Lesson 14 Transcript: Triggers

Slide 1: Cover
Welcome to Lesson 14 of DB2 on Campus Lecture Series. Today, we are going to talk about Triggers. 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(1) Overview
And we will start with a short overview about Triggers.

Slide 4: Triggers
Triggers are database objects that you define on a table and they will be fired or they will be triggered when you perform an operation like INSERT, UPDATE or DELETE on the table where you define the trigger. The operations or the statements that will be executed as part of the trigger are named or are called triggering SQL statement, so in there you can put, for example, another statement like an INSERT on a different table or you can perform some validation as part of the logic of the trigger. We will provide some examples as part of this series, or as part of this lesson.

Slide 5: Types of Triggers
So there are three types of triggers, we have a **before**, **after** and **instead of** trigger.

- Before trigger means, let's say, you are doing an INSERT, then, before the INSERT actually finishes, then the trigger will be doing something else, and then once the trigger finishes whatever it has to do (whatever it was doing), the INSERT would be allowed to continue with its process, which was inserting something.
- After trigger means after the operation finishes then the trigger will be fired. So for example, if you are doing an UPDATE, then after the UPDATE completes, the trigger will be fired.
- And then we have an instead of trigger, which are triggers that you will define on a view. Normally, you are doing, let's say an UPDATE operation on a view. Views are read-only, so you cannot really update a view. So when you perform an UPDATE operation on a view, you really have to be updating the base tables that define that view. Through an instead of trigger, you can do these update of the base tables.

Slide 6: Agenda(2)Before Trigger Example
Moving on to the next section, we have an example of a before trigger.

Slide 7: A Simple BEFORE Triggers

And here is the trigger that we will be illustrating, so we have, this is the syntax of the trigger, and this is the name of the trigger, in this case, it is called default_class_end. And then we have **No Cascade** that is just part of the syntax. And here we have before INSERT, so that is basically, what it is telling me is that this is a before trigger, and it is before an INSERT operation; and then we have ON cl_sched so this is the table where I am defining the trigger on. Then we have REFERENCING NEW AS *n* – this is used for correlation and I will explain this in more detail in the next few slides, especially when I talk about after triggers, but you can use this as a prefix on the code. And then we have FOR EACH ROW.
FOR EACH ROW means that the trigger will apply for each row on the table, and MODE DB2 SQL, it is a fixed mode, so… this is the only mode that is supported. And then we have WHEN,… this is a part of the logic or the triggering statement. So here I am saying when (n.ending is NULL, set n.ending=n.starting+1hour), n.ending, let me show you this with a diagram here.

What is happening is that the cl_sched table, which by the way is part of the SAMPLE database, has these four columns: class_code, day, starting, and ending. Now, let's say if I am doing an operation like this, where I am trying to insert into this table and I am putting class_code, day and starting with these values, but note here that I'm not providing ending. So I am trying to insert a record into this table, and then I am not providing the value of ending. So, let's go back to the logic of the trigger, it says here when (n.ending is NULL, set n.ending=n.starting+1hour), so what does this mean? In this case, what is the value of ending? Since we are not providing that value, it means that ending has a value of null, so this trigger logic would apply, and therefore what will be inserted here in this row will be the following: will be, OK, first of all it is abc, right? As the class code, so I'll put here abc, for the day we'll put 1 as is shown here, and then starting it says that it will insert 10 and for ending it will be inserting 11, because based on the logic of the trigger, that will be the value of ending as you can see again from this slide when (n.ending is null, set n.ending=n.starting+1hour). So what we are seeing here is that, you can see basically that this trigger, this before trigger is being used for validation, where validating what type of data we could input into the table. So we have, we were just inserting 10 and now ending will be, we have a value null of 11.

Slide 8: Quicklab BEFORE Trigger Using SQL PL

Alright, now, moving on to the next slide, we have another trigger with the value of validate_sched, and actually if you take a look at the logic of this trigger, the first three lines are very similar in terms of the logic as the previous trigger. So, we can create this trigger and do some tests, and then… also, something interesting about this trigger is that we are using the SIGNAL statement. The SIGNAL statement is basically used to raise an exception yourself. So, again, these before triggers are used for validation, and sometimes you know when you are inserting something that is not supposed to be inserted, then you want to raise an exception. You want to raise an error, so you can do that with the SIGNAL statement, which in this example it is saying if (n.ending>'21:00') so it means if n.ending is greater than

9 p.m., then SIGNAL an error with SQLSTATE of '80 000' , and the message of that error will be class ending time is beyond 9 p.m. So, let's create this trigger, but before we create the trigger, let me first open the Control Center here, and … I have here the trigger as well.

So let me just copy-paste this trigger, while the Control Center is starting… copy/paste, and then from the Control Center, we are looking at the SAMPLE database, and right now if we take a look here, we have, we can look at the triggers, and well right now we have not created any trigger, I just explained to you the previous trigger, but we never created it. So, now what we are going to do is we are going to create a trigger. We can create triggers from different places. Ok, let me show you first how to create a trigger from the Command Editor which I just invoked by clicking on that fourth button from the Control Center. Around here I am going to first ADD a SAMPLE database. Basically, I am connecting to the SAMPLE database. The trigger is a database object that resides only in a database. So first, I need to connect to the database, and then I want to create a trigger in this database. Ok, I want to clear the result here just to keep it clean, … and I just copy-paste trigger that I mentioned before, this one validate_sched, and then I am going to run this, but before I run this, I have to make sure that I put a right terminator, in this case it is the @ symbol. And I explain why I need to change this terminator in the lesson where I talk about tools. So now when I execute this command, and as you can see, this trigger was created successfully, so that is great and let me clear the results here. And I just clear this information as well… so now the trigger has been created. So let's run some queries to test if the trigger is working as it is supposed to work. So let's, if I run this, if I paste this INSERT statement, what is going to happen here: in this case you know that I am not providing an ending column. So if I execute this INSERT statement, I get first of all, that the operation was successful; so that is good.

Now I am going to perform a SELECT statement, so let's say **select * from cl_sched**, and when I click on the Execute button, I get this value: def 2 18:00:00 and 19:00:00. Note here, what has happened is I was inserting the value of 18:00, 6 p.m., and for ending I get 19:00 or 7 p.m., right? If we look again at the SQL that I just executed, I was executing or inserting only 18:00 for starting, and I didn't provide any value for ending, however, as you can see from the output, now ending has a value of 19:00, and that is because of the trigger. Ok let's try something else. And here I have another statement where I am providing a value for ending, but I am not providing a value for the day. So let's use NULL, well basically I am providing NULL, for the day. So let's see what happens when I do that, so I am going to highlight this column, this statement, and I am going to execute it, and now I get an error. It looks like an error, but actually when you get an SQL code of -438 +438, that means it is an application-defined error. So the trigger itself is raising this error. Why is it raising this error? Let's see what happened here; so, we were executing the INSERT statement, and we put NULL for the day. So that really does not matter, because what matters here is that we put 22:00 or 10 p.m. for ending. And the error that I am getting is: I got an application raised error with diagnostic text: 'class ending time is beyond 9 p.m.'.

OK, so as you can see ending was set to 10 p.m., and that is why the trigger is giving me this error message with this value. OK, and again it repeats here the error message Application raised error with diagnostic text: 'class ending time is beyond 9 p.m.'. I am going back to the trigger, you can see, that this is what I have here, class ending time is beyond 9 p.m. So that is something that I want that trigger to do. Let's do one more test, which is to run this other INSERT statement. And in this case now what we have is that the last, the value for the day is seven, and looking at the code here we said that if the day is 7, then raise this other condition: class cannot be scheduled on a weekend. So let's run that INSERT statement. Let's clear the result here, let's clear all of these, and let's just execute this statement. And now again I get another error, but again this is an application-defined error. And as you can see when I move up here, it says class cannot be scheduled on a weekend, so that is exactly what we want the trigger to do—class cannot be scheduled on a weekend! So that is an example of the before trigger, and again, before triggers are very good to use for validation as we show in the example.

Slide 9: Agenda(3) After Trigger Example
Ok, so let's move on to the last section of this presentation which is an after trigger example.

Slide 10: A Simple After Trigger
So what we have here in this after trigger example, is a trigger called audit_emp_sal. And before working on this trigger, you have to create this table called audit with these columns. Now again, so we have create trigger audit_emp_sal, that is the name of the trigger. And here we have, what we say…is an after trigger, and it is an after and UPDATE operation. Now it is after the UPDATE on the salary column on the employee table. So the employee table is, as well, part of the SAMPLE database, and there is a column in that table that has a salary column, so let me just quickly show you. Let's use this other command window just… to make you practice with different tools, so from here I am going to connect to the SAMPLE database, and then here I am going to do **select * from employee**, ok? And as you can see if I move up here, well there are many columns, the EMPNO (employee number), FIRSTNAME (first name), etc. Here is the SALARY column. So if we look again at the trigger, it says: "after UPDATE on the salary column on employee, do something".

Now we have here REFERENCING OLD AS o NEW AS n, so this is what I was mentioning before in terms of correlation. And here I am going to explain in more detail. This is used for basically… prefixes in your logic, so OLD AS 'o' what that means is the old value, so in this case we are doing an UPDATE, right? So that means whatever was already on the table that you want to update refer to it in the code, in the triggering statement as 'o', and then the value that you put in the UPDATE statement, refer to that as 'n' or as a new value. So the new value refers to that as 'n'. This will become clearer as I continue with this example. Then we have for each row. It means for each row on the table； and then mode DB2SQL that is the only mode that is supported right now. And then what we have here is INSERT INTO audit (table) VALUES and then we have different information here. So what we are doing here

basically is the following, let's say, I just join a company, and I'm a junior DBA and my senior DBA who was my boss, has granted me authorization to many of the tables. And one of the tables that he wants me to access is the employee table.

Now, the employee table has the salary of all the employees in the company, and it happens to have my salary as well. So I say well, let me, I am not happy with my own salary let's say, so let me raise my own salary, and I say ok, let me do an UPDATE statement on this employee table, and see what happens. So maybe I do something like this: I can say for example db2 UPDATE salary …sorry, UPDATE employee because the name of the table is employee, set salary=, let's say, 100,000 or 1 million dollars, where, well let's pick, let's pick one of these rows… let's say where the first row. The first row says that the first name is or the last name is HAAS, right? So, where first or last name is HAAS, so we say where, last name =, oh, sorry, I pressed the wrong button. It equals to HAAS. I press Enter and the command is executed correctly. I can do, again, a **select * from employee** to double check.

And here you can see now for HAAS the salary has been changed to 1 million dollars. OK, so let's assume, oh, it says Christine HAAS, anyway, let's assume I'm Christine HAAS, and I changed the salary to one million dollars. Now this is before the trigger was created, but if the trigger had been created, what would happen? Well, let's create the trigger right now, see what will happen. And again, I put this information in Notepad because it is just easier to copy/paste. And before we can create this trigger, we need to create this audit table, because I invoke that… or I use that audit table, as part of this after trigger. Now, I am going to work on this Command Editor again, just so you see different tools. And Ok I can execute this, if it is just one statement, that I don't need to put a terminator, but let me just put a terminator, and I am going to execute this. And the table is created successfully, so that is perfect; clear the result. Now I'm going to create the trigger, so let me clear this out from here, and let me create the trigger that is here, so let me just copy/paste. Create trigger etc. and I'm going to create a trigger. Here I do need to put the @ symbol, and I execute, and now the trigger has been successfully created, so that is great as well. And now let's test this trigger.

Well the previous example, I was able to insert 1 million rows, sorry, $1 million as my salary for Christine, but, now I have created the trigger, let's see what happens. So, let's say now I want to, oh, actually, let's continue with this example. Let's see the salary for employee with the employee number of 10. So if I execute this, it happens to be the same record that I picked before, it shows that the salary is $1 million. Now if I change this and I perform an UPDATE operation, on this same record, right? That will be updated. And let's say, so from 1 million, I am going to put 9 million, 9 999 999; so 9 million, 9 hundred and ninety-nine thousand, nine hundred, ninety-nine. So let's execute this UPDATE statement. Let me clear the result here, and I perform the UPDATE and the UPDATE is as well,… successful. So what did the trigger do? Well, the trigger was writing into a different table called audit, right? So let's take a look, first let's take a look at the UPDATE (that) was performed. So if I go here, clear the results, and I **select * from employee**, now we see that the new salary has

been increased, so that is working fine. But now, what did the trigger do? Well, the trigger was actually inserting something into the audit table. What did it insert? Let's do a **select * from audit** to see what it inserted. And when I issue this **select * from audit**… this is what the trigger inserted: it says the time stamp, and then the employee, with this employee number 000010; salary changed from 1 million to 9 999 999 … by the user ID who did it. So if I was again going back to the example when a new employee and a new Database Administrator in this company, and I try to do something that is not honest, and I try to raise my own salary, I will be caught by, let's say my senior or my boss, because he created a trigger, which is writing into this audit table. So this is how a trigger can be used in this case, as an after trigger, and basically for auditing purposes in this example.

By the way, if you look again at the code here, you can see that you have, we have used o.empno, so that will be the old value of employee number (that means the value was on the table), o.salary means the old value or the old salary, and n.salary will be the new salary. And in the example that I was showing, this will be one million, and this will be 9 999 999. So that shows you when and how you can use this correlation prefixes. Alright, so right now I would suggest you maybe… pause and work on how to create a trigger using the Control Center.

Now, let me quickly show you how you can create a trigger from the Control Center. You can go to whichever database you are using in this case, the SAMPLE database. You go to trigger, and right-click, and choose create, and from here, you will say OK, let's say it is a.., you can put any name for the trigger, so you can say default-class-end, for example, for the name of the trigger. For the schema for your table in my case, it is RFCHONG, and in your case, may be db2admin. And then you can choose the table or view, it will be, let's say CL_SCHED. Here it says before trigger, and it is a before INSERT. And then you can click on this tab for triggered action; correlation name for the new rows, you can say 'n', and if it is old, you will put 'o', for example (it is just a letter); here in this case for each row, that is the default, or the only one allowed in this case for each row. And then here is where you can put your logic, so that will start from the WHEN statement. And so let's say I go back, and what I am going to use is the, this statement here, so this WHEN, that is what will be put into the Control Center. Ok, so I am not going to write the entire statement, but you will put here when (n.ending = null….) etc. And if I click on Show SQL, you can see the statement that is being built, so you can compare this to the actual text. So in this case, here is the part that I put when (n.ending = null….) Ok, so let me just cancel from here, let's go back to the presentation.

Slide 11: Quicklab #11-Creating a Trigger Using a Control Center

And again you should be working on this trigger which is part of Quicklab #11, so you can pause right now and work on that lab.

Slide 12: Final Notes

These are some final notes. If I go back, what… it says here, another benefit of dynamic compound SQL is that inline SQL PL is supported. If you go back to this slide here, note here that we are using: begin atomic. So that means we are creating the whole thing as one unit, because normally you can only write or put one single statement after the trigger. But if you use **begin atomic**, then you can treat the whole thing as one unit. And this is basically dynamic compound SQL; it basically has, you have a compound SQL statement because you are executing several statements within this 'begin atomic'. So it is good to use 'begin atomic' within a trigger if you want to execute maybe more logic. There is more information about this in these different articles from the DB2 developerWorks.

Slide 13: What's Next?

OK so, with this, we have finished this lesson, so congratulations for completing Lesson 14. And as to what is next, it will be Lesson 15, which is PureXML. So I recommend you to take a look at that, Lesson 15. Thank you and have a good day.