Course Number & Title:	Using Structured Query language (SQL) Syntax
Course Author(s):	Richard Patlan
Author(s) Affiliation:	www.dbwebsite.com
Phone:	626-221-8435
Fax:	
E-Mail:	rpatlan@dbwebsite.com

Exercise 31:

Using Variables

A variable is a named object that stores values. Although the use and capability of variables varies from one programming language to another, the ability to define and store values in them is universal.

All local variables start with the @ sign. System variables start with two @@ signs. All variables must be declared and the data type specified.

NOTE: All 3 Parts must be executed at the same time I norder for values to be printed

EXAMPLE: Part 1

The following SQL code creates and specifies the datatype of three separate variables. The first statement declares a single variable @age of type int. The second statement declares two variables with type varchar(20). In the second statement a comma is required to separate the two different variables.

Write Code Syntax

```
--Part 1 - This code creates your variables by declaring them
Declare @age Int;
Declare @firstname Varchar(20), @lastname Varchar(20);
```

Result

Assigning Values to Variables Using SET

When you first create a variable no value is assigned to it. To assign value to a variable you use the SET statement

EXAMPLE:

The following SQL code creates the three variables and assigns each a value and nothing more.

Write Code Syntax

```
--Part 2 - This code declares your variables and assignes them values
Declare @age Int;
Declare @firstname Varchar(20), @lastname Varchar(20)

--This code sets the value

SET @firstname = 'Forta'
SET @lastname = 'Ben'
SET @age = 21
```

Result

EXAMPLE:

The following SQL code creates the three variables and assigns each a value using the SELECT statement and outputs their contents:

Write Code Syntax

```
--Part 3 - This code declares your variables
Declare @age Int;
Declare @firstname Varchar(20), @lastname Varchar(20)

--This code sets the value
SET @firstname = 'Forta'
SET @lastname = 'Ben'
SET @age = 21

--This code outputs contents
Print @firstname
Print @lastname
Print @age
```

Assigning Values to Variables Using Select

You can also use the SELECT to assign values to variables as well. The difference is that the SET statement only sets one variable value at a time whereas the SELECT statement allows you to set multiple variable values in a single statement.

EXAMPLE:

The following SQL code creates the three variables and assigns each a value using the SELECT statement:

Write Code Syntax

```
--This code declares your variables
Declare @age Int;
Declare @firstname Varchar(20), @lastname Varchar(20)

--This code sets the value
Select @firstname ='Forta', @lastname='Ben', @age =21
```

Result

EXAMPLE:

The following SQL code creates the three variables and assigns each a value using the PRINT statement and outputs their contents. The PRINT simply outputs text. This statement also concatenates the firstname and the lastname

Write Code Syntax

```
--This code declares your variables
Declare @age Int;
Declare @firstname Varchar(20), @lastname Varchar(20)

--This code sets the value
SET @firstname = 'Forta'
SET @lastname = 'Ben'
SET @age = 21

--This code outputs contents
PRINT @lastname+', '+@firstname
PRINT @age
```

Result

EXAMPLE:

The variable @age is data type integer. If you wanted to output this variable as "AGE: + @age" you would get an error because you cannot combine a string with an integer. You would need to convert @age into a string. To do this could use the CONVERT statement.

The following SQL code converts the variable @age to string and concatenates with the string "Age:'

Write Code Syntax

```
--This code declares your variables
Declare @age Int;
Declare @firstname Varchar(20), @lastname Varchar(20)

--This code sets the value
SET @firstname = 'Forta'
SET @lastname = 'Ben'
SET @age = 21

--This code outputs contents
PRINT 'Name:'+ @lastname+', '+@firstname
PRINT 'Age: '+ Convert(varchar,@age)
```

Result

EXAMPLE:

Now, let's concatenate all three variables:

Write Code Syntax

```
Declare @age Int;
Declare @firstname Varchar(20), @lastname Varchar(20)

--This code sets the value
SET @firstname = 'Forta'
SET @lastname = 'Ben'
SET @age = 21

--This code outputs contents
PRINT @firstname + @lastname + Convert(Varchar(20),@age)
```

Using Variable in Statements

Lets create two queries. One query will return customer information for a specific customer and the second query will return orders placed by that customer. This requires two select statements.

Notice that both select statements use the cust_id value in the WHERE clause. If you wanted to get the same information for another customer you would have to update the WHERE clause in two places. This is prone to human error. Imagine if you had to update the cust_id in lots of places. A better way of handling this is shown in the next example.

NOTE: This is also considered what is called BATCH-PROCESSING. Batch-Processing is a set of SQL Statements all submitted together to the Server.

EXAMPLE:

The following SQL code submits both statements together and returns the data we need in two separate select statements

Write Exercise Code:

```
--This query returns customer data
SELECT
	cust_name,
	cust_email
FROM
	customers
WHERE
	cust_id = 10001

--This query returns orders placed by customer
SELECT
	order_num,
	order_date
FROM
	orders
WHERE
	cust_id = 10001
ORDER BY
	order_date
```

EXAMPLE:

The following SQL code submits both statements together and returns the data we need in two separate select statements. However, in this select statement we use a declared variable for cust_id. This variable replaces the cust_id number as shown below(execute all at once):

Write Exercise Code:

```
--This code declares your variables
DECLARE @cust_id int;
SET @cust_id = 10001;

--This query returns customer data
SELECT
       cust_name,
       cust_email
FROM
       customers
WHERE
       cust_id = @cust_id

--This query returns orders placed by customer
SELECT
       order_num,
       order_date
FROM
       orders
WHERE
       cust_id = @cust_id
ORDER BY
       order_date
```

Using Conditional Processing

Condition processing is used in code to make decisions based on a condition. When the condition is met it performs an action. In conditional processing we use the IF statement.

Lets say you need to process orders by updating values. This process would be run on a regular bases and based on the day of the week would perform something different. The following example updates the variable @open to 0 or 1 based on whether or not it is Sunday.

EXAMPLE:

To get the current date we use the GetDate() function. To determine what day of the week it is we use the DatePart() function. This function returns the part of a day (the day, the day of the week, the month, etc). . The **weekday** (**dw**) datepart returns a number that corresponds to the day of the week, for example: Saturday = 6, Sunday = 7. The number produced by the **weekday** datepart depends on the value set by SET DATEFIRST. This sets the first day of the week.

The following SQL code returns the current date and the day of the week

Write Exercise Code:

```
--The GetDate() returns the current datetime
SELECT Getdate()
--This DatePart() returns the day of the week
SELECT DatePart(dw, getdate())
```

Result

EXAMPLE:

The following is a simple IF statement that sets a variable to either 0 or 1 based of whether or not today's date equals the DatePart() day of the week (Sunday).

Write Exercise Code:

```
--Declare variable
DECLARE @open bit

--Open for business today?
IF DatePart(dw, GetDate()) = 7
       SET @open = 0
ELSE
       SET @open =1

--Output contents of @open
SELECT @open as OpenForBusiness
```

Result

EXAMPLE:

Now you also want the process to take into account Saturday and Sunday.

The following is a simple IF statement that sets a variable to either 0 or 1 based of whether or not today's date equals the DatePart() day of the week (Saturday or Sunday).

An OR operator was added to take into account both Saturday and Sunday. In addition, we also added a new variable called @dow (day of the week) and populated with the correct value. We do this so we do not have to write code twice for both Saturday and Sunday. It is run before the IF statement.

Write Exercise Code:

```
--Declare variable
DECLARE @open bit
Declare @dow numeric

--Get  day of week
SET @dow = DatePart(dw, GetDate());

--Open for business today?
IF @dow = 6 or @dow = 7
	SET @open = 0
ELSE
	SET @open =1
--Output contents of @open
SELECT @open as OpenForBusiness
```

This statement without using the new @dow variable.

Write Exercise Code:

```
--Declare variable
DECLARE @open bit
DECLARE @dow INT

--Open for business today?
IF DatePart(dw, GetDate()) = 6 or DatePart(dw, GetDate()) = 7
	SET @open = 0
ELSE
	SET @open =1

--Output contents of @open
SELECT @open as OpenForBusiness
```

Result

Grouping Statements

On many occasions you will find that you need to execute multiple statements. What would happen if you executed the statement below? You would get an error. This is because the first SET will either be processed or ignored depending on conditions being met.

The second SET will always be executed. However, the error message is caused by the ELSE statement because it does not recognize it as being part of the initial IF statement. In other words you have to specify that the SET statements execute as a block.

We do this by using the BEGIN and END keywords.

EXAMPLE:

The following SQL code executes two statements at same time which generate an ***error.***

Write Exercise Code:

```
--Declare variable
DECLARE @open bit
DECLARE @dow INT, @process bit

--Get  day of week
SET @dow = DatePart(dw, GetDate());

--Open for business today?
IF @dow = 6 or @dow = 7
       SET @open = 0
       SET @process = 0
ELSE
       SET @open =1
       SET @process = 1

--Output contents of @open
SELECT @open as OpenForBusiness

--Output gets following error
Msg 156, Level 15, State 1, Line 12
Incorrect syntax near the keyword 'ELSE'.
```

EXAMPLE:

The following SQL code executes two statements at same time and executes the SET statements as a block of code using the BEGIN and END statements

Write Exercise Code:

```
--Declare variable
DECLARE @open bit
DECLARE @dow INT, @process bit

--Get  day of week
SET @dow = DatePart(dw, GetDate());

--Open for business today?
IF @dow = 6 or @dow = 7
	BEGIN
		SET @open = 0
		SET @process = 0
	END
ELSE
	BEGIN
		SET @open =1
		SET @process = 1
	END
--Output contents of @open
SELECT @open as OpenForBusiness
```

Using Looping

Looping is the ability to repeat blocks of code. Looping is executed using the WHILE statement.

EXAMPLE:

The following SQL code is a simple example of how to use looping. Here we define a variable named @counter and initialize the starting value at 1. The example executes the code incrementing the counter by 1 at every loop until it reaches the number 10

Write Exercise Code:

```
--Declare variable
Declare @counter INT
--Populate variable
SET @counter=1
--Execute while equal or less then 10
WHILE @counter <= 10
--Begin define block of code to excute
BEGIN
        --Output counter
       PRINT @counter
       -- Add 1 to counter every loop
       SET @counter= @counter+1
--End block of code to excute
END
```

Result