Stored Procedures

(Use Adventure Works Database)

- Create a procedure InsertOrderDetails that takes OrderID, ProductID, UnitPrice, Quantiy, Discount as input parameters and inserts that order information in the Order Details table. After each order inserted, check the @@rowcount value to make sure that order was inserted properly. If for any reason the order was not inserted, print the message: Failed to place the order. Please try again. Also your procedure should have these functionalities
  Make the UnitPrice and Discount parameters optional
  If no UnitPrice is given, then use the UnitPrice value from the product table.
  If no Discount is given, then use a discount of 0.
  Adjust the quantity in stock (UnitsInStock) for the product by subtracting the quantity sold from inventory.
  However, if there is not enough of a product in stock, then abort the stored procedure without making *any* changes to the database.
  Print a message if the quantity in stock of a product drops below its Reorder Level as a result of the update.

- Create a procedure UpdateOrderDetails that takes OrderID, ProductID, UnitPrice, Quantity, and discount, and updates these values for that ProductID in that Order. All the parameters except the OrderID and ProductID should be optional so that if the user wants to only update Quantity s/he should be able to do so without providing the rest of the values. You need to also make sure that if any of the values are being passed in as NULL, then you want to retain the original value instead of overwriting it with NULL. To accomplish this, look for the ISNULL() function in google or sql server books online. Adjust the UnitsInStock value in products table accordingly.

- Create a procedure GetOrderDetails that takes OrderID as input parameter and returns all the records for that OrderID. If no records are found in Order Details table, then it should print the line: "The OrderID XXXX does not exits", where XXX should be the OrderID entered by user and the procedure should RETURN the value 1.

- Create a procedure DeleteOrderDetails that takes OrderID and ProductID and deletes that from Order Details table. Your procedure should validate parameters. It should return an error code (-1) and print a message if the parameters are invalid. Parameters are valid if the given order ID appears in the table and if the given product ID appears in that order.

Functions

Review SQL Server date formats on this website and then create following functions

http://www.sql-server-helper.com/tips/date-formats.aspx

- Create a function that takes an input parameter type datetime and returns the date in the format MM/DD/YYYY. For example if I pass in '2006-11-21 23:34:05.920', the output of the functions should be 11/21/2006

- Create a function that takes an input parameter type datetime and returns the date in the format YYYYMMDD

Views

Create a view vwCustomerOrders which returns CompanyName,OrderID,OrderDate, ProductID,ProductName,Quantity,UnitPrice,Quantity * od.UnitPrice

Create a copy of the above view and modify it so that it only returns the above information for orders that were placed yesterday

Use a CREATE VIEW statement to create a view called MyProducts. Your view should contain the ProductID, ProductName, QuantityPerUnit and UnitPrice columns from the Products table. It should also contain the CompanyName column from the Suppliers table and the CategoryName column from the Categories table. Your view should only contain products that are not discontinued.

Triggers

- If someone cancels an order in northwind database, then you want to delete that order from the Orders table. But you will not be able to delete that Order before deleting the records from Order Details table for that particular order due to referential integrity constraints. Create an Instead of Delete trigger on Orders table so that if some one tries to delete an Order that trigger gets fired and that trigger should first delete everything in order details table and then delete that order from the Orders table

- When an order is placed for *X* units of product *Y*, we must first check the Products table to ensure that there is sufficient stock to fill the order. This trigger will operate on the Order Details table. If sufficient stock exists, then fill the order and decrement *X* units from the UnitsInStock column in Products. If insufficient stock exists, then refuse the order (*i.e.* do not insert it) and notify the user that the order could not be filled because of insufficient stock.

Note: Based on the understanding candidate has to create a sample data to perform these queries.