AdventureWorks Database – 2019

Get the entity relationship diagram: Click here

1. Write a script that declares an integer variable called @myint. Assign 10 to the variable and then print it.

```
DECLARE @myint INT = 10;
PRINT(@myint);
```

2. Declare a variable of type tinyint and assign a value of 4000 to it. What is the result and why?.

```
DECLARE @num TINYINT = 4000;
PRINT(@num);
explain: Error comes when we assign value of 4000 to @num variable.  The tinyint
datatype has a range from 0 to 255.
```

3.What is the value of @ID after assigning the value? DECLARE @ID INT = 123.4567, SELECT @ID.

```
DECLARE @ID INT;
SELECT @ID = 123.4567;
PRINT(@ID);
```

Explain: the value of @ID is 123 because the datatype for that variable is INT. When we try to assign value of 123.4567 which is a float value, then mssql will fix it to INT instead.

4. Write a script that declares a VARCHAR(20) variable called @mystring. Assign 'This is a test' to the variable and print it.

```
DECLARE @mystring VARCHAR(20);
SET @mystring = 'This is a test';
PRINT(@mystring);
```

5. Write a script that declares two integer variables called @MaxID and @MinID. Use the variables to print the highest and lowest SalesOrderID value from the Sales.SalesOrderHeader table.

```
DECLARE @MinID INT, @MaxID INT;
SELECT @MinID = MIN(SalesOrderID), @MaxID = MAX(SalesOrderID)
FROM Sales.SalesOrderHeader;
PRINT(@MinID);
PRINT(@MaxID);
```

6. Write a script that declares an integer variable called @ID. Assign the value 70000 to the variable. Use the variable in a SELECT statement that returns all the rows from Sales.SalesOrderHeader table that have a SalesOrderID greater than the value of the variable.

```
DECLARE @ID INT = 70000;
SELECT *
FROM Sales.SalesOrderHeader
WHERE SalesOrderID > @ID;
```

7. Write a script that declares three variables, one integer variable called @ID, an NVARCHAR(50) variable called @FirstName and an NVARCHAR(50) variable called @LastName. Use a SELECT statement to set the value of the variables with the row from Person.Person table with BusinessEntityID = 1.
Print a statement in the "BusinessEntityID: FirstName LastName'' format.

```
DECLARE @ID INT, @FirstName NVARCHAR(50), @LastName NVARCHAR(50);
SELECT @ID = BusinessEntityID, @FirstName = FirstName, @LastName = LastName
FROM Person.Person
WHERE BusinessEntityID = 1;
PRINT(CONVERT(VARCHAR, @ID) + ': ' + @FirstName + ' ' + @LastName)
```

8. Write a script that declares an integer variable called @SalesCount. SET the value of the variable to total count of sales in the Sales.SalesOrderHeader table. Use the variable in a SELECT statement that shows the difference between the @Salescount and the count of sales by customer.

```
DECLARE @SalesCount INT;
SET @SalesCount = (SELECT COUNT(*) FROM Sales.SalesOrderHeader)
SELECT CustomerID, @SalesCount totalsales, COUNT(*) as total_order_by_customer,
@SalesCount - COUNT(SalesOrderID) AS diff
FROM Sales.SalesOrderHeader
GROUP BY CustomerID;
```

9. Write a batch that declares an integer variable called @Count to save the count of all the Sales.SalesOrderDetail records. Add an IF block that prints 'Over 100.000' if the value exceeds 100.000 Otherwise, print "100.000 or less".

```
DECLARE @Count INT = (SELECT COUNT(*) FROM Sales.SalesOrderDetail);
IF @count > 100000
    BEGIN
    PRINT('OVER 100000')
    END;
ELSE
    BEGIN
    PRINT('100.000 or less')
    END;
```

10. Write a batch that contains nested IF blocks. The outer block should check to see whether the month is October or November. If that is the case, print “The month is” and the month name.The inner block should check to make sure the inner the block fires.

```
DECLARE @MonthName VARCHAR(20) = DATENAME(MONTH,GETDATE());
IF @MonthName = 'October' OR @MonthName = 'November'
   BEGIN
   PRINT('The month is ' + @MonthName);
   IF @MonthName = 'October' OR @MonthName = 'September'
      BEGIN
      PRINT('The inner block fires');
      END;
   END;
ELSE
   BEGIN
   PRINT('this month is ' + @MonthName)
   END;
```

11. Write a batch that uses IF EXISTS to check to see whether there is a row in the Sales.SalesOrderHeader table that has SalesOrderID = 1. Print “There is a SalesOrderID = 1” or “There is not a SalesOrderID = 1” depending on the result.

```
IF EXISTS (SELECT * FROM Sales.SalesOrderHeader WHERE SalesOrderID = 1)
    BEGIN
   PRINT 'There is a SalesOrderID = 1'
   END;
ELSE
    BEGIN
   PRINT 'There is not a SalesOrderID = 1'
   END;
```

12. Write a script that contains a WHILE loop that prints out the letters A to Z. Use the function CHAR to change a number to a letter. Start the loop with the value 65. Here is an example that uses the CHAR function: DECLARE @Letter CHAR(1); SET @Letter = CHAR(65); PRINT(@Letter)

```
DECLARE @Letter CHAR(1), @num INT = 65;
WHILE @num < 100
BEGIN
    SET @Letter = CHAR(@num)
    PRINT(@Letter)
    IF @Letter = 'Z'
    BEGIN
```

```
        BREAK
    END;
    SET @num += 1
END;
```

13. Write a script that contains a WHILE loop nested inside another WHILE loop. The counter for the outer loop should count up from 1 to 100. The counter for the inner loop should count up from 1 to 5. Print the product of the two counters inside the inner loop.

```
DECLARE @Outercount INT = 1
DECLARE @Innercount INT;
WHILE @Outercount <= 100
    BEGIN
    PRINT('OuterLoop ' + CONVERT(VARCHAR, @Outercount))
    SET @Innercount = 1
    WHILE @Innercount <= 5
        BEGIN
        PRINT('Innerloop ' + CONVERT(VARCHAR, @Innercount))
        SET @Innercount += 1
        END;
    SET @Outercount += 1
END;
```

14. Change the script in Q13 so the inner loop exits instead of printing when the counter for the outer loop is evenly divisible by 5.

```
DECLARE @Outercount INT = 1
DECLARE @Innercount INT;
WHILE @Outercount <= 100
    BEGIN
    PRINT('OuterLoop ' + CONVERT(VARCHAR, @Outercount))
    SET @Innercount = 1
    WHILE @Innercount <= 5
        BEGIN
        IF @Outercount % 5 = 0
            BEGIN
                BREAK
        END;
        PRINT('Innerloop ' + CONVERT(VARCHAR, @Innercount))
    SET @Innercount += 1
    END;
    SET @Outercount += 1
END;
```

15. Write a script that contains WHILE loop that counts up from 1 to 100. Print “Odd” or “Even” depending on the value of the counter.

```
DECLARE @counter INT = 1;
WHILE @counter <= 100
BEGIN
    IF @counter % 2 = 1
        BEGIN
        PRINT('Odd: ' + CONVERT(varchar, @counter))
        END;
    ELSE
        BEGIN
        PRINT('Even: ' + CONVERT(varchar, @counter))
        END;
    SET @counter += 1
END;
```

16. What is the behavior of the following code and why? WHILE 1=1 PRINT “Oops”.

```
WHILE 1=1
BEGIN
PRINT('Oops')
END;
```

explain: This batch of codes is always True, so it’s generate multiple output ‘Oops’ until we stop the code.

17. Create a temp table called #CustomerInfo that contains CustomerID (INT), FirstName and LastName columns (NVARCHAR(50) for each one). Include CountOfSales(INT) and SumOfTotalDue (MONEY) columns. Populate the table with a query using the Sales.Customer, Person.Person, and Sales.SalesOrderHeader tables.

```
DROP TABLE IF EXISTS #CustomerInfo;
CREATE TABLE #CustomerInfo(
CustomerID INT,
FirstName NVARCHAR(50),
LastName NVARCHAR(50),
CountOfSales INT,
SumOfTotalDue MONEY
);

INSERT INTO #CustomerInfo (CustomerID, FirstName, LastName, CountOfSales,
SumOfTotalDue)
```

```
SELECT SOH.CustomerID, P.FirstName, P.LastName, COUNT(SOH.SalesOrderID),
SUM(SOH.TotalDue)
FROM Sales.Customer SC
JOIN Sales.SalesOrderHeader SOH ON SOH.CustomerID = SC.CustomerID
JOIN Person.Person P ON P.BusinessEntityID = SC.CustomerID
GROUP BY SOH.CustomerID, P.FirstName, P.LastName;
```

18. Change the code written in Q17 to use a table variable instead of a temp table.

```
DECLARE @CustomerInfo
TABLE (
CustomerID INT,
FirstName NVARCHAR(50),
LastName NVARCHAR(50),
CountOfSales INT,
SumOfTotalDue MONEY
);

INSERT INTO @CustomerInfo (CustomerID, FirstName, LastName, CountOfSales,
SumOfTotalDue)
SELECT SOH.CustomerID, P.FirstName, P.LastName, COUNT(SOH.SalesOrderID),
SUM(SOH.TotalDue)
FROM Sales.Customer SC
JOIN Sales.SalesOrderHeader SOH ON SOH.CustomerID = SC.CustomerID
JOIN Person.Person P ON P.BusinessEntityID = SC.CustomerID
GROUP BY SOH.CustomerID, P.FirstName, P.LastName;
```

19. Create a table variable with two integer columns, one of them an IDENTITY column. Use a WHILE loop to populate the table with 1000 random integers using the following formula. Use a second WHILE loop to print the values from the table variable one by one. CAST(RAND() * 100000 AS INT) + 1.

```
DECLARE @ID
TABLE (ID INT IDENTITY(1,1),
       RandNum INT
);
DECLARE @Counter INT = 1
WHILE @Counter <= 1000
BEGIN
INSERT INTO @ID (RandNum)
VALUES (CAST(RAND() * 100000 AS int) + 1)
SET @Counter += 1
END;
```

20. Create a temp table called #TestTempt with the following specification. ID int, val1 varchar(20), val2 varchar(30). The ID column should be the primary key.

```
DROP TABLE IF EXISTS #TestTempt
CREATE TABLE #TestTempt (
        ID int PRIMARY KEY,
        val1 varchar(20),
        val2 VARCHAR(30)
);
```

21. Instead of a temp table, create a table variable called @TestTemp with the same specification as the temp table in Q20.

```
DECLARE @TestTempt
TABLE (
    ID int PRIMARY KEY,
    val1 varchar(20),
    val2 VARCHAR(30)
);
```

22.Insert a row into @TestTemp. Modify the row and then select from the table variable.

```
DECLARE @TestTempt
TABLE (
ID int PRIMARY KEY,
val1 varchar(20),
val2 VARCHAR(30)
);
INSERT INTO @TestTempt (ID, val1, val2)
VALUES (1, 'Pratice', 'SQL')
```