Lesson 13 Transcript: User-Defined Functions

Slide 1: Cover
Welcome to Lesson 13 of DB2 ON CAMPUS LECTURE SERIES. Today, we are going to talk about User-defined Functions. My name is Raul Chong, and I'm the DB2 on Campus Program Manager.

Slide 2: Agenda
This is the agenda for today.

Slide 3: Agenda- inline SQL PL
And we will start with inline SQL PL.

Slide 4: inline SQL PL
So what is inline SQL PL? Well in Lesson 12, we were talking about stored procedures and we were using the SQL PL language which is the SQL Procedural Language. Now, inline SQL PL is basically a subset of SQL PL and is used in dynamic SQL statement, triggers and UDFs. You can use inline SQL PL in a way to extend the SQL language, so let me give you an example.

Slide 5: Example: Randomly populating a table

Demo:
So, in this example, there are two prerequisites, the first one is you need to first create a sequence as is shown in this example, called, sequence name is myseq, so you will do create sequence myseq, you are creating this sequence with default values, then you are creating a table called t1, with three columns. Now here in this other square or rectangle, is where we have the inline SQL PL code, now inline SQL PL code, as you can see, is started with **begin atomic** and **end**. There is no, as you can see, it's just code that we put into; we can put into a script, or just write it directly from the Command Editor. There is no create procedure, or create function or create trigger statement, it's just code itself. And these, because it has **begin atomic**, it means that the whole compound statement is basically treated as one single statement, so that's what I was saying that you know in the sense it's like extended SQL language, because in SQL you have you know select, insert, delete, update. But if you want to do something else, you could do it using inline SQL PL, using some procedural logic and you can have you know **begin atomic**…**end** and you can treat all these blocks as just one single statement, and whatever you are doing within that will be treated as one statement.

Anyway, what we are doing for this particular example is we are going to be looping 20,000 times and we are going to be inserting into table t1, some random values, so we will be inserting 20,000 rows with random values. Ok, so let's execute this from the Command

Editor. But I have copied the same statements from, I put it in notepad, because it's easier to copy/paste. So, I'm connecting already to the SAMPLE database. And I'm going to paste the information here ready to be executed; now note that each statement is ending with an @ symbol. Ok, so I have to make sure that in the Command Editor, I have put the @ symbol as my terminal character, otherwise I'll have problems. Why did I change the terminator to @ symbol instead of the semicolon? It could be something else other than @, but as long as it basically has to be different than the semicolon. Ok, now why did I have to put this different terminal character is because within inline SQL PL, each line or each statement ends with a semicolon. So I need to use another terminal character to distinguish the end of the statement termination, the entire you know inline SQL PL statement termination versus the terminate, the ending of a given statement within inline SQL PL. This is also something that happens for stored procedures, the same thing for user-defined functions when you are working from the Command Editor.

Ok, so let's execute these three statements, and once execute this, let me go up here to see the result, and you can see that the sequence was, well first of all this is just listing the commands; and here we have that the sequence was created successfully, that the table was created successfully, and at the time the table was created successfully, there were obviously no rows, and then this inline SQL PL was executed which also successfully, let's clear the results here. So now let's take a look, or let's do a count, count table t1, right? So we do **select count (*) from t1** and execute. And we verify here that we have 20,000 rows, and let's do another one, which is, let's take a look at some rows from table t1, so we can select * from t1 fetch first 10 rows only, right? Let's clear the result here. And here we have the result. So I just put fetch first 10 rows only because I don't want to get the 20,000 rows back, so this is a way to limit the number of rows that I get back, right? But it just takes a look at what type of data was inserted into this table. And again, it's a very nice way to generate, let's say you are doing a test, and you need to insert 20,000 rows into a table rather than maybe creating a little program or a script, you can just create this very short inline SQL PL to generate these random information into the table. So that's one example of the use you can have for inline SQL PL.

Slide 6:Agenda (2) User-defined Functions-Overview
Moving on to the next section…

Slide 7:User-defined Functions
We have, we're going to talk about user-defined functions or UDFs, and this is just an overview first of all. So you know the difference between a stored procedure and a function is that a function always returns you a value. Now, there are built-in functions as point two for example, SUM, AVG, DIGITS etc. and you can create UDFs in different languages, in SQL PL, in C/C++, Java, etc. but we are focusing on SQL PL in the same way that we focused on SQL PL when we were using the stored procedures because it's very easy to learn SQL PL, or in this case will be inline SQL PL, which is a subset of SQL PL.

Slide 8: Types of Functions
There are different types of functions, a scalar function means that there will be only one value returned, and when you use a scalar function, you cannot change the data state as part of logic within the scalar function, for example, you cannot issue an insert, update, or delete statement as part of the function. There are a couple of functions that are listed here, COALESCE () and SUBSTR (), those are built in functions as well, they come with the product, but I will show you later on another scalar function. And then we have table functions which return a table. They don't return one single value, but they return an entire table, and because they return an entire table, when I call this function, they will have to be called in the FROM clause of a query; and the nice thing is that with these functions, I can change a database state, because I can issue insert, update or delete statement. And here we have some examples, and these are also built-in table functions, and we have also row functions and column functions which will not be covered in this e-learning series.

Slide 9: Agenda (3) Scalar Functions
So let's move on to the next section and talk about scalar functions.

Slide 10: Scalar Functions
So, this is an example of one scalar function where basically the name of the function is deptname-department name, and we are using as parameter the employee ID. Ok, then if we take a look at the logic: RETURNS VARCHAR (30); this says SPECIFIC unique name for the function, and then we have begin atomic…end, right? So this is like the inline SQL PL part, and from here we declare some variables, we set the values for some of these variables, and basically what we are doing here is you see we are setting v_department_name to a select statement, where we are joining, we are joining the department and employee tables in this part of the SAMPLE database, we are joining them based on the workdept, and deptno columns, and also we are saying that we want only for employee number = p_empid, and you can see here that the parameter name was p_empid. So basically we are using the parameter to limit the number of rows that we will get back after we join these two tables, and what we are trying to do here basically is pass me the employee ID of any employee. And then we return to you that the department name of that employee ID, right?

Now if you take a look at the department and employee table, the employee table has an employee ID, but it does not have the department name, what it has is the department number. And then in the department table, we have a department number and also a department name, so basically what we need to do is do the join based on the department number in order to get that department name. Alright, so that's the purpose of this function. And here we have the statement called signal which is used to raise an exception, so for example, in this case, if this department name is NULL, so you know I try to retrieve the department name, but this select statement returns NULL, because for some reason, you provide maybe an incorrect employee ID, and therefore the return for the department name is NULL, then I am going to raise exception, an exception as part of the function, so the function itself will return a value

with an incorrect error. And this error V_ERR is defined here where it has V_ERR , 'error: employee' etc. 'was no found'; so let's run that function. And there are two ways to create the function. The first way, as you can see from here, is to simply, and that's the same function, I just put it again in notepad, because it's just easier to copy and paste, is to simply, copy/paste the functions from the Command Editor as shown here. So let me do it from here first, and when I execute this, and I'm going to have to terminate with the @ symbol, and I have to make sure that the @ symbol is used as a terminator as well, and execute, right? So as you can see, the function was created successfully, let me clear the result here, let me clear the information here. And then I'm going to say, I'm going to call the function, so it's deptname. So basically, to call a function, you can use values like this, deptname, and then you can put an employee ID right? (value, deptname), and these are employee ID from the employee table. And I'm going to execute, and then I get these results back, ok? Let me open the Control Center, so that you can see the tables that I am using in this example, now you may be wondering why I didn't put an @ symbol, the reason for that is when you just execute one statement, there is no need to put a terminator, if you have several statements like another select…. Then you do need to put terminator on each parameter, on each statement.

Ok, so let's take a look at the Control Center, and from here, let's look at the SAMPLE database, we are going to look at the employee table and department table. So, employee table, if I double click, I get some sample information, and this is the employee ID, right? So this is the employee ID that was using, and, let me just enlarge this (window). So here is the work department number, but I don't have the department name, so what I am going to do is I am going to take this work department number, then I will be going to the department table, and what the parameter is doing, it will find the department number, which happens to be the first one, and it will return the department name, which in this case is spiffy computer services, OK, so that's the reason why we got the spiffy computer services.

So I showed you one way to invoke the function which was using, deptname, sorry, which was using values, but you can also invoke this by issuing a select statement, so you can say **select deptname ('000010') from**…. And here you can put any table, so for example, I can say: **from employee**. Now the problem of this.., ok, let's execute this to see the problem with this is that I get the same value returned, in this case 42 times, because there are 42 records in the employee table, So the function is going to be executed per row in the table, right, so if you don't want this to happen, you can create a dummy table yourself with only one row and one column, or you can use the table sysibm.sysdummy1@ which is a table from the system. But it was basically created, so that it's basically a table with one row and one column, and was created for this purpose, where you know you just need to get one return back within the function, so if I execute this now, I just get one record back. Ok, so and you can use these functions within your stored procedures, so you have the stored procedures, you can invoke the function, and or you can call them directly from your applications within a SQL statement or using the value statement as I showed you before. Ok, so this is the first way to create a function which is by using the Command Editor.

The second way similar to stored procedure is to create a function from IBM Data Studio, so I am showing you here I am in IBM Data Studio, and I am showing the same project that we use for stored procedures, but this time we are going to use user-defined function. Now IBM Data Studio again is just a tool, that it's very useful, because maybe you can do some debugging using this tool, you can list all your applications from this tool, but it's not really required to create a stored procedure or user-defined functions, as I show you in this lesson and the previous one, because you can create them from the Command Editor. However, for administration purposes, for debugging purposes, you may be better to use IBM Data Studio.

Ok, let me show you how you create a function using IBM Data Studio. You first select the folder (user-defined function) within your project, and your project I showed you how you create that from the previous Lesson 12, for stored procedure, and then I am going to right-click, choose New > User-Defined Function, then I am going to associate that function with this DB2 on Campus program so there is no problem for that. For the name, I'm just going to, you can put any names, so let's say my func, language, I'm going to use SQL, and then I'm going to say next. Ok, now I have to connect to the database that is associated to this project. So, in this case, it will be, for your case, it will be your own password, and then I say

Ok. Now this tool, IBM Data Studio, is providing me with some templates or some sample SQL that I can use. I'm only to use this, I could write my own, but just to save time, I'm just going to use this SQL, and you can continue with this wizard, which will ask you for more questions, but I normally, just click Finish. And I will get a template as you can see, so this is the select statement that was provided to me as part of the tool, and I could use something else, but I'm just to save time, just assume that I wrote these codes, so what we are doing here is we just create a function with this name: RETURNS INTEGER. And basically at the end of the logic, is to return the **SELECT count * FROM SYSCAT. FUNCTIONS;** so how many rows there are in this table, syscat.functions. So if I want to execute this, if I made a change, I will have to right-click and choose save, but in this case, it's grayed out because I didn't make any change.

But now I want compile these functions, so what I first have to do is to basically right-click on the name of the function and choose Deploy. Deploy is basically…, here they allow you to change the schema, like for example, in here, inside the procedure or the function, if it did not provide a schema to your tables or your objects, then this will be the schema that will be used. Anyway, so I normally click Finish, and as I was saying, deploy is basically like compiling the function, or basically what it means is you are actually executing this create function statement into the database and the database will store the function in the database, after it was executed, right? So you can think of create function and so on like creating a table, right? You create a table, then the object is created in the database, the same thing for the function, if there is a function, the database is created in the function. But when you are working with IBM Data Studio, if you don't deploy, then this function will not be on the database, because it will just be on this IBM Data Studio tool. So it's like, assume you are working on any

editor like maybe notepad, you know you can save things in Notepad. But if you don't execute the create function statement, it will just be in Notepad; once you execute the create function statement, then you are actually invoking DB2, you are going to DB2 and DB2 will store internally this function. Ok, so as you can see from the bottom right corner, the function was deployed successfully, and then what we are going to do is we're going to run the functions. So you go to the left tree, left side of the panel, and to the tree, right-click, or choose myfunc and right-click and choose run, and when you choose run, here at the bottom right corner, you can see that you get the result back.

Ok, so that's how you can create a function from IBM Data Studio, you can deploy it or compile it, so that it's in the database, so you can run it by right-clicking on the function name and choosing Run. And this same function, for example is called myfunc, So I could invoke it from the Command Editor, so I can say, and I can use values myfunc (), and in this case, it doesn't take any parameters, and as you can see, it returns the same output, the same thing here, I can connect to the SAMPLE database from here, and then I can say values myfunc () and execute, pressing enter, and then I get the same result.

Ok, so that was a way to create a function from the Command Editor, and then from the Control Center, you cannot create functions, but you could view the application objects, so if you choose the application objects, you can say user-defined functions, and you should be able to find these myfunc functions that we just created. It should be under… well, sometimes, if it doesn't show right away….. you may have to use a.., as in this case, it looks like, Aha, there is myfunc, right? But if for any reasons, you cannot see it, you may have to do a View > Refresh, and then it starts going back to try and look for it. So that's sometimes you need to do that when you don't see an object that you had just created.

Slide 11: Invoking Scalar UDFs:
Ok, let's move on. This I already talked about, this is how to invoke a scalar function, and one way was to use select, invoking the function and using the SYSIBM or SYSDUMMY1 or any table, but it will be repeated according to the number of rows in that table, or you can use values statement as with this.

Slide 12: Agenda (4) Table Functions
Alright, so let's move on to the next section of this presentation which is table functions.

Slide 13: Table UDFs
So a table function as the name says, it will return a table. It's not going to return just a single value but an entire table. Therefore when you invoke this function, you have to invoke it from the FROM clause of a query, so if you are doing a select, you don't put it as a column as you are doing for the scalar function, but you put it on the FROM clause. And from here you can perform insert, update, and delete operation as part of the logic of the function.

Slide 14: Table Function Example
Ok, so, let's take a look at this example. In this example, we have a function called getEnumEmployee and we are passing a department number, and what will return is what is shown in red- returning a table with 3 columns. Now if you take a look at the code, it is basically doing a select statement of these same three columns, but we are limiting the number of rows that we are returning, based on this parameter P_dept which is what is passed here as a parameter, so we are passing a parameter p_dept and we are using it to filter or narrow the rows that will be returned. Alright, so again, I have here in notepad, copy-paste that function, because it is just for me to save some time, and easier to copy, paste. Let's go back to the Command Editor, let's clear the results here, and we need to put the @ symbol at the end, make sure that is put in there and make sure that you have the @ symbol as well here, and then we are going to execute the function or the creation of the function, so the function was created successfully. Now what we need to do is we need to execute or we want to invoke this function that we just created. So to do that, we can use syntax like this, so let me delete all of this.

Slide 15: Calling a Table Function
So in this syntax, which is also shown here in this next Slide, you invoke again a table function after the FROM clause, so here is the FROM clause, and after the FROM clause, we are invoking the getEnumEmployee, but we need to add this special function called Table, and we need also to add an alias. So basically if you look at this, this is like saying **select * FROM T** but T is basically an alias to a Table which is based on the function getEnumEmployee, which we just created, so let's invoke that function. So, let's run this and see what we get, and in this case what we get is one record which is the one that you see here. So for this employee, sorry, for this department number, we are getting this information for the employee, ok? That was the purpose of this function, and that's how you can invoke it.

Slide 16:QuickLab #13-Creating a Scalar UDF Using IBM Data Studio
Ok, now I will suggest you to pause this presentation, and take a look at Quicklab #13 and we are creating a scalar UDF function using the IBM Data Studio or you can also try using the Command Editor.

Slide 17: Recommended Book
This is a book that is recommended if you want to know more about SQL PL, Stored Procedure and User-defined Functions as well as Triggers and as well as inline SQL PL.

Slide 18: What's Next
And with this, we have finished this lesson, so congratulations for completing Lesson13, User-defined Functions, and as to what is next, it will be Lesson 14, Triggers, Thank you very much.