## Instructions:

* The assignment aims to reinforce SQL query skills using the **WideWorldImporters** database. You will write and execute SQL queries to retrieve and manipulate data using various SQL constructs covered in Chapter 8 of our SQL text.
* Write SQL queries that fulfill the requirements listed below.
* Use appropriate column names, functions, and sorting techniques where necessary.
* Ensure queries return meaningful results based on the dataset.
* Save your work as a .sql file and upload it to D2L Dropbox folder for the assignment. Use the following file naming convention: *yourFirstName\_yourLastName\_SQL\_HW\_06.sql*

**Section 1: Data Conversion with CAST**

1. Converting Data Types with CAST:
   * Retrieve the InvoiceID (integer) and calculate the total invoice amount by summing the ExtendedPrice (money) from the Sales.InvoiceLines table for each invoice.
   * Convert the total invoice amount to VARCHAR(20) using CAST and display both the original and converted values.
   * You will need to use a subquery in the FROM clause to get the total invoice amount:

SELECT InvoiceID, SUM(ExtendedPrice) AS TotalInvoiceAmount

FROM Sales.InvoiceLines

GROUP BY InvoiceID

1. Integer Division with CAST:
   * Retrieve the OrderID and Quantity from Sales.OrderLines, and divide Quantity by 3.
   * Display the result before and after using CAST to convert Quantity to DECIMAL(10,2), ensuring proper division.

**Section 2: Data Conversion with CONVERT**

1. Formatting Date Using CONVERT:
   * Retrieve InvoiceID and InvoiceDate from Sales.Invoices.
   * Use CONVERT to display InvoiceDate in the following formats:
     + 101 (MM/DD/YYYY)
     + 103 (DD/MM/YYYY)
     + 120 (YYYY-MM-DD HH:MI:SS)
2. Converting Real to Character Using CONVERT:
   * Retrieve TypicalWeightPerUnit from Warehouse.StockItems where TypicalWeightPerUnit is greater than 5.
   * Convert it to VARCHAR(20) using CONVERT and display both original and converted values.
3. Converting Money to Character Using CONVERT:
   * Retrieve the invoice ID and total invoice amount calculated by summing the ExtendedPrice from the Sales.InvoiceLines table for each invoice where the total is greater than 500. You will need to use a similar approach to build your query as in Problem 1.
   * Convert it to VARCHAR(20) using CONVERT with style 1 (commas as thousand separators) and style 0 (default).
   * Display the original and converted values.

**Section 3: Error Handling with TRY\_CONVERT**

1. Using TRY\_CONVERT to Handle Conversion Errors:
   * Attempt to convert StockItemName from Warehouse.StockItems to INT using TRY\_CONVERT.
   * Return NULL values for non-convertible data instead of throwing an error.

**Section 4: Other Data Conversion Functions**

1. Using STR to Convert Numeric Data:
   * Retrieve TypicalWeightPerUnit from Warehouse.StockItems where TypicalWeightPerUnit is greater than 5.
   * Convert it to CHAR(10) using STR and display both original and converted values.
2. Using CHAR and ASCII to Examine Character Codes:
   * Retrieve CustomerName from Sales.Customers where CustomerID is less than 10.
   * Extract the first letter of each CustomerName and display its ASCII code using ASCII.
3. Using NCHAR and UNICODE to Work with Unicode Data:
   * Retrieve CustomerName from Sales.Customers where CustomerID is less than 10.
   * Extract the first letter and display both its Unicode representation using UNICODE and its character equivalent using NCHAR.
4. Challenge: Using Multiple Conversions Together:
   * Retrieve InvoiceID, InvoiceDate, and calculate the total invoice amount by summing the ExtendedPrice from the Sales.InvoiceLines table for each invoice where the total is greater than 5000.
   * Convert InvoiceDate to VARCHAR(10) format YYYY-MM-DD using CONVERT.
   * Convert the total invoice amount to VARCHAR(20) using CAST.
   * Display all converted values along with original data.

**Grading Rubric**

Each query is worth \*\*10 points\*\*, evaluated based on the following criteria:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Excellent (10 - 9 pts)** | **Good (8 - 7 pts)** | **Satisfactory (6 - 5 pts)** | **Needs Improvement (4 - 0 pts)** |
| **Correctness of Query (50%)** | Query executes without errors and returns the expected result set. | Query executes with minor logical errors but mostly correct. | Query contains errors affecting correctness. | Query does not execute properly or is incorrect. |
| **Use of Required SQL Concepts (30%)** | Successfully applies the required SQL concepts. | Uses most required SQL concepts correctly. | Uses some SQL concepts but omits key elements. | SQL concepts are misused or omitted. |
| **Query Formatting & Readability (10%)** | Query is well-structured and properly formatted. | Query is readable but could use better formatting. | Query is difficult to read with inconsistent formatting. | Query lacks readability and proper structure. |
| **Use of Proper Column Naming and Aliasing (10%)** | All columns are named meaningfully. | Most columns are appropriately named. | Some columns are meaningfully named. | Column naming and aliasing are unclear or missing. |