Database Lab 2 - Querying Multiple Tables

Objectives

- To provide practice in the use of multiple table queries
- To provide experience in formulating nested queries
- To understand and overcome the limitations of SQL regarding nested queries
- To familiarise with the database schema to answer questions in test 2

Approach

This lab worksheet provides a combination of initial non-nested and nested queries for you to practice. You could make use of the SQLite on GitHub CodeSpaces to develop and test your queries before checking the answers presented. The test 2 contains questions based on the schema of the given database. It would be good to familiarise yourself with it by going through the following queries before taking the test. You can fork this repository as a base https://github.com/hss70/SQLMultipleTableLab-Sheet

Setting up the test tables

To complete the example queries and problems provided in this worksheet you will have to set up a number of tables and import some data into them. Use the following declarations to create the tables:

```
CREATE TABLE Item (
 ItemName VARCHAR (30) NOT NULL,
 ItemType CHAR(1) NOT NULL,
 ItemColour VARCHAR(10),
 PRIMARY KEY (ItemName));


CREATE TABLE Employee (
 EmployeeNumber SMALLINT UNSIGNED NOT NULL ,
 EmployeeName VARCHAR(10) NOT NULL ,
 EmployeeSalary INTEGER UNSIGNED NOT NULL ,
 DepartmentName VARCHAR(10) NOT NULL REFERENCES Department,
 BossNumber SMALLINT UNSIGNED NOT NULL REFERENCES Employee,
 PRIMARY KEY (EmployeeNumber));

CREATE TABLE Department (
 DepartmentName VARCHAR(10) NOT NULL,
 DepartmentFloor SMALLINT UNSIGNED NOT NULL,
 DepartmentPhone SMALLINT UNSIGNED NOT NULL,
 EmployeeNumber SMALLINT UNSIGNED NOT NULL REFERENCES
 Employee,
PRIMARY KEY (DepartmentName));
```

```
CREATE TABLE Sale (
 SaleNumber INTEGER UNSIGNED NOT NULL,
 SaleQuantity SMALLINT UNSIGNED NOT NULL DEFAULT 1,
 ItemName VARCHAR(30) NOT NULL REFERENCES Item,
 DepartmentName VARCHAR(10) NOT NULL REFERENCES Department,
 PRIMARY KEY (SaleNumber));
```

```
CREATE TABLE Supplier (
 SupplierNumber INTEGER UNSIGNED NOT NULL,
 SupplierName VARCHAR(30) NOT NULL,
 PRIMARY KEY (SupplierNumber));
```

```
CREATE TABLE Delivery (
 DeliveryNumber INTEGER UNSIGNED NOT NULL,
 DeliveryQuantity SMALLINT UNSIGNED NOT NULL DEFAULT 1,
 ItemName VARCHAR(30) NOT NULL REFERENCES Item,
 DepartmentName VARCHAR(10) NOT NULL REFERENCES Department,
 SupplierNumber INTEGER UNSIGNED NOT NULL REFERENCES
  Supplier,
 PRIMARY KEY (DeliveryNumber));
```

The data for these tables is held in the files item.txt , employee.txt , department.txt , sale.txt , supplier.txt , and delivery.txt . Use INSERT INTO to enter all the data.

For each table use the command `SELECT * FROM tablename` to check that the data has been entered into each table correctly.

Some Queries to Try...

This section provides some examples of problems and their translation into an SQL query that will answer the problem. For each case, take time to identify how the query might have been produced from the problem statement, and to identify how the query operates. Try out each query in SQL on the database using GitHub CodeSpaces before looking at the answers.

1. What are the names of employees in the Marketing Department?

```
SELECT EmployeeName
FROM Employee
WHERE DepartmentName = 'Marketing'
```

2. Find the items sold by the departments on the second floor.

```
SELECT DISTINCT ItemName
FROM Sale, Department
```

```
WHERE Sale.DepartmentName =
Department.DepartmentName AND
Department.DepartmentFloor = 2;
```

Note that this query performs an equijoin on Sale and Department.

```
SELECT DISTINCT ItemName
FROM (Sale NATURAL JOIN Department)
WHERE Department.DepartmentFloor = 2;
```

Now replace NATURAL JOIN with JOIN in the same query. Was there any difference in the result? Why / Why not?

3. Identify by floor the items available on floors other than the second floor

```
SELECT DISTINCT ItemName,
  Department.DepartmentFloor AS 'On
  Floor'
FROM Delivery, Department
WHERE Delivery.DepartmentName = Department.DepartmentName
AND
Department.DepartmentFloor <> 2
ORDER BY Department.DepartmentFloor, ItemName;
```

4. Find the average salary of the employees in the Clothes department

```
SELECT AVG(EmployeeSalary)
FROM Employee
WHERE DepartmentName = 'Clothes';
```

5. Find, for each department, the average salary of the employees in that department and report by descending salary.

```
SELECT DepartmentName, AVG(EmployeeSalary) AS 'Average
 Salary'
FROM Employee
GROUP BY DepartmentName
ORDER BY 'Average Salary' DESC;
```

6. List the items delivered by exactly one supplier (i.e. the items always delivered by the same supplier).

```
SELECT ItemName
FROM Delivery
GROUP BY ItemName HAVING COUNT(DISTINCT SupplierNumber) =
1;
```

7. List the suppliers that deliver at least 10 items.

```
SELECT Supplier.SupplierNumber, Supplier.SupplierName
FROM Delivery, Supplier
WHERE   Delivery.SupplierNumber    =
Supplier.SupplierNumber   GROUP   BY
Supplier.SupplierNumber,
Supplier.SupplierName         HAVING
COUNT(DISTINCT    Delivery.ItemName)
>= 10;
```

Note: The HAVING clause provides a simple form of WHERE clause on the GROUP BY part of the SELECT statement. Think of it as a WHERE clause that is applied to the result returned by the query, rather than to the data in the tables being queried. You cannot use it to compare with values that can only be computed through a further query. In these cases, a nested query would be required.

8. Count the number of direct employees of each manager

```
SELECT Boss.EmployeeNumber,
 Boss.EmployeeName, COUNT(*) AS
 'Employees'
FROM Employee AS Worker, Employee AS Boss
WHERE Worker.BossNumber = Boss.EmployeeNumber
GROUP BY Boss.EmployeeNumber, Boss.EmployeeName;
```

Note that this example makes use of Tuple Variables to distinguish between the Equi Join of Employee with itself when we are querying the recursive relationship on the Employee table.

9. Find, for each department that sells items of type 'E' the average salary of the employees.

```
SELECT Department.DepartmentName,
  AVG(EmployeeSalary) AS 'Average Salary'
FROM Employee, Department, Sale, Item
WHERE Employee.DepartmentName =
Department.DepartmentName AND
Department.DepartmentName =
Sale.DepartmentName AND Sale.ItemName =
Item.ItemName
AND ItemType = 'E'
GROUP BY Department.DepartmentName;
```

Note that we could try to translate this query into one using joins. However, we would have to be careful in doing this, because the tables Employee and Department have both DepartmentName and EmployeeNumber in common (to establish two different relationships). We are therefore better to use an Equi-Join to establish this query.

10. Find the total number of items of type 'E' sold by departments on the second floor

```
SELECT SUM(SaleQuantity) AS 'Number of Items'
FROM Department, Sale, Item
WHERE Department.DepartmentName =
Sale.DepartmentName AND
Sale.ItemName = Item.ItemName
AND ItemType = 'E' AND
DepartmentFloor = '2';
```

Translate this query into its NATURAL JOIN equivalent, and demonstrate that your translation produces the same results.

11. What is the average delivery quantity of items of type 'N' delivered by each company?

```
SELECT Delivery.SupplierNumber,
SupplierName, Delivery.ItemName,
AVG(Delivery.DeliveryQuantity) AS
'Average Quantity'
FROM ((Delivery NATURAL JOIN Supplier) NATURAL JOIN Item)
WHERE Item.ItemType = 'N'
GROUP BY Delivery.SupplierNumber,
SupplierName, Delivery.ItemName ORDER BY
Delivery.SupplierNumber, SupplierName,
'Average Quantity' DESC,
Delivery.ItemName;
```

Would re-ordering the JOINs create a more efficient query in this case? If the query were translated into an EQUI JOIN equivalent, could we produce a more efficient query? Try out your translation into the EQUI JOIN equivalent.

Nested Queries

The section below provides examples of problem statement and their translation into an SQL query using set of nested queries. Study them carefully so that you understand how this translation process proceeds. You could try to solve them yourself using GitHub CodeSpaces before looking at the answers. Was your answer different than the answers given below? Did they produce the same results?

1. What are the names of items sold by departments on the second floor? This was previously solved in the preceding section by the use of a join. However, it could be more efficiently solved by using an inner query:

```
SELECT DISTINCT ItemName
FROM Sale
WHERE DepartmentName IN
      (SELECT DepartmentName
  FROM Department
  WHERE DepartmentFloor = 2);
```

2. Find the salary of Clare's manager.

```
SELECT EmployeeName, EmployeeSalary
FROM Employee
WHERE EmployeeNumber =
      (SELECT BossNumber
      FROM Employee
      WHERE EmployeeName = 'Clare');
```

3. Find the name and salary of the managers with more than two employees

```
SELECT EmployeeName, EmployeeSalary
FROM Employee
WHERE EmployeeNumber IN
      (SELECT BossNumber
      FROM Employee
      GROUP BY BossNumber HAVING COUNT(*) > 2);
```

4. List the names of the employees who earn more than any employee in the Marketing department

```
SELECT EmployeeName, EmployeeSalary
FROM Employee
WHERE EmployeeSalary >
      (SELECT MAX(EmployeeSalary)
      FROM Employee
      WHERE DepartmentName = 'Marketing');
```

5. Among all the departments with a total salary greater than

£25000, find the departments that sell Stetsons.

```
SELECT DISTINCT DepartmentName
FROM Sale
WHERE ItemName = 'Stetsons' AND DepartmentName IN
      (SELECT DepartmentName
      FROM Employee
      GROUP BY DepartmentName HAVING
           SUM(EmployeeSalary) > 25000);
```

6. Find the suppliers that deliver compasses and at least one other kind of item

```
SELECT DISTINCT Delivery.SupplierNumber,
Supplier.SupplierName FROM (Supplier
NATURAL JOIN Delivery)
WHERE (ItemName <> 'Compass' AND
      SupplierNumber IN (SELECT
      SupplierNumber
      FROM Delivery
      WHERE ItemName = 'Compass'));
```

7. Find the suppliers that deliver compasses and at least three other kinds of item

```
SELECT DISTINCT Delivery.SupplierNumber,
Supplier.SupplierName FROM (Supplier
NATURAL JOIN Delivery) WHERE
SupplierNumber IN
      (SELECT SupplierNumber
      FROM Delivery
      WHERE ItemName = 'Compass')
      GROUP BY Delivery.SupplierNumber,
      Supplier.SupplierName
```

```
            HAVING COUNT(DISTINCT ItemName) > 3;
```

8. List the departments for which each item delivered to the department is delivered to some other department as well

```
SELECT DISTINCT DepartmentName
FROM Delivery AS Delivery1
WHERE NOT EXISTS
      (SELECT *
       FROM Delivery AS Delivery2
      WHERE Delivery2.DepartmentName =
      Delivery1.DepartmentName
      AND ItemName NOT IN
             (SELECT ItemName
              FROM Delivery AS Delivery3
              WHERE Delivery3.DepartmentName <>
              Delivery1.DepartmentName));
```