Joins

Exercise Handout

Contents

Overview

In this set of exercises, you will first work with the Orders, Customers, Order Details and Products tables to find out detailed information about the orders on the system.

In the second exercise, you will investigate individual customers' ordering patterns.

Objectives

At the end of this activity, you will be able to:

- write a query that uses an inner join to select rows from two tables.
- write a query that filters joined data.
- write a query that uses an inner join to select rows from more than two tables.
- write a query that investigates the difference between left and right outer joins.

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Launch SQL Server Management Studio.
3. Connect to the server.

Exercise 1: Join fundamentals

You want to write a query about UK-based customers and the orders that they have placed.

In this exercise, you will use inner joins, the fundamental type of join in SQL.

The main tasks for this exercise are as follows:

1. Write a query that selects the customer id, city and company and contact names from the Customers table and the order number and order date from the Orders table.
2. Modify the query so that it only picks up orders placed by UK Customers and sorts the results by customer and order date.
3. Modify the query so that it also includes details about the name of the product that was ordered, and how many units of each product were ordered.

Task 1: Write a query that selects rows from the Customers and Orders tables

1. Create a new query and save it with a name of "UKCustomerOrders.sql".
2. Write a query that uses the Northwind database and selects the CustomerID, CompanyName, ContactName and City columns from the Customers table. Alias the table as 'c'.
3. Execute the query and verify that 91 rows are returned.
4. Modify the query to join rows from the Orders table that have the same value in the CustomerID column. Alias Orders as 'o'.
5. Add the OrderID and OrderDate columns from the Orders table to the select list.
6. Execute the query and verify that 830 rows are returned.

Task 2: Write a query that filters and sorts the results

1. Modify your existing query.
2. Add a where clause to limit the rows returned to those Customers whose Country column is equal to UK.
3. Execute the query and verify that 56 rows are returned, all of whose City values are British cities.
4. Add an order by clause to ensure that results are sorted on CompanyName then OrderDate
5. Execute the query and verify that 56 rows are returned, the first one is for AROUT from November 1996 and the last one is for SEVES from February 1998.

Task 3: Write a query that includes rows from more than two tables

1. Make a copy of your existing query.
2. Join the [Order Details] table aliased as 'od'. Use the OrderID column in Orders and Order Details to find matching rows.
3. Add the ProductID and Quantity columns to the select list. You can also remove the City column from the earlier queries – that was just to add a visual verification that you were only getting UK customers.
4. Execute the query and verify that 135 rows are returned – we're getting more rows back because we're selecting every row down at the Order Detail level now.
5. Join yet another table – the Products table. Alias it as 'p' and use the ProductID column from Order Details to find the matching rows. Include the ProductID and ProductName columns in your select list.
6. Execute the query and verify that the first order is for 25 lots of Guarana Fantastica and the last one is for 20 Scottish Longbreads.
7. [Extra Credit] limit the results to only include products in the Seafood category and display the category name as well. (18 rows)

   HINT: You will need to add the Categories table to your existing query to do this.

Exercise 2: Investigate outer joins

In this exercise you will investigate left and right outer joins. You will also revisit aggregation from an earlier exercise.

The main tasks for this exercise are as follows:

1. Create a query that gets a count of Customers.
2. Create a query that selects the Company Name from Customers and the number of orders for those customers from the Orders table.
3. Modify the query to retrieve all customers, regardless of whether they have placed orders and add a column showing the minimum order date to the select list.

Task 1: Create a query that counts customers

1. Create a new query and save it with a name of "CustomerOrderAnalysis.sql".
2. Write a query that uses the Northwind database and displays a count of all the rows in the Customers table.
3. Execute the query and make a note of the result.

Task 2: Write a query that groups orders based on the customer's name

1. Create a new query that selects the CompanyName column from the Customers table and a count of OrderID columns from the Orders. Alias the count column as 'NumOrders'. You will need a group by clause.
2. Add an order by clause to sort the records in number of orders placed.
3. Execute the query and make a note of the number of rows returned. You will see that it differs from the number of customers. Take a few moments to think about why that might be. HINT: what is the lowest count of orders?

Task 3: Write a query that uses left and right outer joins

1. Modify the existing query.
2. Change the query so that it selects all of the rows from the Customers table and those rows from the Orders table where there is a match.
3. Execute the query and verify that it now returns 91 rows, the top two now being FISSA and Paris Specialités, both with a count of 0.
4. Change the left outer join to a right outer join and confirm that you're back to only seeing 89 rows. In this case, we get the same results as for an inner join.
5. Change the order in which the tables are listed so that the right outer join returns 91 rows.
6. Finaly, add another column to the query that selects the minimum order date from the Orders table and alias it as 'MinDate'.
7. Execute the query and note that we have a couple of null values for the inactive customers. The COUNT aggregate was able to come up with a total of zero for those customers but the MIN has no values to work on so it must return null. In an actual query, you might want to tidy that up so that it shows some text instead.

Learn. To Change.

QA.com