Lab - 9: SQL Queries

Team Members:

Jay Sanjaybhai Patel (jy451478@dal.ca)

Kenil Kevadiya (kn486501@dal.ca)

Date:

March 21, 2024

Subject:

Software Development Concepts

Professor:

Michael McAllister

Part 1 - Starter Queries

Q - 1. How many territories are in each region?

→ USE csci3901;
SELECT r.RegionDescription, COUNT(t.TerritoryID)
FROM region r, territories t
WHERE r.RegionID = t.RegionID
GROUP BY r.RegionDescription;

Q - 2. Which products need to be reordered?

→ USE csci3901;
SELECT ProductID, ProductName
FROM products
WHERE ReorderLevel > 0

Q - 3. Which orders have not been shipped yet?

→ USE csci3901;
SELECT OrderID, OrderDate
FROM orders
WHERE ShippedDate is null

Q - 4. Which orders were shipped to a city different than the city of the customer's headquarters?

→ USE csci3901;
SELECT o.OrderID, o.OrderDate, o.CustomerID, c.CompanyName
FROM orders o, customers c
WHERE o.CustomerID = c.CustomerID and o.ShipCity != c.City;

Part 2 - Medium Queries

Q - 5. How many orders were sent by each shipper?

→ USE csci3901;
SELECT s.ShipperID, s.CompanyName, count(o.OrderID)
FROM shippers s, orders o
WHERE s.ShipperID = o.ShipVia
GROUP BY s.ShipperID

Q - 6. How many customers did each employee get an order from in the first quarter of 1998? (The first quarter of the year is January to March)

→
```
USE csci3901;
SELECT o.EmployeeID, count(o.OrderID)
FROM customers c, orders o
WHERE c.CustomerID = o.CustomerID and YEAR(o.orderDate) = 1998 and
QUARTER(o.orderDate) = 1
GROUP BY o.EmployeeID;
```

Q - 7. What is the cost of order 10256? (All discounts are currently 0 in the database)

→
```
USE csci3901;
SELECT OrderID,  SUM(UnitPrice * Quantity) As Total_Cost
FROM orderdetails
WHERE OrderID = 10256
```

Q - 8. What is the total value of orders in 1997 that were sent via each shipper?

→
```
USE csci3901;
SELECT o.ShipVia, SUM(d.UnitPrice * d.Quantity) AS TotalValue
FROM orders o, orderdetails d
WHERE o.OrderID = d.OrderID and YEAR(o.ShippedDate) = 1997
GROUP BY o.ShipVia;
```

Part 3 - Queries needing a bit more thought

Q - 10. Who is the top salesperson in the fourth quarter of 1997?

→
```
USE csci3901;
SELECT e.EmployeeID, e.FirstName, e.LastName, count(o.orderID) As TotalOrders
FROM employees e, orders o
WHERE e.EmployeeID = o.EmployeeID and YEAR(o.OrderDate) = 1997 and
QUARTER(o.OrderDate) = 4
GROUP BY e.EmployeeID
ORDER BY TotalOrders DESC
LIMIT 1;
```

Q - 11. Who are the supervisors in the company?

→ USE csci3901;
SELECT e.EmployeeID, e.LastName, e.FirstName
FROM employees e
JOIN (
  SELECT DISTINCT ReportsTo
  FROM employees
) AS subquery ON e.EmployeeID = subquery.ReportsTo;

Q - 12. How many people directly report to each of the supervisors?

→ USE csci3901;
SELECT e.EmployeeID, e.LastName, e.FirstName
FROM employees e
JOIN (
  SELECT DISTINCT ReportsTo
  FROM employees
) AS distinct_reports_to ON e.ReportsTo = distinct_reports_to.ReportsTo;

Q - 13. Which customers bought more than $5000 in products in 1997 that could be traced back to a single supplier? We might want to give these customers a discount to avoid having them buy directly from our supplier…

→ USE csci390;
Select DISTINCT CustomerID from (
  SELECT c.CustomerID, o.OrderID, o.OrderDate, o.ShippedDate, o.ShipVia,
  SUM(d.UnitPrice * d.Quantity) AS TotalValue
  FROM customers c, orders o, orderdetails d
  Where c.CustomerID = o.CustomerID and o.OrderID = d.OrderID
  GROUP BY c.CustomerID, o.ShipVia
  HAVING TotalValue > 5000 and YEAR(o.ShippedDate) = 1997)
As Subquery;

Broadening Questions

Q - 1: Describe a strategy that you can use to develop a query for a given question. Aim for a strategy that will be portable to new scenarios, will develop a query quickly, and will be efficient.

Ans.: Below is a strategy to make SQL queries according to various situations:

→ **Carefully analyze the question** to determine which tables to look for the data you need. Divide complicated questions into simpler sub-questions.

→ **Identify the tables** that hold the required data based on the inquiry. Examine the relationships between tables with the help of foreign keys.

→ Choose the **specific columns** that you require from the identified tables. To increase efficiency, only include the columns that are necessary.

→ Use the **WHERE** clause to narrow down the information according to particular criteria. Apply **logical operators** in order to effectively combine conditions.

→ Use **JOIN** clauses to establish connections between tables containing data that are related to each other.

→ Think of using **aggregate functions**. if the topic calls for computations or summaries.

→ If the question being asked requires grouped results, group the data according to particular columns by using the **GROUP BY** clause.

→ To sort the results according to a certain column (either ascending or descending), use the **ORDER BY** clause.

→ Consider **NULL values** if they have an impact on the query's outcomes. Use IS NULL or IS NOT NULL as needed.

→ **Execute the query and review the results.** Modify the query until it gives the desired response, make adjustments based on the output.

Advantages of above strategy:

→ This method emphasizes on understanding the underlying logic, allowing it to be modified for new queries and database.

→ Writing effective queries that get data quickly only requires you to focus on choosing relevant data and utilizing the appropriate clauses.

→ Writing queries can be much faster when difficult questions are broken down and addressed step-by-step.

Q - 2: Explain which characteristics of queries (1-13) suggest that a subquery or avoiding anti-patterns might be useful. Not all queries require a subquery.

Ans.: Subqueries are not necessary for every query, although a few characteristics suggest their usefulness:

→ Subqueries have the ability to compute data and then utilise the output to filter the outer query. For instance, in questions 11, 12, and 13, we utilize subqueries to retrieve data, and then perform subsequent operations on that data to obtain the desired result.

→ Subqueries are capable of handling complex data structures and complicated logic with nested conditions.

Anti-patterns to Avoid:

→ **Extremely long WHERE clauses** with many conditions might be hard to understand and maintain. For clarity, divide them into subqueries.

→ **Excessive use of self-joins** might degrade performance. Think about using different strategies, such as subqueries or correlated subqueries.

→ Using **SELECT *** to select every column is wasteful and may result in the retrieval of extraneous data. Instead, choose a few columns.