

Lesson 11 Transcript: Concurrency and locking

Slide 1: Cover

Welcome to Lesson 11 of the DB2 on Campus Lecture Series. We are going to talk today about concurrency and locking. My name is Raul Chong and I'm the DB2 On Campus Program Manager. This is the first lesson within the Application Development part of this course.

Slide 2: Agenda

This is the agenda for today.

Slide 3: Concurrency & locking overview

We will start with a concurrency and locking overview.

Slide 4: What is a transaction?

There are some basic concepts that we will talk about in the next few slides. The first one is: what is a transaction? If these topics are too basic for you, you can always move forward to the next few slides. A transaction could be better explained using this simple example: let's say, you have two accounts at a bank; one is a savings account, and the other one is a checking account. In the savings account, you have a balance of one thousand dollars. In the checking account, you have a balance of two hundred dollars. Now, let's say you want to transfer one hundred dollars from savings to checking. Behind the scenes (probably) what is going to happen is you will have to debit one hundred dollars from the savings account and then you have to credit one hundred dollars to the checking account. Now what is going to be the problem here? What happens if you first issue a debit of the one hundred from the savings account and all of a sudden there is a power outage and the operation is not completed. It is not able to complete. So you basically took one hundred from the savings account, but you didn't put the other one hundred to the checking account. So you just lost, basically, one hundred dollars. Now, in a bank or many operations in many industries this type of situation could not happen. I mean, you basically have to create these as one unit. So this is a transaction, basically treating several operations or several, maybe, SQL statements, as one transaction; as one unit of work. So a transaction and a unit of work is an equivalent name.

Slide 5: Concurrency overview

Now this is a brief introduction to concurrency: there are several applications; application A, B, C and D. Applications A, B, C, and D try to access the same row in the same table. Now if there is no control, then you may have the application A doing some sort of... update or maybe all of these applications are doing an update on this same row. So at the end what will be the final result? It would be hard to tell what will be the final result if there is no control. But, concurrency, in general means that several applications can work at the same time on the same database, or same tables on the database.

Slide 6: Locking overview

Now in terms of control for allowing a consistent way to do concurrency, or a way that is integrating your data, you can use locks. For example, in this case, you can have the same applications, but this time application B is the one who accesses, let's say, the 2nd row first. So this application A took a lock on that row, so that is when a lock can help to allow for concurrency, but at the same time allow for integrity of your data. So now that application B is holding the lock, the other applications basically have to wait until this lock is released.

Slide 7: Concurrency

So that is just a very brief introduction or concept on concurrency and locking. Now we are going to talk about concurrency in a little bit more detail.

Slide 8: Concurrency

So, concurrency is designed for allowing users to access a database simultaneously, so that several applications can work on the same database at the same time. And there are several problems that can happen if there is no concurrency control. The four problems that can happen are: lost update, uncommitted read, non-repeatable read, and phantom read.

Slide 9: Lost Update

Let us explain each of these problems very quickly. So 'lost update'... what it means is... let's use this example... let's say you have a table called Reservations and in this table you have three columns; flight, seat, and passenger name. Now one application is connected to this database and tries to access this table, and issues an UPDATE statement where it is going to change the passenger name to 'instruct' where the passenger name is 'null'. The seat is seven-C, so basically it is trying to change the first row here and trying to put here an 'instruct'. Now the other application at the same time is trying to do the same, but now it is trying to change the passenger name to 'manager' on this side. So what is going to happen if the first application was the one who performed the update first (and this was saying 'instruct') and then the second application comes next (and that is updated), and the final result ... would say 'manager'. So basically we just lost an update from the first application.

Slide 10: Uncommitted Read

Then moving on to the next type of problem, we have an 'uncommitted read'. So what that means is... using the same example, I am doing an update, so that I want to update the first row so that now the passenger name is 'instruct', but then comes the other application that issues a SELECT statement where P-name is NULL. So given that the first one now is not NULL because it is saying 'instruct', then it will only pick the second row which is the one that has a seat of seven-B. But then what is going to happen is that the first application is going to rollback, so the change it made for the first row, it is going to roll it back, so it is going to, or it is not bothered basically (by) what it is doing, so it is going to rollback and what is going to happen is that the second application basically has incorrect results because it just has one row when it should have two rows. So this is called uncommitted read.

Slide 11: Non-repeatable Read

Then we move on to another problem which is 'non-repeatable read'. So in this example we have the same reservation table, or... sorry, this is another table, but it is related to the same theme on reservations of flights and in this case, let's say... I obtain a cursor. So I basically do a SELECT statement and get several rows, now what is going to happen if one application issues a SELECT statement and gets, let's say, all of these rows you see on this screen. Then another application comes in and performs an update or some sort of operation that will remove one of these rows from the cursor. So that means when the first application issues the same cursor again, the same SELECT statement again, he will see that there are less rows available, or returned. So basically that is what is called a non-repeatable read. You cannot repeat the same result for the same read operation; for the same SELECT operation, or for the same cursor.

Slide 12: Phantom Read

The next problem is called ‘phantom read’, which is similar to non-repeatable read. But instead of reading less rows, you are going to read more rows. So for example, here you have the reservations table. Right now it says that application A is doing a SELECT statement where P-name is NULL. So it is going to get the first row. Then comes ... the second application which performs an update, so that the 2nd row where the P-Name is Susan Liu, now is going to be NULL. This 2nd row is going to be NULL and then when the 1st application issues the same SELECT again or puts the same cursor again, he or she will get more rows, she will get 2 rows now; the 1st one and the 2nd one, because now there are two, now both rows will have a P-Name of NULL. So this is called a phantom read because in the first case when I open the cursor, I only got one row, now, the second case, I get two rows.

Slide 13: Locking

Ok so now I am moving onto the third section of this presentation, we are going to talk a little bit about locking in more detail.

Slide 14: Locking

So locking allows for maintaining data integrity while it allows for concurrency. And locks are taken automatically, so as you are performing operations, updates, inserts, deletes, then DB2 will take care of handling locks. There are two types of locks. There are locks for an entire table and locks, also, for rows. And also, there are other qualification(s) in terms of types of locks, which is to have shared locks and exclusive locks. Shared locks are locks that are taken when you are performing maybe a read operation. We just call it an S-lock. And exclusive lock is (maybe) when you are, when application is doing an update, insert or delete; so that would be an exclusive lock, so that means that particular application can do things exclusively. No other application can work on the same row at the same time.

Slide 15: Isolation Levels

Now within locks, we have what is called... isolation levels. You can think of isolation levels as policies on how you want DB2 to work with locks. So there are different isolation levels and ... the names are similar to the problems that we were seeing before. So we have uncommitted read or UR, also known as dirty read, Cursor Stability or CS, Read Stability or RS, Repeatable Read or RR. You can set these isolation levels at different levels. You can set at the application level or session level, where the default is Cursor Stability. You can set at Connection level; you can set at Statement level. If you are using embedded SQL, you can set it at bind time. For dynamic SQL, you can set it at run time.

Slide 16: Isolation Levels -Uncommitted Read

So let's go through these isolation levels one-by-one. And you can pause this presentation and look at the information here and you can see what type of situations can happen and situations that could be prevented as well.

Slide 17: Isolation Levels –Cursor Stability

So if you move on to this next slide, you will see the same thing for Cursor Stability, you can again pause; you can read more about this.

Slide 18: Isolation Levels – Read Stability

Or the next slide, Read Stability. You can pause again and read more about this.

Slide 19: Isolation Levels – Repeatable Read

The same for repeatable read...

Slide 20: Comparing isolation levels

But now, let's move on to this next slide where I will explain these same isolation levels using this chart. So what we have here ... let's say you have a table on the left side. And ... these are the rows of the table. And what you are going to do now is you are going to do a fetch operation. Now let's assume your application is using UR. So it is using uncommitted read. So that means that as you are fetching (as you are going down the rows), no row locks will be taken. So this is when you have an application with isolation level of UR. If you use isolation level of CS, what is going to happen is as you are going down and reading from this table, you are going to take locks, so for example, here I am reading the first row, so I am going to take a lock on the first row, then I go down to the next row and what is going to happen is I am going to release the lock on the first row and then I am going to take a lock on the second row. As I move down to the third row, I am going to release the lock on the 2nd row and take a lock on the 3rd row, and so on, and so on.

Now with respect to RS or RR, what is going to happen is when you access the 1st row, I am going to take a lock, when I move down and I fetch or access the second row, I am going to take a lock on the second row, but I am still going to keep the lock on the first row. When I move down to the third row, the same thing, I am going to take a lock on the 3rd row and I am still going to keep the locks for the 2nd row and the 1st row and so on and so on. So as you can see, the 1st isolation level which is UR allows for maximum concurrency. While, RR or RS allows for the least concurrency, but at the same time, UR provides probably the least accuracy in terms of results while RS or RR provides the most accuracy, in terms of the result. So that is a quick overview of how UR, CS and RR and RS work in terms of locking.

Slide 21: Comparing isolation levels Terminology

Now you can set (as mentioned before), the isolation level at the application. So this is an example where if you are using JDBC, you will set a property in the JDBC application for the different isolation levels. So if you set the property in JDBC for TRANSACTION_READ_UNCOMMITTED, then when it reaches DB2, it will be translated to Uncommitted Read. When you set TRANSACTION_READ_COMMITTED, it will be converted to Cursor Stability. And so on, and so on. Something interesting here is that for JDBC when you use TRANSACTION_REPEATABLE_READ, that doesn't translate to Repeatable Read in DB2, but to Read Stability. And when you use TRANSACTION_SERIALIZABLE then that one translates to Repeatable Read in DB2. If you have a .NET application (similar), you will get, depending on the property that you set in .NET, you would translate to different isolation levels in DB2.

Slide 22: Statement level isolation

Now you can also set isolation at the statement level. And that is done simply by adding the term 'with' and then the isolation level that you want to use. So for example, here you say, 'with UR' and then that would apply to this statement. So what is going to happen is let's say ... you have, an application that is using the cursor stability in general, but for a specific statement you can use this 'with UR', so ... for that particular statement, you just use 'with UR'.

Slide 23: Lock Escalation

And we will show you later a scenario of how that works. Next section in this presentation is lock escalation.

Slide 24: Lock Escalation

So how does this work? Well, let's say you have many applications that are accessing the given table and they will be taking locks on the different rows of the table. Now what is going to happen is, each time you take a lock on a row, you will be using some amount of memory for locks; so there is an area in memory called LOCKLIST which is basically the memory for locks. And, as you take more and more row locks, you will be using up that space. There will be a point of time that you run out of space in that area and then what DB2 will do, it will perform this escalation of the locks, so for many, many small locks, it will escalate to one single table lock. So that will not be good ... because, let's say, you have one million rows and you were doing locking on the rows. Now, let's say, you have five hundred thousand row locks, then at that time, you have one million rows and you are locking five hundred thousand rows, then that is not good, but at the same time, you have the other five hundred thousand rows still available for use ... or other applications can use it. But if you take a table lock, then the entire table, that means the one million rows are locked. So lock escalation, in general, is not good.

Slide 25: Lock Escalation

And you can manipulate or configure escalation from not happening by (as mentioned before), taking a look at the LOCKLIST and the other one is the MAXLOCK. These are the parameters from that dbcfg; database configuration file. And you can basically change or increase the LOCKLIST; increase the memory. You can also change the MAXLOCK which is the maximum percentage of the entire lock list for one single application. And you can change the value of this parameter so that the lock escalation doesn't happen. The other way you can do is you can change your applications, so maybe you should issue more COMMITs and that way you will be releasing the locks as well. But lock escalation is not something good.

Slide 26: Lock Escalation (continued)

And you can detect whether there is a lock escalation by looking at that diagnostic file; the db2diag.log, which we'll cover in more detail on the Troubleshooting section.

Slide 27: Lock Escalation indication in the db2diag.log

But if we look at the db2diag.log, you could see these sqlEscalateLocks, so that tells you there was an escalation. And this is the statement that caused the escalation and this is the table where that problem happened. So using the DB2diag.log, you can determine when there is lock escalation.

Slide 28: Lock Snapshot

You can also use lock snapshot. Snapshot is a facility in DB2 which allows you to take a picture on your environment and you can do it for different sections, but in this case it is just for locks. And with that, basically, that is the end of lock escalation. And again lock escalation is not good; you probably see degradation in performance. So if you see the degradation in the performance, take a look at db2diag.log and see if there has been a lock escalation. If there has, then try to either COMMIT more in your application, if the logic permits for that to happen, or increase the LOCKLIST or change the MAXLOCKS. So moving on to the next section, we are going to now talk about Lock Wait.

Slide 29: Lock wait

What is lock wait? Well lock wait is, basically the case where there are two applications trying to access the same row, maybe one of them is holding an exclusive lock, for example and the other one is maybe trying to do either... trying to get an exclusive lock or a shared lock on the same row. So the second application will have to wait. How long will it have to wait? Well that is determined based on this parameter called LOCKTIMEOUT. Now LOCKTIMEOUT by default is set to minus one, which means infinite wait. So that means the application may have to wait forever. You could set... LOCKTIMEOUT to 30 seconds, and then that means after 30 seconds, the application will get an error message saying that... there is a LOCKTIMEOUT. You can also use that special register which is CURRENT LOCK TIMEOUT. And that is for a particular connection only.

Slide 30: Lock wait

Now we can move on to Deadlocks. At the end of this presentation, I am going to give you some demos as to how lock wait works, as to using 'with UR' for example, for one statement, and also examples of deadlock.

Slide 31: Deadlocks

So what are deadlocks?

Slide 32: Deadlock Causes and Detection

Normally deadlocks are situations that happen due to bad application design. That is what we covered the section on the application part of these e-learning courses because the deadlock and this type of locking problems are normally caused by bad application design. So a deadlock will be explained with an example where we have user A that is holding the Raisin Bran (or the cereal), and then we have user B that is holding the milk. User A will not release the Raisin Bran until he or she gets the milk. Well user B will not release the milk until he or she gets the Raisin Bran. So that is why we have a deadlock in this situation.

Slide 33: Deadlock Settings

Deadlocks in DB2 are managed by the deadlock check time configuration parameter. Actually what DB2 does is when there is a deadlock, DB2 will use an internal algorithm to pick one of the two applications that are causing the deadlock, and then it will say, ok, you, based on my internal algorithm, you are going to rollback and you are, the other one, you are going to be allowed to continue. So the one that is going to rollback is going to get an error message saying that, you got a deadlock and I had to roll you back, while the other one will be allowed to continue. Again, deadlocks are problems normally from the science perspective. And in terms of best practices, for working with locking, we recommend you to commit as often as you can, depending on obviously on the logic of your application. Depending on what you are trying to do, you could for example (if you are trying to delete a large amount of data), you can use something like **alter table activate not logged initially with empty table** rather than doing a **delete** ... issuing a delete of the entire table. You can delete just batches, so whatever you just need to, just delete what you need to. You can set a LOCKTIMEOUT to 30 seconds rather than using the default to infinite. And you can just retrieve what you need. Don't retrieve more than what you need to. So those would be the best practices.

Slide 34: Best Practices

So now let me give you some demos on the things that I have talked about, like LOCKTIMEOUT, like isolation level and deadlocks.

Switch to paint

Ok so for this, I am going to use an example where, let's say, I have an application A, and I have another application B. And they are trying to access rows from this table which is let's say the 'employee' table that is part of the SAMPLE database. So that is the 'employee' table. There are many columns on the table, I only put two; 'Employee number' and 'first name' and ... we are going to use two rows in particular. So I have not put more rows than that, but let's say application A is doing an update, so let's open as well two windows. So I go Start > Run, and then I type **db2cmd**.

Switch to command window

I am opening one command window at the top. So let's say that is application A. And I am going to do the same things here. Run **db2cmd**, and I open another application B. And from windows, it will be the same thing, but you can open two terminal windows. Now from each of them, I am going to connect, so from the application A, I am going to say **db2 connect to sample**, so I am going to connect to the SAMPLE database. From here, I am also going to connect to the SAMPLE database. So I have two applications connected, both of them to the SAMPLE database, so the one on the top is application A, the one at the bottom is application B.

Switch to Notepad

Now what I am going to do is (I copied this to Notepad), so it is easier for me to perform these operations. So from application A or command line processor window number one, I am going to do the following:

Switch to paint

I am going to do, let's say, an update. And then I am going to do an update on the row with (sorry for my writing here, but anyway), application A is trying to access or do an update on the row with an employee number of fifty. And it is going to because it is doing update, it is going to take an X lock; an exclusive lock. Ok.

Switch to paint

Now I am going to do that by running this statement. So I execute this statement. And note that I am using the flag '+c', that means do not commit. So for example, for this particular application, which is the DB2 command window, there are several flags and the '+c' means do not commit because this particular application **autocommits**, so I want to prevent from auto committing because I don't want to release the locks. So I am going to press Enter. And the statement performs successfully. And the lock is held.

Switch to paint

So right now, what we have is this application, for example, application A, which is the first command window, and this takes the next lock.

Switch to notepad

Now what we are going to do is... we are going to go to the other application. And we are going to run an update, but on another row, which is row number thirty.

Switch to paint

So basically from application, as you can see, application B, we are going to do an update on this particular row. So we are also going to take an X lock (exclusive lock) because I am doing an update. So we do an X here as well.

Switch to cmd window

Now let me run that from this window here. And I press Enter. As well, I am using that '+c', which means do not commit. So now I have two applications that do updates on different rows; therefore, I have exclusive and exclusive over here.

Switch to notepad

So now the first thing we are going to look at is we are going to see a timeout situation. So from the command line window number one, I am going to run this statement: 'db2 +c' again, so I am not auto committing. And issue a SELECT first name, from employee, where employee number equals to thirty. So, basically from the first application, within the same transaction, because I am not committed that means it is still the same transaction. What I am doing now is I am doing a SELECT on this particular row. So basically I am taking a shared lock. A shared lock here, now, because the application B was taking exclusive lock on this row, then this other application A will have to wait, so it will hang and hang in there.

Switch to cmd window

So let's see the behavior here after I press Enter here. So I press Enter and you see the prompt has moved to this side; to the bottom. So now it is waiting, waiting, and waiting. And it is basically going to be there forever because as I showed you before, there is a parameter LOCKTIMEOUT, by default is set to minus one, and I have not changed that. So that means it is an infinite wait. So this is the timeout and because LOCKTIMEOUT was set to minus one, I have to wait forever. I set to 30 seconds and I will get an error message. Now I am going to kill this SELECT. And I am going to, I am just going to do a CTRL-C to interrupt. I am going to interrupt this; CTRL-C. And I am going to run the same query again but I am going to add 'with UR'.

So basically for this particular SQL, I am going to say that I want isolation 'with UR' because otherwise by default, this application is using the Cursor Stability or CS. Ok, so it is the same scenario. But now I am saying 'with UR' and I press Enter. And look, I do get the result! Even though the locks are being held/taken, let's see from this chart, even though there is an exclusive lock over here, now I am going, when I am using 'with UR', then there are no locks taken here at all. So this can go in and read the information. So there is no shared lock and you can read the information. Ok that is the second demo, let's say.

Switch to cmd window.

On using 'with UR' for isolation on one particular statement, let me go back and remove this 'with UR'. And try again. And again you will see it will hang. So we will leave it like this.

Switch to notepad

And now I am going to move on with an example. But this time, we are going to do a SELECT, but on row number 50. And this is going to be done by the application B. So basically this is application A, this is application B. And now what we are going to do is (this application is) we are going to do a SELECT on this row; on this row. And because we are doing a SELECT, basically, I am trying to get a shared lock. I am getting a shared lock here. But what I just caused is basically a deadlock, right? Because one application is waiting for the other one and, for a resource, and application B is holding a resource...(and) application A is holding the other resource. So what is going to happen is we just created a deadlock. So let's execute that SQL statement from this other window. Right click, sorry, let's do it again. And I am going to press Enter here.

And what is going to happen is (probably) will take about 10 seconds or so. And then one of the applications will rollback, and the other one will be allowed to continue. You just saw that happening, right? The one at the bottom was rolled back, so application 2 or application B was rolled back. And here is the explanation 'the current transaction has been rolled back because of a deadlock or a timeout. Reason code 2 '. And the other one was allowed to continue, so I got here, the right result, which is the first name which was Sally for this particular employee number. So this is the other demonstration I wanted to show you with respect to deadlocks. And it took about 10 seconds because it is based on the parameters ... deadlock check time (which I mentioned to you before). And this parameter basically says, if you set 10 seconds (it) means that DB2 will check every 10 seconds if there is a deadlock. So every 10 seconds, is there a deadlock? Every 10 seconds, is there a deadlock? And so on, and so on. So anyway, you can try and practice this scenario yourself.

Switch to ppt

So let's move on with the next slide

Slide 35: What's Next?

Basically that would be the end of this presentation. So thank you very much for listening to this lesson and congratulations because you completed the lesson. And what is next: you can continue with Lesson 12, which is SQL PL Stored Procedures. Thank you and have a good day.