**IS 420: Database Application Development**

**Spring 2024**

Group Project

Online Job Portal

## Overview

You will be assigned into groups of four to five people in this project. Please read the whole document carefully before starting your project. Your assignment is to design an Online Job Portal. You will design the database, insert some sample data, and implement a set of required features in the backend. Each feature will be implemented as one Oracle PL/SQL procedure. You do **NOT** need to write a graphic user interface. You can test your features using SQL or PL/SQL scripts to call the implemented procedures.

**Assumptions:**

You can make the following assumptions in this project.

1. The system will store information about accounts, including account ID, account holder's name, address, zipcode, state, email, phone number, account type, and registration date. There are two account types: job seeker or recruiter.

2. The system stores information about companies. Each company has a company ID and company name.

3. The system stores profiles of job seekers. A profile includes the associated account ID, highest degree (associate, bachelor, master's, doctoral), major, school where the degree is awarded, number of years of work experience.

Hint: you can use number of integer to represent degree level, e.g., associate degree = 1, bachelor = 2, master's = 3, doctoral = 4.

4. The system stores a list of skills. Each skill has a skill ID, skill name (e.g., Java, Python, SQL, PL/SQL).

5. The system stores skills associated with each job seeker, including skill ID, account ID, skill level (basic=1, proficient=2, expert=3).

6. The system stores information about job posts. Each post has a job ID, associated company ID, account ID of the recruiter who posted the position, job title, a description of the position, job type (full time or part-time), expected annual pay range (a minimal and a maximal), job city and state, a minimal degree level, a minimal number of years of work experience, and a status (1 means active, 0 means not active).

7. Each job post has a set of required skills and the minimal required level (basic, proficient, or expert).

8. Each job seeker can apply for job posts. The system stores information about each application, including an application id, the account ID of the job seeker, the job post ID, application date, and application status (1=submitted, 2=first round, 3=second round, 4=final round, 5=accepted, 0=denied).

9. The system stores a message table, which has a message ID, the associated account ID, message time, and message body.

**Features:** There are five individual features and five group features. Each member needs to implement one individual feature. So if your group has X members your group will implement any X individual features. Each group also needs to implement X group features. For example, if your group has five members, your group will do all 10 features. If your group has four members, your group will implement four individual features plus four group features.

Individual features will be graded individually (your group member's individual feature will have no impact on your grade), but group features will be graded group-wise. So each group should work together to make sure the group features are done correctly.

**Individual features: (one feature per member)**

**Feature 1:** add a new account. Input includes user name, phone number, address, state, zip, email, account type (job seeker or recruiter) and registration date. This feature does the following:

1. It checks whether any user with the same phone number as the input exists. If so, it prints a message 'the account already exists' and updates address, state, zip, email.
2. If there is no user with the same phone number, it generates a new account ID (using a sequence) and inserts a row into the account table with a new account ID, given phone, name, address, state, zip, email and account type and registration date. Please also print out the new account ID. You can use a sequence to generate the new account ID.

**Feature 2:** add a company. Input includes a company name. The procedure does the following:

1) It first checks whether there is a company with the same name as the input, if so prints a message 'company already exists' and stops.

2) If the company does not exist, it inserts a row into the company table with a newly generated company ID (using sequence) and the input company name. It also prints out the new company ID.

**Feature 3:** search for job posts. The input includes a keyword, job type (full time or part time), and a state. The procedure does the following:

1) It finds all job posts that are active and the job title or job description has a substring that matches the input keyword, and the job post's state and job type match the input state and job type. The procedure prints out the matching job posts' job post id, associated company name, job title, description, min and max pay, city and state.

2) In case there are no matching job posts, print out 'No job posts found'.

Hint: to match a substring in the procedure, use

where column\_name like '%' || input\_keyword || '%'

**Feature 4:** Apply for a job post. Input: account ID, job post ID, a date. The procedure does the following:

1) It first checks whether there is an existing account with the input account ID and the account type is job seeker. If these conditions are not satisfied, print a message 'Invalid account ID' and stop.

2) It then checks whether there is a job post with the input job post ID and the status is active. If these conditions are not satisfied, print a message 'Invalid job post' and stop.

3) It then inserts a row into the application table, with a newly generated application ID (using a sequence), input account ID, input job post ID, the input date as application date, and status as submitted. Print a message 'New application X created' where X is the new application ID.

**Feature 5:** list all applications to a particular job post. The input is a job post ID.

This procedure does the following:

1) it checks whether there is a job post with the input post ID.

If there is no such job post print a message 'Invalid job post ID') and stop.

2) it then prints out all job seekers who have applied for this job, including their account ID, name, application date, highest degree, number of years of work experience, and the name and skill levels of each skill the applicant has.

**Group features:**

**Feature 6:** adds a job post. Input includes an account ID of a recruiter, company ID, job title, job description, job type, job city and state, minimal pay, maximal pay, city, state, minimal degree level, minimal number of years of experience, and a list of required skill IDs and minimal required level.

The procedure does the following:

1) it first checks whether there is an account with matching input account ID and the account type is recruiter. If there is no such account, it prints a message 'Invalid account ID' and stops.

2) it then checks whether there is a company with the input company ID. If there is no such company, print a message 'Invalid company ID' and stop.

3) if both the account ID and company IDs are valid, the procedure inserts a row into job\_post table with a newly generated job post ID, the input company ID, the input account ID, the input job title, the input job description, the input job type, minimal pay, maximal pay, city, state, minimal degree, minimal number of years of experience, and job status as active.

Print a message 'New job post ID is X' where X is the new job post.

4) for each pair of input skill ID and minimal skill level, insert a row into the table that stores required skills for the job post.

Hint: create a varray type of integers and use this type for the list of skill IDs and the list of minimal skill level.

**Feature 7:** Generate job alerts. The input is a job post ID and a timestamp. The procedure does the following:

1) it first checks whether there is a job post with the input ID and is still active. If there is no such a job post, print a message 'Invalid job post ID' and stop.

2) it then finds all job seekers who satisfy ALL of the following conditions:

a) the job seeker's highest degree meets the minimal degree requirement of the job post (if you use 1 for associate degree, 2 for bachelor, 3 for master's and 4 for doctoral degree, then the highest degree >= minimal degree of the job post);

b) the job seeker's number of years of work experience is greater or equal to the minimal experience of the job post;

c) for each skill required by the job post, the job seeker has that skill and the skill level is greater or equal to required level.

3) for each job seeker found in step 2), print out the name of the job seeker and insert a row into the message table with a newly generated message ID, account ID as the job seeker's account ID, message time as the input time, and body of the message is 'A job post X is available and you are qualified to apply' where X is the input job post ID.

Hint: to check condition c) in step 2, you can use an explicit cursor to return job seeker who satisfies condition a) and b) and another explicit cursor to return the required skill ID and minimal level for the job post. A nested loop can then be used to check condition c using the following pseudo code:

-- c1 is the first cursor to check condition a and b, c2 is the second cursor to get required skill IDs and minimal skill levels.

for r1 in c1 loop

flag:=1;

for r2 in c2 loop

-- use select count(\*) to check whether condition c is satisfied (i.e., whether job seeker r1 has a skill that matches the skill returned in r2 and the skill level >= r2's minimal level.

if the count is zero (not satisfied), set flag to 0 and exit the loop

end for;

if flag = 1 then

-- do step 3.

end if;

end for;

**Feature 8:** Assign an initial score for applicants to a job post. The input is a job post ID.

This procedure does the following:

1) it first checks whether there is a job post with the input ID and is active. If there is no such job post, print out 'Invalid job post ID' and stop.

2) it then finds all job seekers who applied to this job post and whose highest degree level is greater or equal to the minimal degree of the job post and the number of years of work experience is greater or equal to the minimal work experience of the job post.

3) it then computes a score for each job seeker returned in step 2. Score = the job seeker's number of years of experience + (degree level of job seeker - minimal degree level for the job post)\*2 + number of skills required by the job\_post that are satisfied by the job seeker.

A skill required by the job post is satisfied if the job seeker has that skill and the skill level is greater than or equal to the minimal level required by the job post.

1. print out job seeker's name of each application returned in step 2 along with the score computed at step 3.

**Feature 9:** Update status of an application. Input is an application ID, a status, and a timestamp. This procedure does the following:

1) it first checks whether there is an application whose application ID matches the input application ID. If there is no such an application, print 'Invalid application ID' and stop.

2) it then updates the status of the application whose application ID matches the input ID. The new status is the input status.

3) insert a row to the message table with a newly generated message ID, account ID as the application's account ID, message time as the input time, and the message body based on the input status as:

in case the new status is 1: message body is 'Application to job X submitted' where x is job post ID;

in case the new status is 2: message body is 'Application to job X reaches first round';

in case the new status is 3: message body is 'Application to job X reaches second round';

in case the new status is 4: message body is 'Application to job X reaches final round';

in case the new status is 5: message body is 'Congratulations! Your application to job X is accepted'

in case the new status is 0: message body is 'Sorry your application to job X is declined';

in case the new status is none of the above value, print a message 'Invalid status'

**Feature 10:** print statistics. Input is a start date and an end date. The procedure does the following:

1) it prints out the total number of active job posts.

2) it prints out the total number of job seekers and the total number of recruiters.

3) it prints out the total number of applications submitted between the input start and end date.

4) it prints out the average number of applications per active job post between the start and end date. Hint: you can compute total number of applications between start and end date and then divide it by #of active job posts.

5) it prints out the average number of applications submitted per job seeker between the start and end date. Hint: compute total number of applications between start and end date and divide it by total number of job seekers.

6) it prints out the average number of active job posts posted by each recruiter. Hint: compute the total number of active job posts and then divide it by the number of recruiters.

7) it prints out the number of job seekers at each degree level.

8) it prints out the number of active job posts at each minimal degree level.

**Deliverables:**

1. 10%. Due 2/20. Project Management Schedule.

a. Include team members and a timeline showing each phase of your project with its activities and time duration, for the entire effort.

b. It is expected that every member should participate in all phases of the project.

c. Please specify which feature is assigned to which member (for group features it is still possible to assign a lead for each feature).

d. Activities should include system design, populating tables, writing code, testing code, running example queries, writing documents, preparing for presentation, etc. Smaller milestones shall be set for deliverable 3 and 4.

e. This deliverable will be graded based on whether items a) to d) are included and whether the schedule is reasonable (e.g., enough time such as 2-3 weeks are left for testing and integration).

2. 25%. Due 3/12. Design Document which includes the following:

a. ER diagram of the database. You don’t have to follow exact notations of the ER diagram, but need to show tables, columns, primary keys, and foreign key links.

b. SQL statements to create database tables and to insert some sample data (at least 3 rows per table). Please include drop table and drop sequence statements before create table, create sequence and insert.

c. Specification for each required feature. The specification should include a description of input parameters and output (usually printing a message), and a few test cases (normally there should be one normal case and a few special cases). You don’t need to implement any of these procedures at this point.

3. 35%. Due 5/8. Presentation of database design and demonstration of all individual features plus two group features. You can finish the remaining group features by D4 deadline (5/21). The project demo will be online (through Webex). You can sign up for the time of presentation/demo through a google form. To demo each feature, you need to prepare a couple of test cases, usually one normal case and the rest as special cases. For each test case you need to be able to explain why your answer is correct (this can be typically done by showing some tables or screen output).

4. 30%. Due 5/21. Please upload the final code through blackboard. The code should include:

a. Drop table and sequence statements to drop tables if they exist (remember to use cascade constraints).

b. Create table statements and create sequence statements

c. Insert statements

d. Create procedure statements (with code for the procedures) for all features. Each feature can be implemented as one PL/SQL procedure (in the procedure you may call other procedures or functions). Please include some comments in your code explaining the major steps. You should use create or replace to avoid procedure name conflict.

e. Test script to show that all your features work correctly. The script shall include some examples to test different cases. E.g., for feature 1, one example for new customer added (phone is not in database) and one example for existing customer. Please include:

i. PL/SQL script to call the appropriate PL/SQL procedure for this feature. E.g., exec procedure-name(parameter values)

ii. Explanation of what should be the correct output. The output could be updated tables (you can have some select statement to show the updated tables), some print out, etc.

iii. Make sure you have tested your examples from beginning to end. Remember that database tables may have been changed in the process. So you may need to start with a clean database (i.e., right after you execute all the drop table, create table, and insert statements).

**Grading Guidelines**

What I look for while grading software code (deliverable 4):

1. Existence of code and whether all code can be compiled without any error.
2. Comments: Both descriptive and inline for every procedure/function
3. Software quality
   1. Whether it is correct (giving correct results).
   2. Whether it is complete and clear.
   3. Efficiency of code. You shall not use too many SQL statements, and you shall put as much work as possible in SQL. For example, if you can do a join, do not use two select statements and then do a join in your program.
   4. Whether it has considered all special cases such as whether a user has already registered in Feature 1.

Regarding the presentation of your project: Each student must participate in the project demonstration by presenting to the entire class some slides. You will be graded on:

1. Timeliness of presentation
2. Presentation Style
3. Demo (running the code)

For the demo, you will be graded on the following items:

1. Existence of tables and data. You need to have at least 3 rows in each table.
2. The correctness of features. This can be shown by checking whether the screen output is correct and the database has been updated correctly.

Each member of the team shall contribute more or less equally. It is unfair for a few members to do most of the work while others do less. You will be asked to evaluate your teammate’s effort at the end of the project. The instructor will adjust the grade based on the evaluation. Normally if most of your teammates agree that you do not contribute at all or contribute too little (e.g., your group has 4 members and you contribute only 5%), you may lose up to 80% of your project grade. If your teammates agree that you contribute much more than anyone else (e.g., your group has 4 members and you contribute 40%), you may gain up to 20% of your project grade (but not exceeding 100% of project grade). A peer evaluation will be conducted at the end of the semester to determine the contribution of each team member.

Tips:

1. Work as a team. Each member can do individual features by yourself but should work on other parts of the project including group features as a team. This means do NOT miss group meetings, do not miss internal deadlines, and help each other for tasks that belong to the whole group. You should also divide up group tasks fairly and according to everyone's strength.
2. Start early. Do not wait until last month to start coding. Do not wait until one week before the demo to start putting things together. Past experiences show that more than 50% of time shall be devoted to testing and putting things together.
3. Learn how to debug SQL and PL/SQL code. Most of the time the error is from the SQL part of your code. So you can test SQL part separately (e.g., by copy & paste the SQL statement in a cursor and replace PL/SQL variables/parameters with values). You can insert screen output statements to check intermediate results. Oracle also returns error messages and error code. You can google the error messages and error code to find possible causes. You may also use Oracle SQL Developer which allows you to insert breakpoints during debugging.
4. It is highly recommended to use SQL Developer rather than the web interface for the project.
5. Use homework, in class exercises, and programs in slides as templates of your PL/SQL program. For example, if you need to write a cursor, find a cursor example and use it as a starting point.
6. Make sure special cases are handled.
7. At demo time, different data in the database may lead to different results. So usually you will start with a standard database (with a fixed set of tables and rows), and keep track of the sequence of the demo (e.g., a course can only be scheduled if it has been added first).