ECE656 W25 – Lab 1 – SQL Queries

Due date: Friday January 24th, 2025 (submit to dropbox in LEARN)

Group work policy: In this lab, students are expected to work in groups no larger than 3. Groups of 1 or 2 are fine.

Weight: Please see course outline (linked posted in LEARN).

Objective: In this lab, you will practice SQL statements on a small schema.

Submission: Certain problems are marked for submission in **boldface**. These will be graded automatically via grading scripts that check the output of your queries. When you are done, please enter your solutions (i.e., your SQL) into the provided template file (`lab1.sql`), and submit this file using the Lab 1 dropbox in LEARN. **Please do not enclose your SQL code in comments, or rename the `lab1.sql` file, as that will break the grading script.** The remaining problems, which are not marked for submission, are practice problems and will not be graded.

1 Database setup

You will use the `mysql` command line utility to connect to a MySQL DBMS on the server `manta.uwaterloo.ca`. You will then create some tables, add some data, and use SQL statements to manipulate this database.

1.1 Connecting and logging in

Using SSH, first connect to one of the ecelinux hosts (e.g., `eceubuntu1.uwaterloo.ca` or `ecetesla1.uwaterloo.ca`) with your Nexus user name and password.[1] Once you are connected to the ecelinux host, you can access the DBMS via the following command:

```
mysql -h manta -u mysql_<nexususerid> -p
```

For example, for user `x12cool`, the command will be:

```
mysql -h manta -u mysql_x12cool -p
```

Each student is initially assigned a temporary MySQL password of the following form:

```
mysql_<nexususerid>_HeLLo<&W25!
```

[1] The firewall blocks access to the MySQL port from outside the university, and so you will need to **use the campus VPN if you are working from home**. Alternatively, you can SSH into `eceterm1.uwaterloo.ca` or `eceterm2.uwaterloo.ca` or `eceterm3.uwaterloo.ca` first, and then SSH into one of the other Linux servers.

For example, for user `x12cool`, the temporary password is:

```
mysql_x12cool_HeLLo<&W25!
```

Please change your temporary password as soon as possible. To do so, use a command of the following form:

```
set password = 'myCleverNewPassword';
```

Remember to replace 'myCleverNewPassword' with your own clever password.

1.2 Selecting a database

The `mysql` utility provides a command line interface to the DBMS running on `manta`. A DBMS may store many databases, and so upon connecting to the DBMS, you must first specify the database that you will be working with. This is accomplished by executing a command of the following form:

```
use db_<nexususerid>;
```

For example, for user `x12cool`, the command will be:

```
use db_x12cool;
```

Alternatively, you can specify the name of the database on the command line when you launch the `mysql` utility, and avoid the `use` command later on. For example, for user `x12cool`, the revised command line will be:

```
mysql -h manta.uwaterloo.ca -u user_x12cool -p db_x12cool
```

1.3 Creating the schema

Once you have chosen the correct database, you can create the required tables and add data to these tables. We provide a script for this purpose, called `createTables.sql`. Examine this file to familiarize yourself with the tables and data that your database will contain. The logical schema is illustrated below in Figure 1 using the relational schema diagram format discussed in lecture.

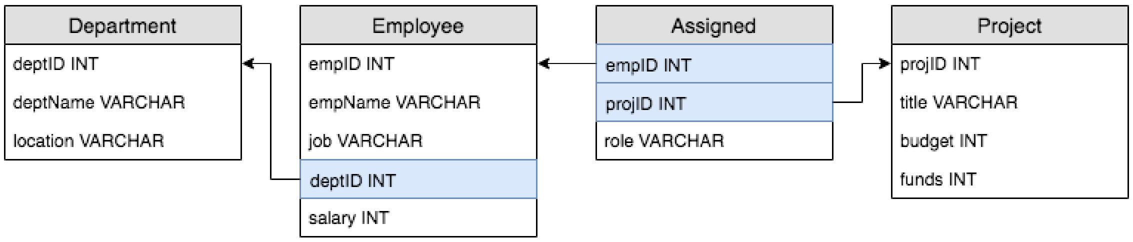


Figure 1: Enterprise schema for Lab 1.

To create the schema, first copy the `createTables.sql` script to the host where you are running the `mysql` utility (e.g., `eceubuntu1` or `ecetesla1`). This can be accomplished using tools like `scp` or `sftp`. After you have copied the script to the current working directory of the `mysql` utility, you can execute the script using the following command:

```
source createTables.sql;
```

Verify that the script ran to completion without producing any warnings or errors, and check that the required tables now exist in your database by running the following command:

```
show tables;
```

Please contact the teaching team as soon as possible if you experience any technical difficulties with the setup. Contact information for the instructor and TA(s) is included in the course outline.

2 Submission guidelines

Your mission for Lab 1 is to write SQL code that solves the problems posed in Section 3. Please follow the submission guidelines in this section carefully before submitting your solution to the Lab 1 dropbox in LEARN, as otherwise your submission may break our grading script and lead to academic penalties.

- First, download the provided `lab1.sql` template file from LEARN.
- Write your answers (i.e., your SQL code) in this file for each problem in the appropriate place, which is indicated by the block comments (`/* ... */`). You must remove these block comments and replace them with SQL.
- Do not remove the separator lines, which start with four dashes (e.g., `---- 1b`). Similarly, do not add any new lines containing four dashes. Standard SQL comments use double dashes, and are allowed.
- Submit your completed `lab1.sql` file to the Lab 1 dropbox in LEARN before the due date (as specified in the dropbox details). Note that only one file is accepted per dropbox submission, and each re-submission prior to the deadline overwrites the previous one.

Note 1: Submissions that do not comply with the above guidelines may be penalized.

Note 2: The grading script will use the same schema as defined in `createTables.sql`, but with a different data set. As a result, testing your SQL code successfully on the provided data set does not guarantee that your solutions will pass all the test cases of the grading script. You are responsible for testing your SQL code thoroughly to ensure correctness.

Note 3: Table names are case-sensitive in MySQL running on Linux, and case-insensitive on Windows. **The grading script will run on a case-sensitive Linux server.**

3 Problems

Note 1: The solution to each problem must be a **single SQL statement terminated by a semicolon**. Do not submit the result set generated by this code.

Note 2: Assume that `deptID`, `empID`, and `projID` are not nullable.

Single table queries

1a) Which departments are located in Cairo?

1b) How many people are employed in each job type (see `Employee.job` column) in the company, ordered ascending by job type? The names of the output columns for the job type and number of people should be `job` and `count`, respectively.
(Marked for submission)

1c) What are the names and salaries of the engineers?

1d) What is the average salary by job type?

1e) What is the ID of the department (or departments if there is tie) with the most engineers? The name of the output column should be `deptID`. Assume that there is at least one department with at least one employee in it.
(Marked for submission)

1f) What is the percentage of engineers in each department (provide the corresponding department ID and percentage)?

1g) Return the `empID` of all employees earning the second-highest salary, as defined in Appendix A for $k = 2$. Assume that there are at least two distinct non-null salaries in the data set.
(Marked for submission)

Multi table queries

2a) What are the names and IDs of employees who are currently not assigned to any project? The names of the output columns should be `empName` and `empID`, respectively.
(Marked for submission)

2b) What are the names, jobs and roles of all employees whose role is different than their job?

2c) What is the number of employees, by job, whose role is identical to their job?

2d) What is the sum of the salaries of all employees assigned to each project (return the project title and total salary for each project)?

2e) Repeat (2d) but include an additional row for the total of the salaries of all employees not assigned to any project. The names of the output columns should be `projID` and `projectSalary`. The `projID` for the additional row representing employees without projects should be the SQL `null` value. If every employee is assigned to at least one project then the additional row is optional, meaning that you may include it but you are not required to do so.
(Marked for submission)

2f) Identify the employees (if any), by name and ID, who are assigned to more than one project.

Data insertion, deletion, and update

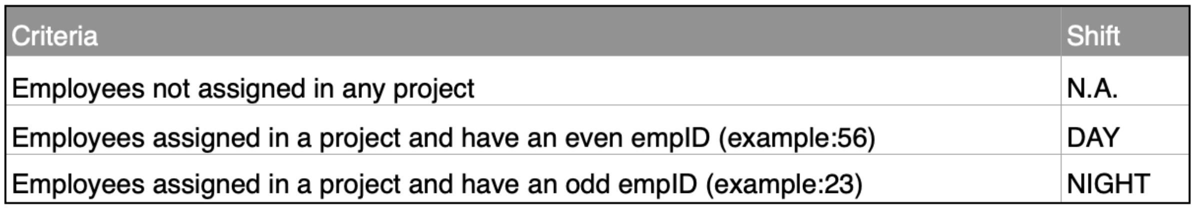
To ensure that your results are reproducible, please ensure that you reload your database (i.e., rerun the provided SQL script) before running the code for each of the following problems, except between 3c and 3d.

3a) Raise the salary of everyone working on the compiler project by 10%.
(Marked for submission)

3b) Raise the salary of all janitors by 5%, and all Waterloo employees by 8%; if there are any janitors in Waterloo, their pay should be raised by 8%, not by 13%.
(Marked for submission)

3c) Add a nullable column `shift` of type `VARCHAR(5)` to the `Employee` table.
(Marked for submission)

3d) Populate the `shift` column created in part (3c) with a value for each employee that satisfies the criteria defined in Figure 2.
(Marked for submission)

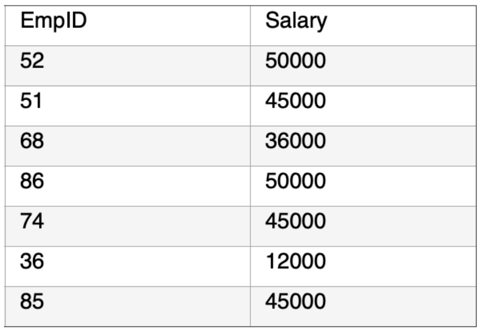


Criteria	Shift
Employees not assigned in any project	N.A.
Employees assigned in a project and have an even empID (example:56)	DAY
Employees assigned in a project and have an odd empID (example:23)	NIGHT

Figure 2: Criteria for values in the new column `shift`.

A Definition of kth-highest salary

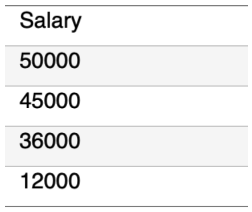
This appendix defines the meaning of kth-highest salary, taking into account cases where multiple employees may share the same salary. Consider the table shown below in Figure 3:



EmpID	Salary
52	50000
51	45000
68	36000
86	50000
74	45000
36	12000
85	45000

Figure 3: Projection of Employee table onto employee ID and salary.

Next, consider the distinct salaries in descending order, as shown in Figure 4:

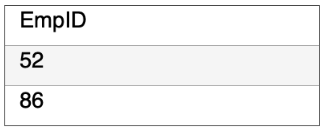


Salary
50000
45000
36000
12000

Figure 4: Salaries sorted in descending order.

The kth-highest salary is then defined as the value in the kth row of the table in Figure 4. (We will assume that there are at least k distinct salaries in the Employee table.) For example, when $k = 1$, the kth-highest salary is the highest salary, or 50000. Similarly, when $k = 2$, the kth-highest salary is the second-highest salary, or 45000.

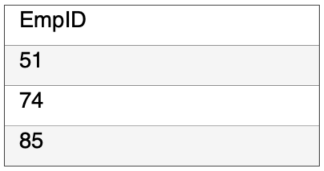
Note that when asked to compute the IDs of employees earning the kth-highest salary, the result set may contain more than one row. For example, for $k = 1$, two employees are tied for first place, as shown in Figure 5:



EmpID
52
86

Figure 5: IDs of employees with highest salary ($k = 1$).

For $k = 2$, there is a three-way tie, as shown in Figure 6:



EmpID
51
74
85

Figure 6: IDs of employees with second-highest salary ($k = 2$).