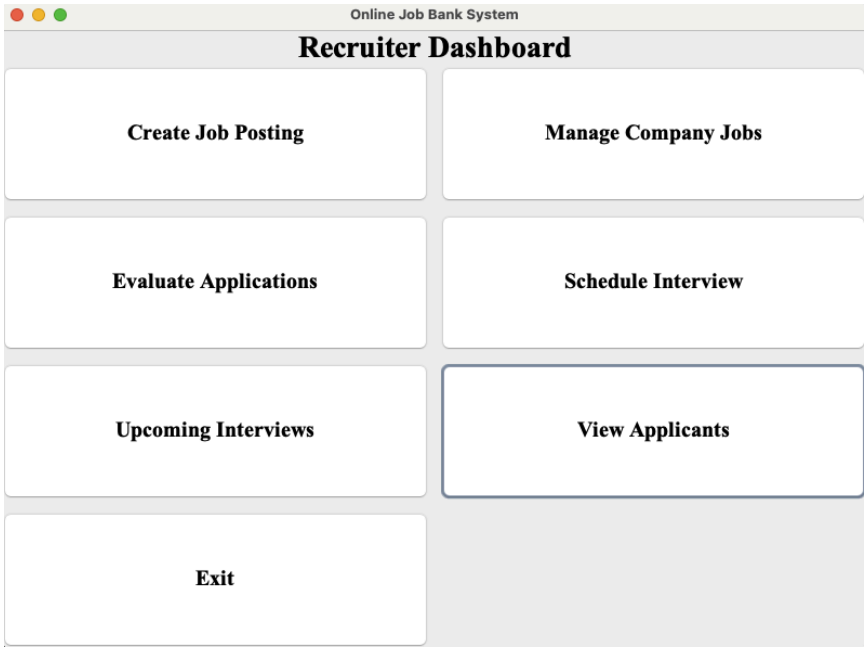
a. If the username and password are in the correct format, then a pop-up will appear telling you to enter the recruiter’s email address (e.g. jane.doe@apple.com)



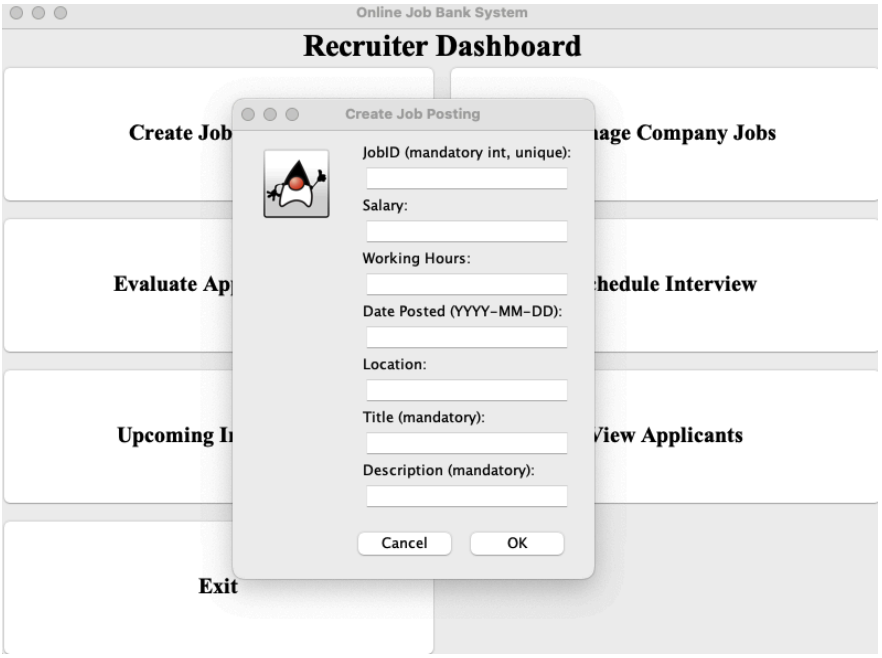
i. If you enter a recruiter email that is in the database, then the login is successful and you will be directed to the home page.
ii. If you enter a recruiter email that is not in the database, then you will get an error message saying that the recruiter email is not in the database and you will have to login again.

b. If the username and password are not in the correct format, then a pop-up will appear telling you that the login is unsuccessful with an error message. When you click “Ok”, you will be prompted to re-enter your username and password.

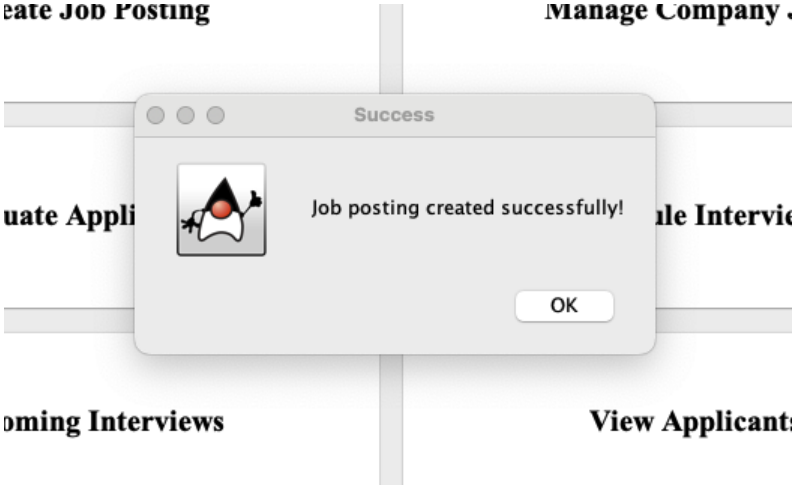
3. After logging in, you will be directed to the home screen, which consists of many features, such as Create Job Posting, Manage Company Jobs, Evaluate Applications, Schedule Interview, Upcoming Interviews, View Applicants, and Exit.



4. To create a new job listing, click “Create Job Posting”.
    a. A new window will open asking you to enter all details about the job, which includes JobID, Salary, Working Hours, Date Posted, Location, Title, and Description.

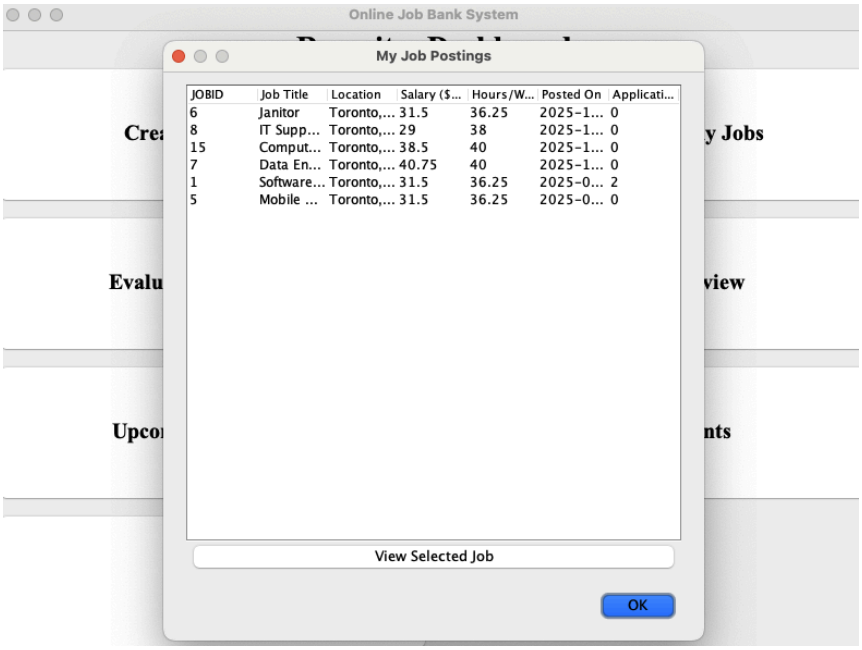


b. Once you have entered all the information, click “Ok”.
c. The job will be added to the database, with a popup saying that the job posting was created successfully.



d. Click “Ok” to go back to the main menu.

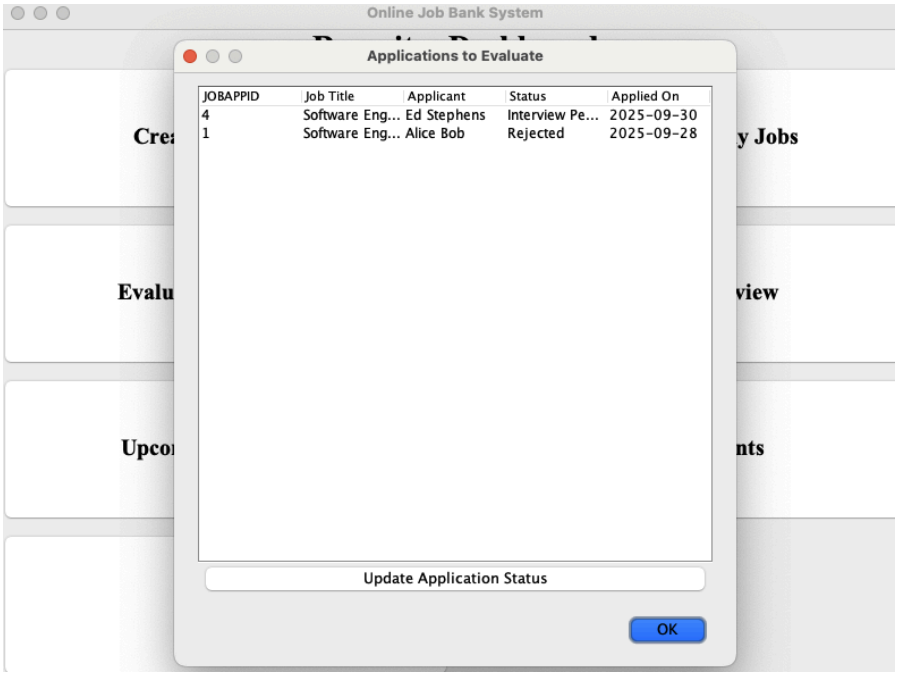
5. To look at the jobs posted at the company, click “Manage Company Jobs”.
   a. A list of the job postings at the company will be displayed, showing you each job, which consists of the jobID, job title, location (city) of the job, salary ($/hr), hours per week, when the job was posted, and the number of applicants that have applied for the job.



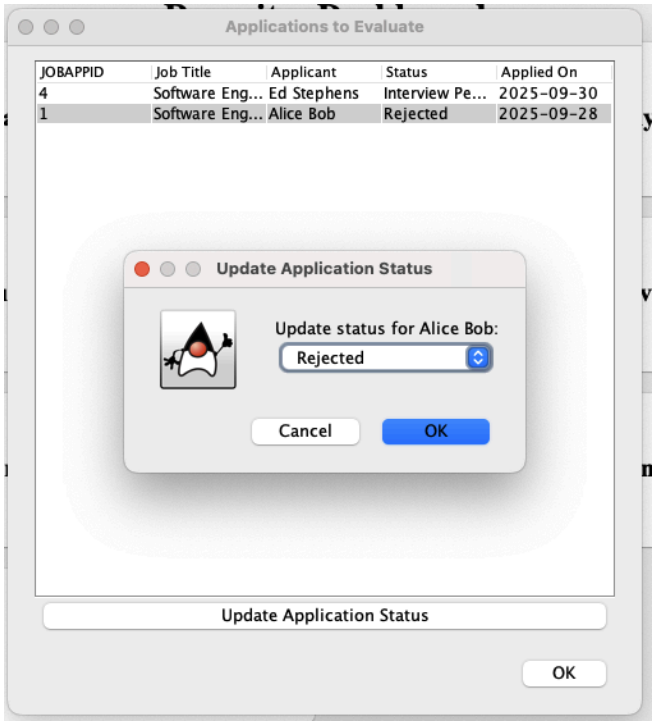
b. To view the job, click on a job listing and then click on the “View Selected Job” button, which will display the full job details.

c. Click “Ok” to go back to the home screen.

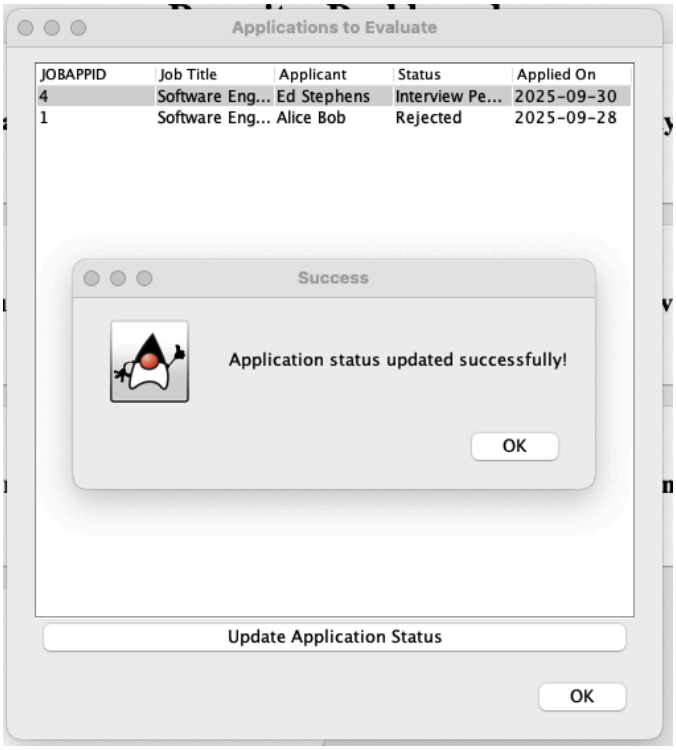
6. To evaluate job applications for job postings, click “Evaluate Applications”.
   a. A list of job applications will be displayed, showing you each application, which consists of the JobAppID (job application ID), the job title the application is for, the applicant that applied for the job, the application status, and the date that they applied (submitted the application).



b. To update a job application status, select a job application from the list and click “Update Application Status”.

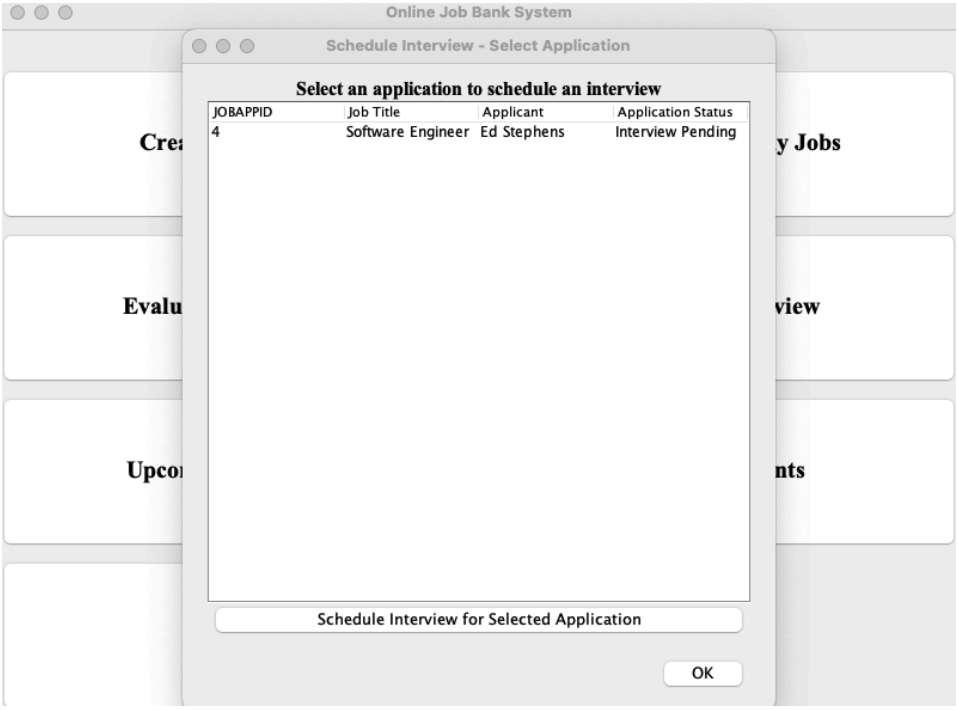


c. You can change the status of the job application to one of the following: Submitted, Under Review, Interview Pending, and Rejected.
d. Once you have selected the status, click “Ok” and a popup will appear saying that the status was updated successfully.

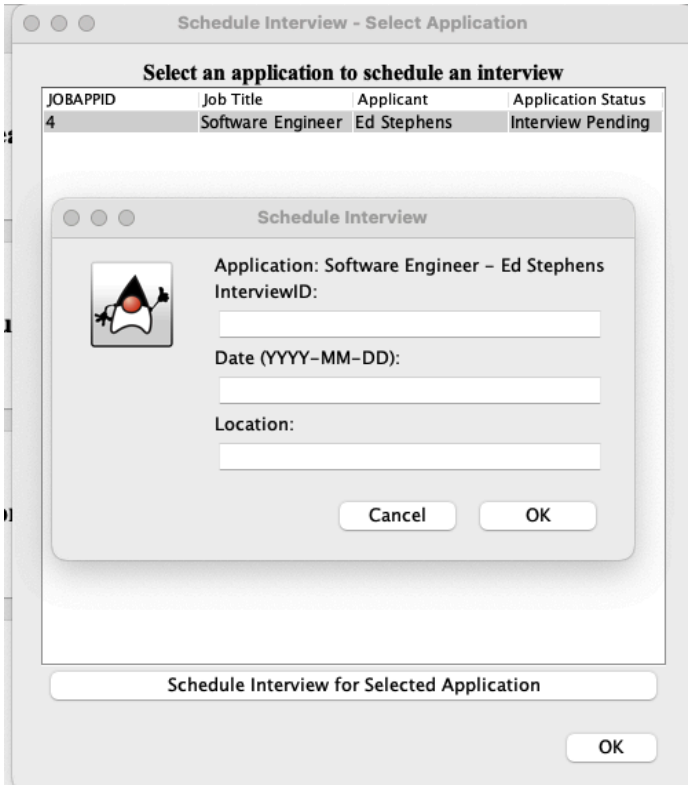


e. Click “Ok” to return to the main menu.

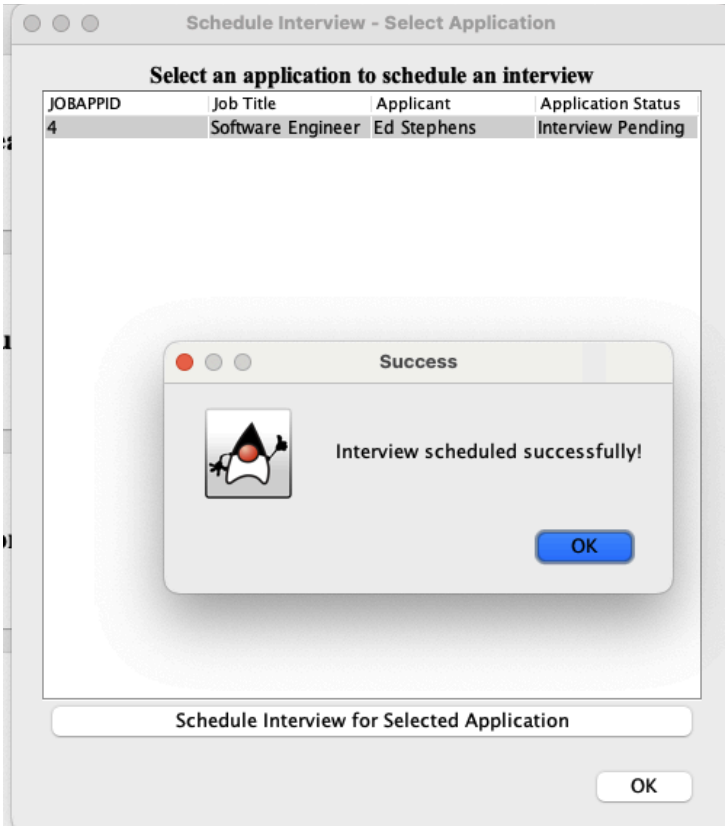
7. To schedule interviews for job applications, click “Schedule Interview”.
    a. A list of job applications will be displayed, showing you each application, which consists of the JobAppID (job application ID), the job title the application is for, the applicant that applied for the job, and the job application status.



b. To schedule an interview for a job application, select a job application from the list and click “Schedule Interview for Selected Application”.
c. A new window will open asking you to enter in details about the interview, such as the interviewID, date of the interview, and location of the interview.

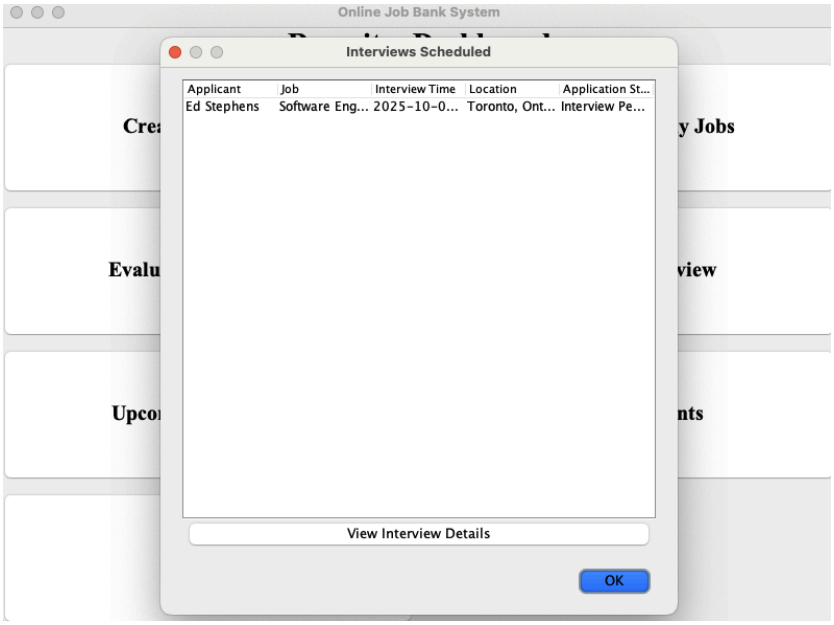


d. Once you have entered in all the information and click “Ok”, the interview will be created, as a popup will appear saying that the interview was scheduled successfully.

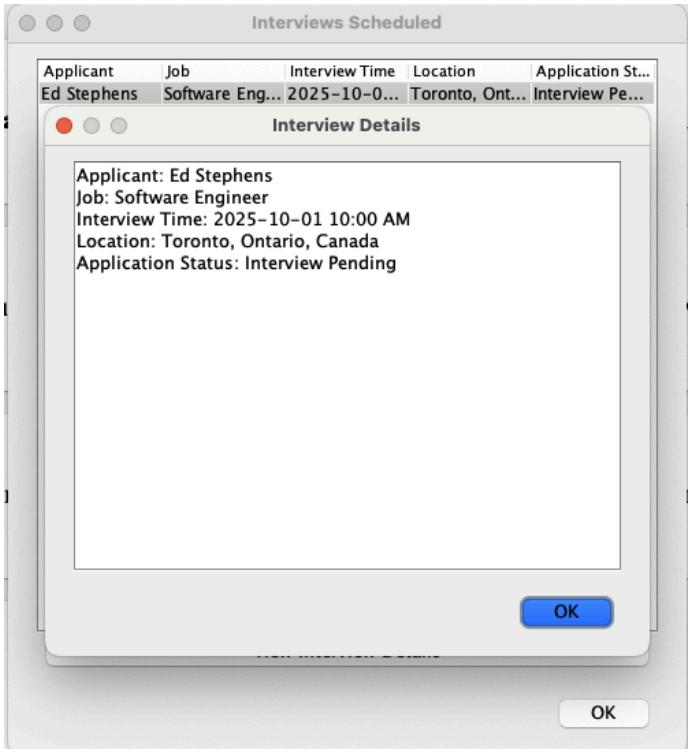


e. Click “Ok” to return to the main menu.

8. To view upcoming scheduled interviews, click “Upcoming Interviews”.
    a. A list of upcoming interviews for job applications will be displayed, showing you each interview, which consists of the applicant that the interview is for, the job they are interviewing for, interview date and time, location of the interview, and their job application status.

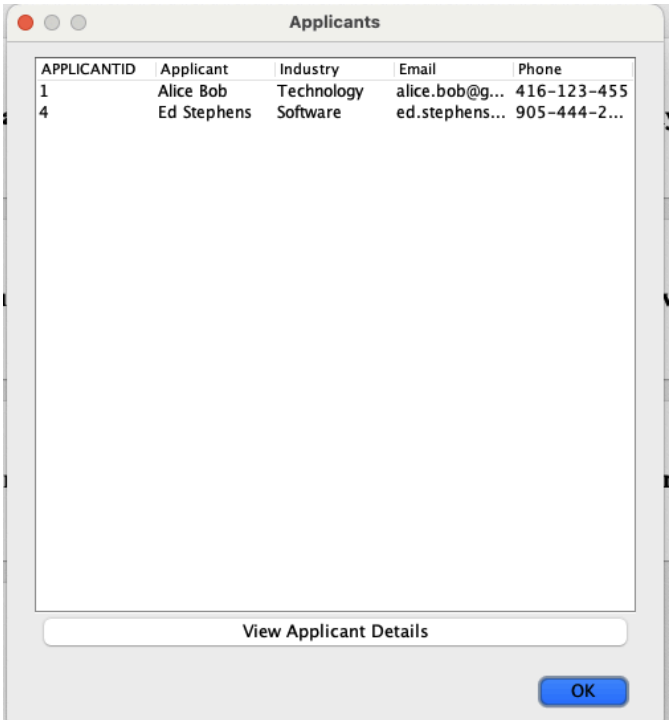


b. To view the interview, click on an interview and then click on the “View Interview Details” button, which will display the full interview details.



c. Click “Ok” to go back to the home screen.

9. To view applicants that have applied for jobs at the company, click “View Applicants”.
    a. A list of the job applicants that applied for jobs at the company will be displayed, showing you each applicant, which consists of their applicantID, name, industry, email, and phone number.



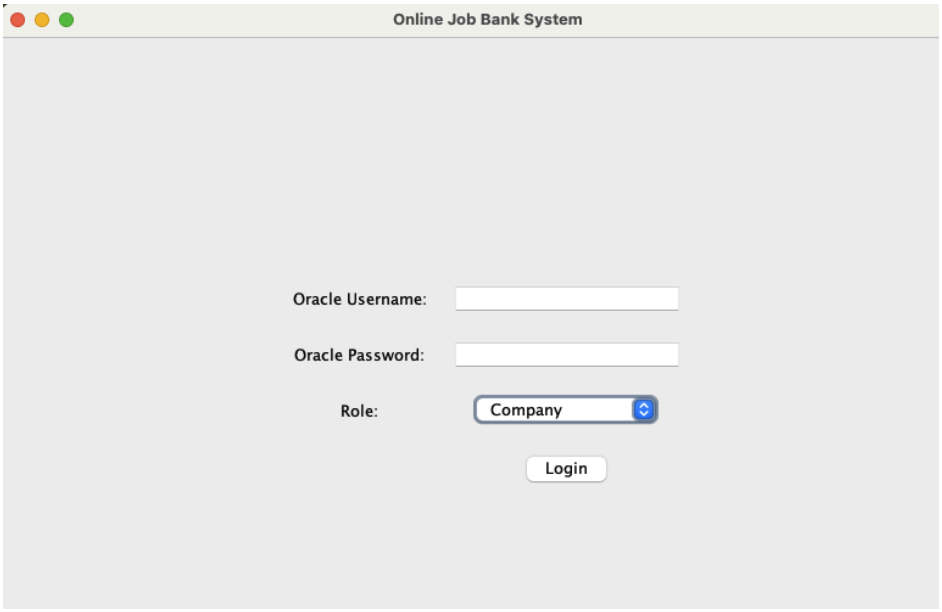
b. To view the applicant, click on an applicant and then click on the “View Applicant Details” button, which will display the full applicant details.

c. Click “Ok” to go back to the home screen.

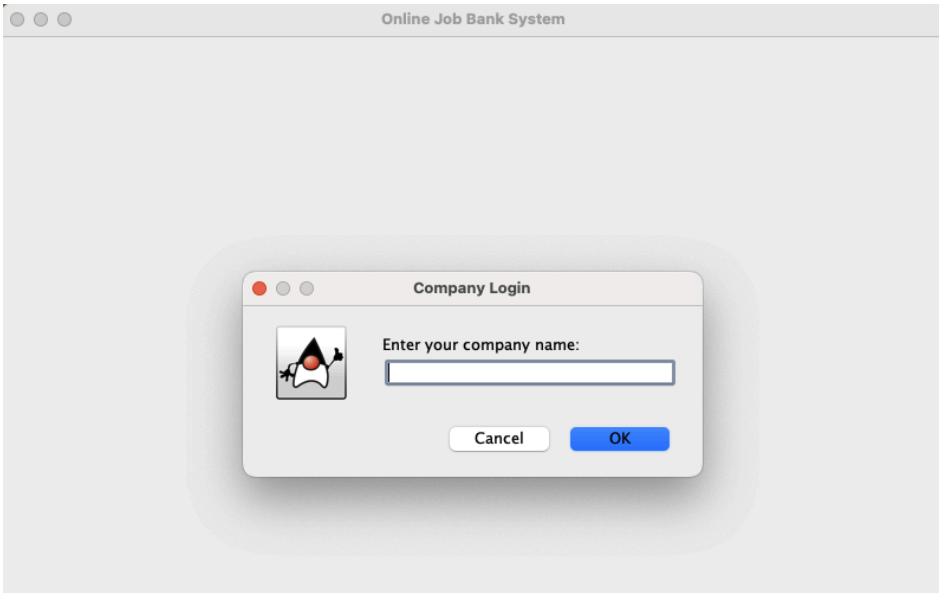
10. Click Exit to exit the application.

If User Selects “Company” Role

1. When the application launches, the login window appears. This is where you enter your Oracle credentials (username and password) and select the role as “**Company**”.



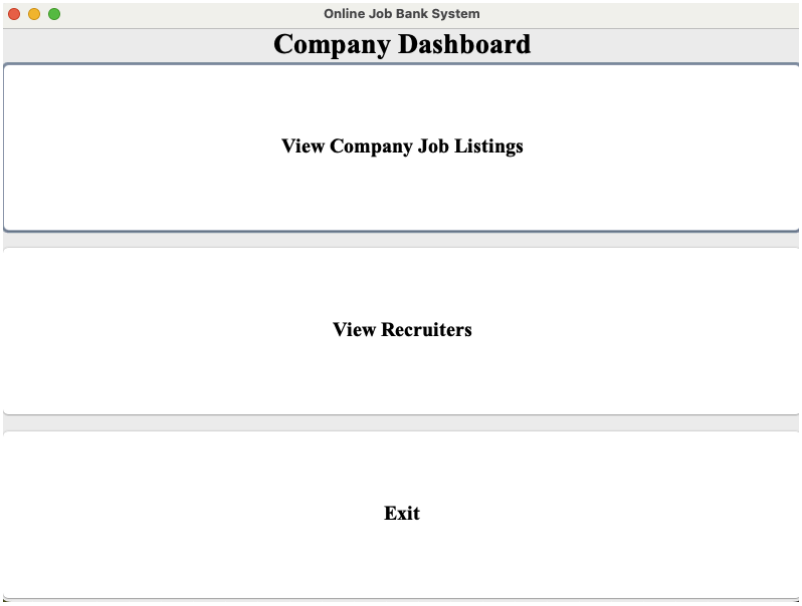
2. Once you have entered your Oracle username and password in the correct format, click the Login button.
    a. If the username and password are in the correct format, then a pop-up will appear telling you to enter your company name (e.g. AMD).



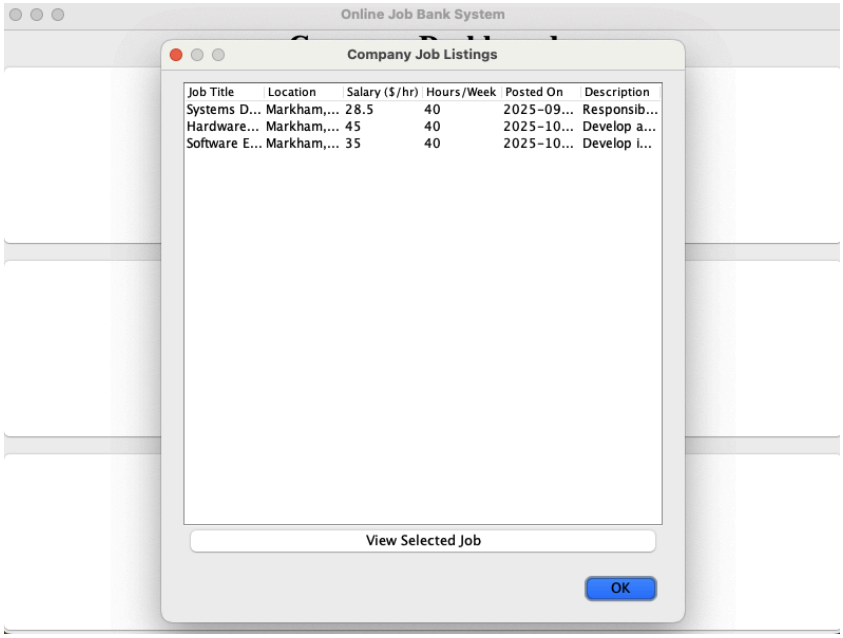
    i. If you enter a company name that is in the database, then the login is successful and you will be directed to the home page.
    ii. If you enter a company name that is not in the database, then you will get an error message saying that the company is not in the database and you will have to login again.

b. If the username and password are not in the correct format, then a pop-up will appear telling you that the login is unsuccessful with an error message. When you click “Ok”, you will be prompted to re-enter your username and password.

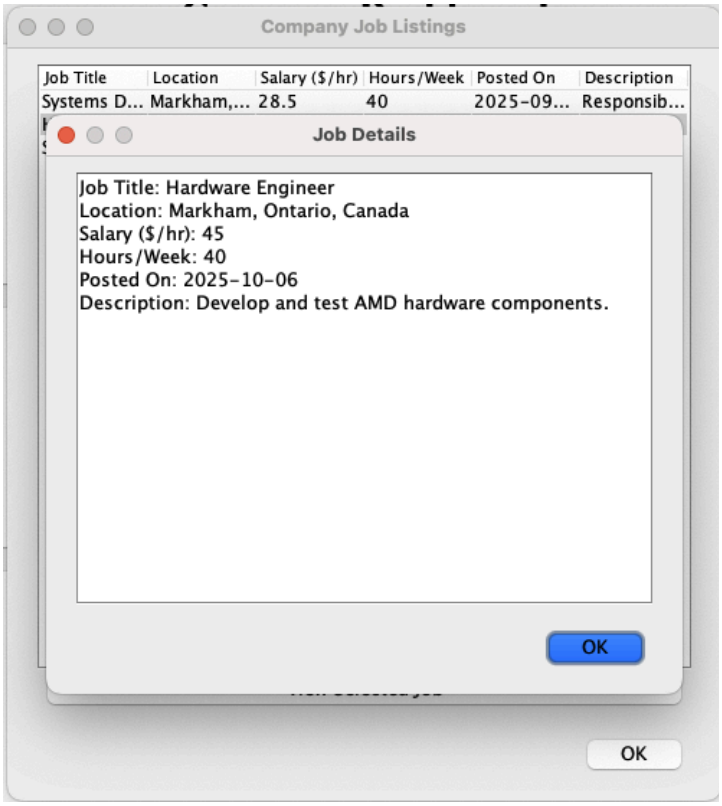
3. After logging in, you will be directed to the home screen, which consists of features such as View Company Job Listings, View Recruiters, and Exit.



4. To view the job listings at the company, click “View Company Job Listings”.
   a. A list of the job postings at the company will be displayed, showing you each job, which consists of a job title, location (city) of the job, salary ($/hr), hours per week, when the job was posted, and the description of the job.

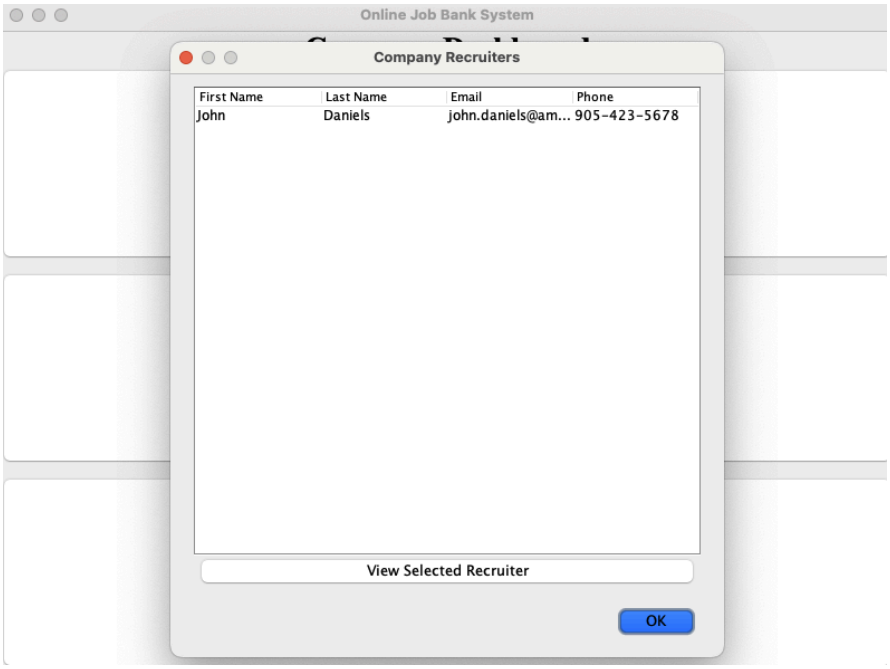


   b. To view the job, click on a job listing and then click on the “View Selected Job” button, which will display the full job details.

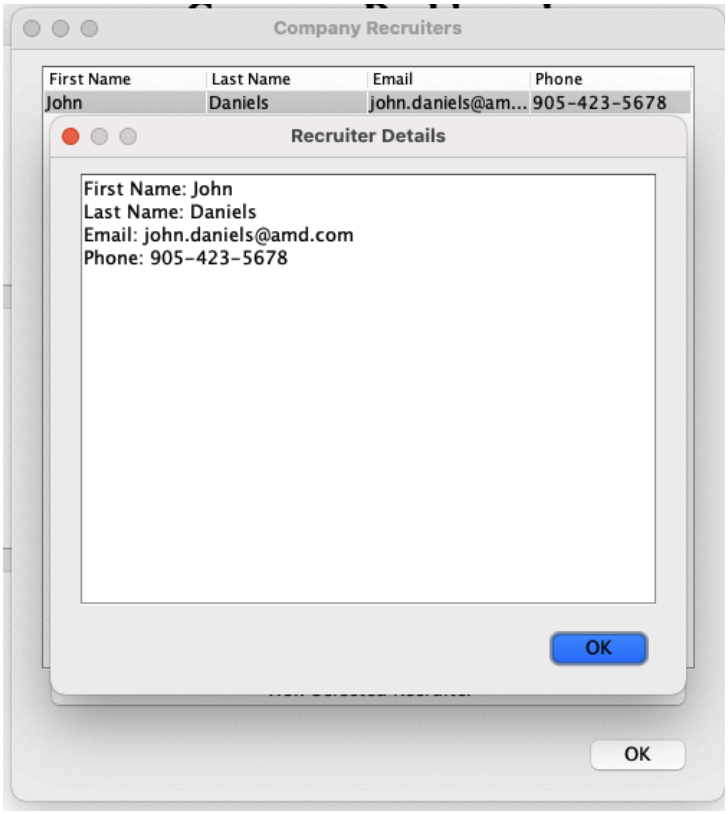


c. Click “Ok” to go back to the home screen.

5. To view the recruiters working for the company, click “View Recrutiers”.
    a. A list of the recruiters working for the company will be displayed, showing you each recruiter, which consists of their first name, last name, email, and phone number.



b. To view the recruiter, click on a recruiter and then click on the “View Selected Recruiter” button, which will display the full recruiter information.



c. Click “Ok” to go back to the home screen.

6. Click Exit to exit the application.

Relational Algebra

Below, every query both in the GUI application and in the previous sections have been converted into the Relational Algebra notation

```
--List all companies who have posted jobs, but have no interviews scheduled yet
SELECT c.name as CompanyName
FROM Company c, Job j
WHERE c.companyID = j.companyID
MINUS
SELECT c.name as CompanyName
FROM Company c, Job j, JobApplication japp, Interview i
WHERE c.companyID = j.companyID AND j.jobID = japp.jobID AND japp.jobAppID = i.jobAppID;
```

Posted ← Π_{name} (Company ⋈ Job)
withInterviews ← Π_{name} (Company ⋈ Job ⋈ JobApplication ⋈ Interview)
Result ← posted - withInterviews

```
--List all companies that have above average working hours for jobs and list their average working hours and the company number of jobs the company has posted
SELECT c.name as CompanyName, AVG(j.workingHours) AS AverageWorkingHours, COUNT(j.jobID) as TotalJobs
FROM Company c, Job j
WHERE c.companyID = j.companyID
GROUP BY c.name
HAVING AVG(j.workingHours) > (
    SELECT AVG(workingHours)
    FROM Job);
```

allJobsAvg ← $F_{AVERAGE\ workingHours}$ (Job)
companyStats ← $_{name}F_{AVERAGE\ workingHours,\ COUNT(jobID)}$ (Company ⋈ Job)
Result ← $F_{AVERAGE\ workingHours}$ > allJobsAvg (companyStats)

```
--List all recruiters and their email that work with apple and applicants with their email that applied for positions at apple
SELECT r.last_name, r.first_name, r.email
FROM RECRUITER r, Company c
WHERE r.companyID = c.companyID AND c.name = 'Apple Canada'
UNION
SELECT ja.last_name, ja.first_name, ja.email
FROM JobApplicant ja, JobApplication japp, Job j, Company c
WHERE c.name ='Apple Canada' and ja.applicantID = japp.applicantID and japp.jobID = j.jobID AND j.companyID = c.companyID;
```

appleRecruiters ← $\Pi_{last\_name,\ first\_name,\ email}$ ($\sigma_{name\ =\ 'Apple\ Canada'}$ (Recruiter ⋈ Company)
appleApplicants ← $\Pi_{last\_name,\ first\_name,\ email}$ ($\sigma_{name\ =\ 'Apple\ Canada'}$ (JobApplicant ⋈ JobApplication ⋈ Job ⋈ Company)
Result ← appleRecruiters ∪ appleApplicants

```
--List all companies that have received atleast one job application still under review
SELECT c.name AS CompanyName
FROM Company c
WHERE EXISTS (
    SELECT *
    FROM Job j, JobApplication japp
    WHERE j.companyID = c.companyID AND japp.jobID = j.jobID AND japp.status = 'Under Review');
```

underReview ← $\Pi_{companyID}$ ($\sigma_{status\ =\ 'Under\ Review'}$ (Job ⋈ JobApplication)
changeName ← $\rho_{(companyID,\ companyName,\ industry,\ location,\ email,\ phone)}$ (Company ⋈ underReview)
Result ← $\Pi_{companyName}$ (changeName)

```sql
--List all applicants and their email who have never been interviewed for a job
SELECT ja.last_name, ja.first_name, ja.email
FROM JobApplicant ja
WHERE NOT EXISTS (
    SELECT *
    FROM JobApplication japp, Interview i
    WHERE japp.applicantID = ja.applicantID AND i.jobAppID = japp.jobAppID);
```

interviewedApplicants ← $\Pi_{applicantID}$ (JobApplication ⋈ Interview)
neverInterviewed ← $\Pi_{applicantID}$ (JobApplication) - interviewedApplicants
Result ← $\Pi_{last\_name,\ first)name,\ email}$ (JobApplicant ⋈ neverInterviewed)

```sql
-- List of job applicants who applied to either Apple Canada or AMD - Combines two sets: Job applicants who applied to Apple Canada or AMD, uses UNION
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Job j ON ja.jobID = j.jobID
JOIN Company c ON j.companyID = c.companyID
WHERE c.name = 'Apple Canada'
UNION
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Job j ON ja.jobID = j.jobID
JOIN Company c ON j.companyID = c.companyID
WHERE c.name = 'AMD';
```

Join ← JobApplicant ⋈ JobApplication ⋈ Job ⋈ Company
toUnion1 ← $\Pi_{applicantID,\ first\_name,\ last\_name}$ ($\sigma_{name\ =\ 'Apple'}$ (Join))
toUnion2 ← $\Pi_{applicantID,\ first\_name,\ last\_name}$ ($\sigma_{name\ =\ 'AMD'}$ (Join))
Result ← toUnion1 ∪ toUnion2

```sql
-- Count total job applications per company and show only those above average - uses GROUP BY, aggregate COUNT, and HAVING with a subquery
SELECT c.name AS CompanyName, COUNT(ja.jobAppID) AS TotalJobApplications
FROM Company c
JOIN Job j ON c.companyID = j.companyID
JOIN JobApplication ja ON j.jobID = ja.jobID
GROUP BY c.name
HAVING COUNT(ja.jobAppID) > (
    SELECT AVG(job_app_count)
    FROM (
        SELECT COUNT(ja2.jobAppID) AS job_app_count
        From JobApplication ja2
        JOIN Job j2 ON ja2.jobID = j2.jobID
        GROUP BY j2.companyID
    )
)
ORDER BY TotalJobApplications DESC;
```

Join ← Company ⋈ Job ⋈ JobApplication
Aggregate ← $\rho_{TotalJobApplications\ /\ COUNT\ jobAppID}$ ($_{name}F_{COUNT\ jobAppID}$ (Join))

JobApps ← JobApplication ⋈ Job
JobAppCount ← $\rho_{job\_app\_count\ /\ COUNT\ jobAppID}$ ($_{companyID}F_{COUNT\ jobAppID}$ (JobApps))
AverageJobAppCount ← $F_{AVERAGE\ job\_app\_count}$ (JobAppCount)

ConditionedAggregate ← $\sigma_{TotalJobApplications\ >\ AverageJobAppCount}$ (Aggregate)

Result ← $\Pi_{name,\ TotalJobApplications}$ (ConditionedAggregate)

```sql
-- Lists all companies that have at least one job posting where at least one job applicant has applied (sent a job application) - uses EXISTS
SELECT DISTINCT c.name AS CompanyName
FROM Company c
WHERE EXISTS (
    SELECT 1
    FROM Job j
    JOIN JobApplication ja ON j.jobID = ja.jobID
    WHERE j.companyID = c.companyID
);
```

Result ← Π_{name} (Company ⋈ Job ⋈ JobApplication)

```sql
-- Lists all the job applicants that have applied (sent a job application) for at least one job, but have never been interviewed - uses MINUS
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
MINUS
SELECT DISTINCT a.applicantID, a.first_name, a.last_name
FROM JobApplicant a
JOIN JobApplication ja ON a.applicantID = ja.applicantID
JOIN Interview i ON ja.jobAppID = i.jobAppID;
```

Total ← $\Pi_{applicantID,\ first\_name,\ last\_name}$ (JobApplicant ⋈ JobApplication)
ToSubtract ← $\Pi_{applicantID,\ first\_name,\ last\_name}$ (JobApplicant ⋈ JobApplication ⋈ Interview)
Result ← Total - ToSubtract

```sql
-- Lists the average salary of all jobs posted, including the minimum and maximum salaries of the jobs, by the company - uses aggregates AVG, MIN, and MAX
SELECT
    c.name AS CompanyName,
    AVG(j.salary) AS AvgSalary,
    MIN(j.salary) AS MinSalary,
    MAX(j.salary) AS MaxSalary
FROM Company c
JOIN Job j ON c.companyID = j.companyID
GROUP BY c.name
ORDER BY AvgSalary DESC;
```

Joined ← Company ⋈ Job
Aggregate ← $_{name}F_{AVERAGE\ salary,\ MIN\ salary,\ MAX\ salary}$ (Joined)

```sql
-- Simple queries
-- Query 1: Returns a distinct list of all the industries from the companies in the Company Table.
SELECT DISTINCT industry
FROM Company
WHERE industry IS NOT NULL
ORDER BY industry;
```

temp← $\sigma_{industry \neq NULL}$ (Company
Result ← $\Pi_{industry}$ (temp)

```sql
-- Query 2: Lists all distinct emails of the recruiters from the Recruiter table.
SELECT DISTINCT email
FROM Recruiter
ORDER BY email;
```

Result ← $\Pi_{email,}$ (Recruiter)

```sql
-- Query 3: Counts the number of jobs posted by each company in the Jobs table.
SELECT companyID, COUNT(jobID) AS job_count, employer
FROM Job
GROUP BY companyID, employer
ORDER BY job_count DESC, companyID ASC;
```

Result ← companyID, employer, $F_{\text{COUNT jobID}}$ (Job)

```sql
-- Query 4: Finds the youngest applicant in the table of Job Applicants.
SELECT DISTINCT first_name, last_name, birthdate
FROM JobApplicant
WHERE last_name IS NOT NULL AND birthdate IS NOT NULL
ORDER BY birthdate DESC;
```

filtered ← $(\sigma_{\text{industry} \neq \text{NULL},\ \text{birthdate} \neq \text{NULL}}$ (JobApplicant)
Result ← $\Pi_{\text{first\_name, last\_name,birthdate,}}$ (filtered)

```sql
-- Query 5: Counts the number of Job Applications from the Job Applications table that have been submitted to a Job posting.
SELECT jobID, COUNT(jobAppID) AS application_count
FROM JobApplication
GROUP BY jobID
ORDER BY application_count DESC, jobID ASC;
```

grouped ← $_{\text{job ID}}F_{\text{COUNT jobAppID}}$ (JobApplication)
Renamed ← $\rho_{(\text{application\_count, jobID, applicantID, dateTime, status})}$ (grouped)
result ← $\Pi_{\text{aoplication\_count}}$ (renamed)

```sql
-- Query 6: Lists all the applicants who have uploaded a resume, ordered by upload date of the resume
SELECT DISTINCT applicantID, uploadDate
FROM Resume
ORDER BY uploadDate DESC, applicantID ASC;
```

Result ← $\Pi_{\text{aoplicationID, uploadDate}}$ (Resume)

```sql
-- Query 7: Counts the number of scheduled interviews per job application.
SELECT jobAppID, COUNT(interviewID) AS interview_count
FROM Interview
GROUP BY jobAppID
ORDER BY interview_count;
```

grouped ← $\rho_{(\text{interview\_count},\ \text{COUNT(interviewID)})}\ (_{\text{jobAppID}}F_{\text{COUNT interviewID}}$ (Interview)
result ← $\Pi_{\text{interview\_count}}$ (grouped)

```sql
-- list all jobs with 'Software' in the title
SELECT *
FROM Job
WHERE Job.title LIKE 'Software%';
```

Result ← $\sigma_{\text{title ='Software',}}$ (Job)

```sql
-- list all of Jake Blake's resumes
SELECT JobApplicant.first_name, JobApplicant.last_name, Resume.resumeID, Resume.upload_file, Resume.uploadDate
FROM Resume JOIN JobApplicant ON Resume.applicantID = JobApplicant.applicantID
WHERE JobApplicant.first_name = 'Jake' AND JobApplicant.last_name = 'Blake';
```

Filtered = $(\sigma_{\text{first\_name = 'Jake', last\_name ='Blake'}}$ (Resume ⋈ JobApplicant)
Result ← $\Pi_{\text{first\_name, last\_name, resumeID, upload\_file, upload\_Date}}$ (filtered)

```sql
SELECT Distinct location
FROM Job
ORDER BY location;
```

Result ← $\Pi_{location}$ (Job)

```sql
SELECT DISTINCT uploadDate as upldDate
FROM Resume
ORDER BY upldDate;
```

nameChange ← $\rho_{resumeID,\ applicantID,\ upload\_file,\ upldDate)}$ (Resume)
Result ← $\Pi_{upldDate}$ (nameChange)

```sql
-- Job applicants who applied for jobs posted by the recrutier 'Bob William'
SELECT a.applicantID, a.first_name || ' ' || a.last_name AS applicant_name
FROM JobApplicant a, JobApplication ja, Job j, Recruiter r
WHERE r.first_name = 'Bob'
    AND r.last_name = 'William'
    AND r.recruiterID = j.recruiterID
    AND j.jobID = ja.jobID
    AND ja.applicantID = a.applicantID
ORDER BY a.applicantID asc;
```

Result ← $\Pi_{applicantID,\ first\_name,\ last\_name}$ $\sigma_{first\_name = 'Bob' \wedge last\_name = 'William'}$ (Recruiter ⋈ Job ⋈ JobApplication ⋈ JobApplicant)

```sql
-- Job applicants who applied for the 'Software Engineer' job at the company 'Apple Canada'
SELECT a.applicantID, a.first_name || ' ' || a.last_name AS applicant_name
FROM JobApplicant a, JobApplication ja, Job j, Company c
WHERE j.title = 'Software Engineer'
    AND c.name = 'Apple Canada'
    AND j.companyID = c.companyID
    AND ja.jobID = j.jobID
    AND ja.applicantID = a.applicantID
ORDER BY a.applicantID asc;
```

Result ← $\Pi_{applicantID,\ first\_name,\ last\_name}$ $\sigma_{title = 'Software\ Engineer' \wedge name = 'Apple\ Canada'}$ (JobApplicant ⋈ JobApplication ⋈ Job ⋈ Company)

```sql
-- Number of job applicants who have interviews for jobs posted by each recruiter
SELECT r.first_name || ' ' || r.last_name AS recruiter_name, COUNT(DISTINCT a.applicantID) AS applicant_count
FROM Interview i, JobApplication ja, Job j, Recruiter r, JobApplicant a
WHERE r.recruiterID = j.recruiterID
    AND j.jobID = ja.jobID
    AND ja.jobAppID = i.jobAppID
    AND ja.applicantID = a.applicantID
GROUP BY r.first_name, r.last_name
ORDER BY applicant_count DESC;
```

Join ← Recruiter ⋈ Job ⋈ JobApplication ⋈ JobApplicant ⋈ Interview
Result ← $_{first\_name,\ last\_name}F_{COUNT\ applicantID}$ (Join)

```sql
-- list number of job postings per company (showing company name as well)
SELECT Company.companyID, Company.name, COUNT(jobID) as "Jobs Posted By Company"
FROM Job, Company
WHERE Job.companyID = Company.companyID
GROUP BY Company.companyID, Company.name
ORDER BY Company.companyID DESC;
-- alternative notation (we're just gonna use this because i dont like the ambiguity implicit joins)
SELECT Company.companyID, Company.name, COUNT(jobID) as "Jobs Posted By Company"
FROM Job JOIN Company ON Job.companyID = Company.companyID
GROUP BY Company.companyID, Company.name
ORDER BY Company.companyID DESC;
```

Join ← Company ⋈ Job

Result ← $_{companyiD,\ name}F_{COUNT\ jobID}$ (Join)

```sql
-- list the job applicants, who have job applications to Apple Canada, and the status of those applications
SELECT JobApplicant.first_name, JobApplicant.last_name, JobApplication.status,  Job.title, Company.name, Company.companyID
FROM JobApplication JOIN Job ON JobApplication.jobID = Job.jobID
    JOIN Company on Job.companyID = Company.companyID
    JOIN JobApplicant ON JobApplication.applicantID = JobApplicant.applicantID
WHERE Company.name = 'Apple Canada';
```

Join ← JobApplicant ⋈ JobApplication ⋈ Job ⋈ Company

Result ← $\Pi_{first\_name,\ last\_name,\ status,\ title,\ name}\ \sigma_{name\ =\ 'Apple\ Canada'}$ (Join)

```sql
-- list the (super awesome and effective) resumes from those who have a pending interview to a job
SELECT Resume.resumeID, Resume.upload_file, JobApplicant.first_name, JobApplicant.last_name, Job.title, Job.employer, JobApplication.status, Interview.dateTime, Interview.location
FROM Resume JOIN JobApplicant on Resume.applicantID = JobApplicant.applicantID
    JOIN JobApplication on JobApplicant.applicantID = JobApplication.applicantID
    JOIN Interview on JobApplication.jobAppID = Interview.jobAppID
    JOIN Job on JobApplication.jobID = Job.jobID
WHERE JobApplication.status = 'Interview Pending';
```

Join ← Resume ⋈ JobApplicant ⋈ JobApplication ⋈ Job

Result ← $\sigma_{status\ =\ 'Interview\ Pending'}$ (Join)

```sql
SELECT JobApplicant.applicantID, last_name, first_name
FROM JobApplicant, JobApplication, Interview, Company, Job
WHERE Company.name = 'Apple Canada' AND JobApplicant.applicantID = JobApplication.applicantID AND JobApplication.jobAppID = Interview.jobAppID
AND JobApplication.jobID = Job.jobID AND Job.companyID = Company.companyID
ORDER BY applicantID ASC;
```

Join ← JobApplicant ⋈ JobApplication ⋈ Interview ⋈ Job ⋈ Company

Result ← $\Pi_{applicantID,\ last\_name,\ first\_name}\ \sigma_{name\ =\ 'Apple\ Canada'}$ (Join)

```sql
SELECT JobApplicant.applicantID, last_name, first_name
FROM JobApplicant, JobApplication, Job, Company
WHERE Company.name = 'AMD' AND Job.companyID = Company.companyID AND JobApplication.applicantID = JobApplicant.applicantID AND Job.jobID = JobApplication.jobID
ORDER BY jobAppID ASC;
```

Join ← JobApplicant ⋈ JobApplication ⋈ Job ⋈ Company

Result ← $\Pi_{applicantID,\ last\_name,\ first\_name}\ \sigma_{name\ =\ 'AMD'}$ (Join)

```sql
SELECT Company.name as companyName, last_name, first_name, Job.jobID
FROM Recruiter, Job, Company
WHERE recruiter.companyID = Company.companyID AND Job.recruiterID = Recruiter.recruiterID
ORDER BY Company.name ASC;
```

Join ← Recruiter ⋈ Job ⋈ Company

Result ← $\Pi_{name,\ last\_name,\ first\_name,\ jobID}$ (Join)

Conclusion

In this report, a database management system for an online job bank was proposed and then elaborated upon. The database and the process of its creation and refinement was extensively detailed from its very preliminary stage (nature of business, motivation), to its conceptual stage (schema, ER diagrams), to initial drafts/implementations (UNIX shell script, unnormalised format), and then to its final implementation (GUI, normalisation). Relational algebra notation was also used to describe the implemented queries. Afterwards, plans for future development and possible expansions to the project were proposed.

We hope that this document serves as an informative one, and potentially as a great foundation for any others to make their own database job bank systems.