Database Query Assignment

Introduction

The assignment consists of two parts: database setup and performing queries to extract required information from the database.

The assignment has been completed using MySQL 9.1.0.

Database setup

First, the database `COMPANY1` is created and the `USE` statement is used to set it as the default database for all consequent requests.

```
CREATE DATABASE COMPANY1;
USE COMPANY1;
```

Create tables

The database contains two tables:

- `DEPT` containing the information about the departments,
- `EMP` with the employees' records.

The `DEPT` table should be set up first, as it is referenced via a foreign key in the `EMP` table.

```
CREATE TABLE DEPT
(
    DEPTNO INT PRIMARY KEY, -- Primary key: Department number
    DNAME  VARCHAR(50),     -- Department name
    LOC    VARCHAR(50)      -- Location of the department
);

CREATE TABLE EMP
(
    EMPNO    INT PRIMARY KEY, -- Primary key: Employee number
    ENAME    VARCHAR(50),     -- Employee name
    JOB      VARCHAR(50),     -- Job title
    MGR      INT,             -- Manager's employee number
    HIREDATE DATE,            -- Date of hire
    SAL      DECIMAL(10, 2),  -- Salary
    COMM     DECIMAL(10, 2),  -- Commission
    DEPTNO   INT,             -- Foreign key: Department number
    FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO)
);
```

The `SAL` and `COMM` fields that contain monetary values are stored as `DECIMAL` values. `DECIMAL` is a precise fixed-point type that does not suffer from the rounding errors typical for floating-point arithmetic (Brylow, 2019, p. 80; Oracle Corporation, no date a).

After the tables are set up, they are populated with the data provided in the assignment.

Query the database

1. List employees based on their salary

> List all Employees whose salary is greater than 1,000 but not 2,000. Show the Employee Name, Department and Salary

The following command joins the records from the `EMP` and `DEPT` tables based on the department number, filters the records based on the salary value, and displays the required fields:

```
SELECT EMP.ENAME, DEPT.DNAME, EMP.SAL
FROM EMP
         JOIN DEPT on EMP.DEPTNO = DEPT.DEPTNO
WHERE SAL BETWEEN 1000 AND 2000;
```

The returned data is below:

ENAME	DNAME	SAL
ALLEN	SALES	1600.00
WARD	SALES	1250.00
MARTIN	SALES	1250.00
TURNER	SALES	1500.00
ADAMS	RESEARCH	1100.00
MILLER	ACCOUNTING	1300.00

Alternatively, it's possible to reduce the dataset before the `JOIN` operation, which can improve performance on larger datasets. For this purpose, a subquery can be used to get the matching employees and select the fields from the resulting derived table:

```
SELECT FilteredEmp.ENAME, DEPT.DNAME, FilteredEmp.SAL
FROM (SELECT ENAME, SAL, DEPTNO
      FROM EMP
      WHERE SAL BETWEEN 1000 AND 2000) AS FilteredEmp
         INNER JOIN DEPT ON FilteredEmp.DEPTNO = DEPT.DEPTNO;
```

It is also possible to use a common table expression (CTE) for the same purpose by invoking a `WITH` clause. Similarly to derived tables, CTEs allow to pre-fetch data, but also have multiple advantages: CTEs can be reused multiple times in a query, referenced in other CTEs and in itself forming a recursive query. A recursive query can be useful for exploring hierarchical data, such as manager-subordinate relation in the `EMP` table.
The solution for the current task involving a CTE:

```
WITH FilteredEmp AS (SELECT ENAME, SAL, DEPTNO
                     FROM EMP
                     WHERE SAL BETWEEN 1000 AND 2000)
```

```sql
SELECT F.ENAME, D.DNAME, F.SAL
FROM FilteredEmp F
         INNER JOIN DEPT D ON F.DEPTNO = D.DEPTNO;
```

2. Count employees based on criteria

> Count the number of people in department 30 who receive a salary and a commission.

The command below counts the entries in the `EMP` table where the `SAL` and `COMM` values are greater than zero.

```sql
SELECT COUNT(*)
FROM EMP
WHERE DEPTNO = 30
  AND SAL > 0
  AND COMM > 0;
```

The result of the query execution is below:

COUNT(*)
3

It is worth noting that there's a record of the employee with `EMPNO = 7844` with `COMM` value set to `0.00`, which contradicts the premise from the database description in the assignment: "not all employees receive commission, in which case the COMM field is set to null" (University of Essex Online, 2024).

EMPNO	ENAME	JOB	...	COMM	DEPTNO
7844	TURNER	SALESMAN	...	0.00	30

Without domain-specific knowledge, it is unclear if the zero value represents an operator error or is legitimate and required for the salesman position (other employees who receive commission are also salesmen).

In case the employee with commission of zero should be included in the count, `WHERE` clause must be formulated as follows: `WHERE DEPTNO = 30 AND SAL IS NOT NULL AND COMM IS NOT NULL`.

As an alternative solution, a subquery or a CTE can be used. This query will first filter the table and return `1` for each matching record. Then the returned values are counted.

```
SELECT COUNT(*)
FROM (SELECT 1
      FROM EMP
      WHERE DEPTNO = 30
        AND SAL > 0
        AND COMM > 0) AS FilteredEMP;
```

Another option would be to use conditional count and specify the criteria inside the `COUNT` function, which can be more explicit and readable.

```
SELECT COUNT(CASE WHEN DEPTNO = 30 AND SAL > 0 AND COMM > 0 THEN
1 END)
FROM EMP;
```

It is also possible to execute an `EXISTS` subquery. While unnecessary here, an `EXISTS` subquery can be valuable for cross-referencing databases. For each record in the `EMP` table, a subquery is executed. If the subquery yields any results, it is evaluated as `TRUE`, and the current record is included in the count.

```
SELECT COUNT(*)
FROM EMP E
WHERE EXISTS (SELECT 1
              FROM EMP
              WHERE DEPTNO = 30
                AND EMP.SAL > 0
```

```
                  AND EMP.COMM > 0
                  AND E.EMPNO = EMP.EMPNO);
```

3. Display details for employees who match selection criteria

> Find the name and salary of the employees that have a salary greater or equal to 1,000 and live in Dallas.

The query combines the `EMP` and `DEPT` tables based on the `DEPTNO` field. Then, the dataset is filtered based on the department location and the employee salary.

```
SELECT EMP.ENAME, EMP.SAL
FROM EMP
         JOIN DEPT on EMP.DEPTNO = DEPT.DEPTNO
WHERE DEPT.LOC = 'DALLAS'
  AND EMP.SAL ⩾ 1000;
```

The result of the query execution is below:

ENAME	SAL
JONES	2975.00
SCOTT	3000.00
ADAMS	1100.00
FORD	3000.00

It is possible to rewrite the query with a CTE to find the department located in Dallas first, then generate the intersection of the resulting `DallasDepartments` table with the `EMP` table, and filter the values based on the employee salary. This allows to reduce the volume of the dataset and only work with the records of the employees located in Dallas.

```
WITH DallasDepartments AS (SELECT DEPT.LOC, DEPT.DEPTNO
                           from DEPT
```

```
                             WHERE LOC = 'DALLAS')
SELECT EMP.ENAME, EMP.SAL
FROM EMP
         INNER JOIN DallasDepartments on EMP.DEPTNO =
DallasDepartments.DEPTNO
WHERE SAL ⩾ 1000;
```

4. Find departments without employees

> Find all departments that do not have any current employees.

To locate the departments without any employees, `LEFT JOIN` operation can be used: it will associate each department record with the records of the employees of that department. If no matching employee record is found, `NULL` values are appended to the resulting row in place of the values from the `EMP` table. If the `NULL` value occurs in the field that is originally not nullable in the `EMP` table, such as `EMPNO`, it's an indicator that no matching record was found.

```
SELECT DEPT.DNAME
FROM DEPT
         LEFT JOIN EMP ON DEPT.DEPTNO = EMP.DEPTNO
WHERE EMP.EMPNO IS NULL;
```

The result for this query is an empty set. There are employees in each of the three departments.

To verify that the query works as required, a temporary record can be added to the `DEPT` table:

```
INSERT INTO DEPT
    VALUE (40, 'TEST DEPT', 'SEATTLE');
```

After the record is added, the query above returns the name for the newly added department, which confirms that the query is well-formed. The record can be deleted after the queries are tested to return the database to its initial state.

DNAME
TEST DEPT

Another option could be to use the `NOT EXISTS` subquery to cross-reference the tables. For each row in the `DEPT` table, a subquery is executed to determine whether there is an employee referencing the current department, and if there are none, the row is included in the output. This approach focuses on the filtering criteria and can be perceived as more explicit. However, its downside is that the subquery is executed for each row in the `DEPT` table, which can be ineffective on larger tables.

```
SELECT DNAME
FROM DEPT D
WHERE NOT EXISTS (SELECT 1
                  FROM EMP E
                  WHERE E.DEPTNO = D.DEPTNO);
```

5. Display aggregate department data

> List the department number, the average salary, and the number/count of employees of each department.

Both `AVG` and `COUNT` functions operate on sets of data, and a way to gather a set is to use the `GROUP BY` clause. The query below will perform `LEFT JOIN` of the tables `DEPT` and `EMP` (although with the existing dataset an `[INNER] JOIN` would suffice, as there are no departments without employees as proven above) and group the results based on their `DEPTNO` value. This allows to execute the aforementioned functions to perform the calculations. `COALESCE()` function can be used to avoid `NULL` values for average salary in case there are no employee records for the department. `ROUND()` function is used to round the monetary values to the closest 1/100th of a currency unit.

```
SELECT DEPT.DEPTNO, ROUND(COALESCE(AVG(EMP.SAL), 0), 2),
COUNT(EMP.EMPNO)
FROM DEPT
         LEFT JOIN EMP on DEPT.DEPTNO = EMP.DEPTNO
GROUP BY DEPT.DEPTNO;
```

The result of the query execution is below:

DEPTNO	ROUND(COALESCE(AVG(EMP.SAL), 0), 2)	COUNT(EMP.EMPNO)
10	2916.67	3
20	2175.00	5
30	1566.67	6

Alternatively, it's possible to query the `EMP` table for each department record. This solution potentially offers more options for querying the `EMP` table to add more filters or run additional queries, if adds some overhead by executing additional queries.

```
SELECT DEPT.DEPTNO,
       (SELECT ROUND(COALESCE(AVG(EMP.SAL), 0), 2)
        FROM EMP
        WHERE EMP.DEPTNO = DEPT.DEPTNO) AS AVG_SALARY,
       (SELECT COUNT(*)
        FROM EMP
        WHERE EMP.DEPTNO = DEPT.DEPTNO) AS EMPLOYEE_COUNT
FROM DEPT;
```

Conclusion

The structured query language offers great flexibility when it comes to composing requests to a database. This allows to write complex queries to poll data from a table or a set of tables, analyze the information stored in the database, cross-reference

multiple tables based on the business needs. Besides, this flexibility allows to optimize the queries for performance when working with large amounts of data. However, it is important that the database specialist knows the specifics of the database management software (DBMS) they work with, as the performance of different commands depends on the implementation details of the specific DBMS and server settings (Čapligins and Ermuiža, 2016; Oracle Corporation, no date b).
The choice of the database management software and the database engine not only affects the performance of a database and the commands, but also has impact on the sustainability of the implemented database solution (Miranskyy *et al.*, 2018).

References and Bibliography

Brookshear, J.G. and Brylow, D. (2020) *Computer science: an overview*. 13th edition, global edition. NY, NY: Pearson.

Čapligins, O. and Ermuiža, A. (2016) 'MySQL database management system forks comparison and usage'. ResearchGate. Available at: https://www.researchgate.net/profile/Andrejs-Ermuiza/publication/304394705_MySQL_Database_Management_System_Forks_Comparison_and_Usage/links/576e1eb308ae10de6395d848/MySQL-Database-Management-System-Forks-Comparison-and-Usage.pdf (Accessed: 12 January 2025).

Miranskyy, A.V. *et al.* (2018) 'Database engines: Evolution of greenness', *Journal of Software: Evolution and Process*, 30(4), p. e1915. Available at: https://doi.org/10.1002/smr.1915.

Oracle Corporation (no date a) *MySQL :: MySQL 9.1 Reference Manual :: 14.25 Precision Math*. Available at: https://dev.mysql.com/doc/refman/9.1/en/precision-math.html (Accessed: 12 January 2025).

Oracle Corporation (no date b) *MySQL :: MySQL 9.1 Reference Manual :: 10.1 Optimization Overview*. Available at:

https://dev.mysql.com/doc/refman/9.1/en/optimize-overview.html (Accessed: 12 January 2025).

Oracle Corporation (no date c) *MySQL :: MySQL 9.1 Reference Manual :: 15 SQL Statements*. Available at: https://dev.mysql.com/doc/refman/9.1/en/sql-statements.html (Accessed: 12 January 2025).

University of Essex Online (2024) *LCS_PCOM7E October 2024: Prerequisite Activity for Assignment 1 Part 3 of 3* | *UoEO*. Available at: https://www.my-course.co.uk/mod/page/view.php?id=1077883 (Accessed: 12 January 2025).