

# Lesson 6 Transcript: Working with Database Objects

## Slide 1: Cover

Welcome to Lesson 6 of the DB2 on Campus Lecture Series. Today we are going to talk about how to work with database objects. 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: Introduction

And we will start with Introduction.

## Slide 4: Database Objects

So, for introduction, these will be the database objects that we will be covering, starting with schema, tables, views, indices, and then some database application objects. Now, not all of these objects will be covered in this presentation. We will just talk about sequences in this presentation, but triggers, user-defined functions and stored procedures will be covered in a different lesson.

## Slide 5: Agenda- Schema

So, let's start with the schema.

## Slide 6: 1) Schema

The schema is a concept that in other databases may mean something else. Normally, when we talk about the schema in relational databases, we think of entire system structure... the entire structure of your database. But, schema in DB2 normally refers to the prefix of an object. So, every object in DB2 has two parts, as you can see from the slide. The first part is the schema name, and the second part is the object name, and these are divided by a dot. Now, the schema name does not need to be provided. It is not a requirement to always provide a schema name when you work with objects in DB2 because there is an implicit schema name that will be used.

## Demo

So let me just give you some demonstrations on this. Let's say I'll connect to the SAMPLE database. And I can use a specific user, so I can say **user db2admin**, and if I just press Enter, it will prompt me for a password, so I'm going to type the password here. And then I log in. I logged in as "db2admin". Now, if I do **db2 create table abc.t1**, and I put any columns, say column 1, it's an integer, column 2, it's also an integer, and I press Enter. I just created a table abc.t1, where the schema is **abc**, and the name of the object... or the table is **t1**. Ok, good. Now, let's say I want to select from table t1, if I just do **db2 select \* from t1** and I press Enter, I will get an error. And looking at what the error indicates, it says that the table DB2ADMIN.T1 is an undefined name.

So, when I issue a **db2 select \* from t1**, I didn't provide the schema name. And then DB2, by default, will take, as an implicit schema, the name that I used, or the user ID that I used to connect to the database, which was db2admin. Ok, so that's basically that the behavior, that is, the default behavior occurs in DB2 when you don't provide a schema explicitly. Now, as you can see as well from this example, abc does not need to map to any user. So abc doesn't map to a user in the operating system. But, when you don't provide a schema, then the user ID used to connect to the database is used as the schema.

Ok, so let me give you one more example. So if I had a **db2 select \* from abc.t1**, then in that case, yes, I will be getting the right result, because I used the schema abc. Now, if I create a table, let's say, **db2 create table t1 (col1 int)**, and I press Enter, now I have created a table with no schema. So, again as before, I should have this table, which would be db2admin.t1, so I would say **db2 select \* from db2admin.t1**, then I do get the right result because now there are two tables, one with the schema of abc, and the other one with the schema db2admin. And this is a typical problem that happens for many new users. They start creating tables or objects in DB2 with different schemas, because maybe they are connected differently, with different users.

So because the user is used by default as the schema name, if you don't provide one, then they are basically creating two sets of tables with different schema. So that's a typical problem. And, as I said before, a schema applies to all objects in a database in DB2, so even if I create a procedure, if I say, for example, **db2 create procedure**, let's say, **p100**, it's a procedure that does nothing. So I just enter it like this. Then the procedure name would be **p100**, but it's actually... if I try to call this procedure, I would be invoking it as **db2 call db2admin.p100** and DB2 finds it.

Now, if you want to change the schema for your current session, you could do a **set schema**. So you could say **db2 set schema**, and then maybe you can call it whatever you want: **db2 set schema raul**, for example. And from then on, anything you are doing would apply to, or would start using this schema, which is the schema I just created right now. So, if I now do **db2 select \* from t1** and press Enter, then I get this error because it's trying to look for raul.t1, since I said that my schema... my current schema from now on should be raul.

Ok, so that's all I want to mention about the schema, and basically a schema can be used for grouping a bunch of tables together based on the schema. For example, some utilities in DB2 like **db2move**, which we will cover later, may be using schema for just moving some specific tables based on their schema names. So, again, the purpose of having schema is just for grouping objects that may be related.

### **Back to slide 6: 1) Schema**

So, maybe one group of objects could be used for test or for development, or maybe for production, within the database.

## **Slide 7: Agenda**

Ok, moving on to the next section, we are going to talk about tables.

## **Slide 8: 2) Tables**

And, tables, like any other relational database system, we basically have the same, or pretty much similar syntax. So all that I want to mention here is that in DB2 you have to use the clause **in** and then put the name of the table space where you want this table to reside. We talked about table spaces in the lesson about DB2 environment. So, this is the syntax you need to use to specify where, or in which table space you want to put this table. So, defining a table is usual, you just specify a column, the data type, whether it's not null or null, we will talk about this later, and whether it has some default value.

## **Slide 9: Data Types**

Within tables, you have to specify columns, and the columns have to have a data type. So these are the data types that are supported in DB2, and the new one starting with DB2 version 9 is the data type XML. We will talk more about this data type in a different lesson when we talk about pureXML.

## **Slide 10: Large Objects**

We support large objects, which is a data type that is supported in DB2, and these are used, for example, if you store binary, large binary files. So, we will use the BLOB data type. If you use large character strings, then you can use CLOB, or Character Large OBject. Or you could also use DBCLOB, for example, when you want to store double-byte characters like Chinese.

## **Slide 11: User-Defined Types**

Now, besides having built-in data types like integer, character, BLOBs, CLOBs, or XML, we also allow you to define your own user, or your own data types. These are called user-defined data types, or user-defined types. And the syntax that you use is, in this example, `CREATE DISTINCT TYPE`, the name of the type, `AS INTEGER`. So, basically it will be built based on an existing built-in data type.

The purpose of using user-defined data types is the following. Let's say you created a table like shown here, "CREATE TABLE person", and with two columns. Now, for this particular example, before using these user-defined types, let's assume that `weight_p` and `weight_k` are defined using an integer data type. So `weight_p` is storing the weight using pounds, and `weight_k` is storing the weight using kilograms. Now, what is going to happen if I perform a comparison between these two columns, and they were defined as integers? If I issue command comparison, like issuing a `SELECT`, for example, where I say "get me all the records where the `weight_p > weight_k`", what's going to happen is that the query will be allowed to continue—because in terms of syntax there's no problem. But in terms of logic, it doesn't really make sense to compare the pound and kilogram in the same query. So that's basically the purpose of user-defined types: basically to make sure that these types of queries cannot happen. Basically you will get error messages if you try to do such comparisons.

So, going back to this example, we created two distinct types, two user-defined types. One is called POUND, the other one is called KILOGRAM. And then when we created a table, we used these two new data types, which are based on integer built-in data type. And now, when we issue a select statement, for example, here at the bottom we do `SELECT F_NAME FROM PERSON`, and we say `WHERE weight_p > weight_k`. Now, this will fail because these are two different data types. Even though they are based on integer data type, this will still fail. And that's what we wanted. We wanted this to fail, because logically this should not make any sense. So that's what we wanted. Now, also here on this other example, `SELECT F_NAME FROM PERSON WHERE weight_p > POUND(30)`, here, we have to put `POUND(30)`, we cannot just put `POUND`. I'm sorry, we cannot just put "30", because that would mean that we are trying to compare `weight_p`, which is defined as `POUND` to an integer. So we need to do a "cast" of "30", and use the function `POUND`. Now, in this function, `POUND` was created automatically when we added it as **integer with comparisons** in the definition of the `POUND` user-defined type. So that's all I want to mention about user defined types.

### Slide 12: Null Values

Moving on to another slide, we have here "Null Values", and null values are similar concepts as in other relational database management systems, which means that they will represent a null state. And we can put `NOT NULL` to make sure that you always put a value for a particular column. That's what `NOT NULL` is for. You can also put default values. Like in this example, you can say **not null with default 10**. So that means that that column always has to have a value and if you don't put your value, then the default will be 10.

### Slide 13: System Catalog Tables

We also have system tables and these are called system catalog tables. So these are tables from the system, meaning that they store information about the system. These are known as metadata, which means data about your data. So, for example, they will store information about what tables there are, what columns there are, what views there are, what indices, so on and so on. Now, there are three schemas that are specific to catalog tables. And these are `SYSIBM`, `SYSCAT`, and `SYSSTAT`. `SYSIBM` are the base tables. `SYSCAT` are the views. And `SYSSTAT` are also views for databases statistics. Now, you normally should not manipulate these tables. You should not delete or try to make changes to the system tables. However, some of these tables, like database statistics, allow you to make some changes to them. Normally you don't work with the `SYSIBM` tables but you work with `SYSCAT` views.

### Slide 14: Declared Temporary Tables

Moving on to the next slide, it is talking about declared temporary tables. So these are temporary tables in memory that are created by your application. So your application, or many other applications, can create their own temporary tables. And they will be dropped automatically once the application disconnects from the database. So these tables are useful because maybe you want to perform some temporary operations as part of your application, and you could use a temporary table. The nice thing about this is that basically you have your own copy of this table.

Therefore, there will be no contention, so there will be no locking... no locks taking on any of the rows of this table because you are the only one accessing these temporary tables. There will be no logging, though logging is optional. There will be no contention with the catalog, no authority checking. So basically the performance when you access these tables will be very very good. And you can also, if you want, create indices, and you can also RUNSTATS, which we will talk about in a later lesson.

### **Slide 15: Temporary Tables**

If you want to create a temporary table, you don't use the create table statement, what you use is the DECLARE GLOBAL TEMPORARY TABLE statement. And there are different ways to do this. So we provide here some examples. And when you create... DECLARE GLOBAL TEMPORARY TABLE, before you do that you have to CREATE USER TEMPORARY TABLESPACE. So these temporary tables will reside on this user temporary table space, which is different than the system temporary table space. And the system temporary table space is, for example, tempspace1, which was discussed in Lesson 2 when we talk about the DB2 environment.

### **Slide 16: Identity Columns**

Moving on to the next slide, we have *Identity Columns*. Identity columns allow me to generate a unique number for a column of a table. So this is specific to a table, and you can generate these columns either using **generated always**, which means DB2 will always generate the values, or if you try to insert a value into this column, you will get an error because DB2 will always generate these values. On the other hand, you also have another option, which is **generated by default**, and when you issue, or when you use this clause, which means that DB2 will always generate the values but it will also allow you to insert a value yourself. So, you can insert your values into this column and DB2 will not give you an error message.

### **Slide 17: SEQUENCE objects**

Now, another concept that is similar is SEQUENCE objects. Now, sequences, they should not be covered here because they are not part of a table, and we are still talking about tables. But because they are similar in terms of what they do with respect to identity columns, I decided to put them here. So sequences allow me also to generate a unique value, but these unique values will be across the entire database. So it's not like identity columns, which are only going to be for a specific table, but sequences would be for an entire database. To create a sequence you use the following syntax, and I'm just providing a very simple syntax where you say **create sequence**, and the name of the sequence starting with what number, incrementing by how much, and whether it would cycle or not cycle; so whether it will start from the beginning or not. And you can insert these sequence values into tables, as shown here. And you can use **nextval**, the next value for the sequence, or you could use **prevval**, the previous value for the sequence.

### **Slide 18: Quicklab #7**

Ok, so, moving on to the next slide. Now, you can work on Quicklab number 7 to allow you

to practice on how to create a new table. So I suggest you to pause this presentation right now, and take a look at Quicklab #7 instructions.

### **Slide 19: Row Compression**

Ok, so, assuming that you already finished with Quicklab #7, I'm going to talk about two features related to tables but they are not in DB2 Express-C. I'm just putting them here because... again, you may be wondering if these features are available on DB2, because many people asked me if these features are available. They are present in DB2, but not with DB2 Express-C. So they are available with other editions of DB2 but not with DB2 Express-C.

So, row compression is available, and one easy way to remember what is available with DB2 Express-C and what is not available with DB2 Express-C is the following: DB2 Express-C was designed for small-to-medium size businesses, for community, for university students. Therefore, things that are applicable to large companies would maybe not be available in DB2 Express-C. So row compression is only something good when you work with data warehouses, when you are working with large amount of data, which would be available in big companies. Therefore, row compression is not available with DB2 Express-C. Now, row compression allows me to basically compress the rows in a table, and there needs to be a dictionary, and it needs to be created beforehand. But... anyway, you can read more about this from the DB2 manuals.

### **Slide 20: Table Partitioning**

We also support table partitioning, which allows me to do what we call *roll-in/roll-out* type of operations, where, let's say, you have a huge table, and you can partition this table across many table spaces. And you can, maybe, roll-out some information that you don't use, for example, December 2006 is old information. So, you can roll that information out, so that it's not part of the table anymore. And you can roll-in, let's say, January 2008, which is now something that will be required as part of your report that you are writing. So you could do these types of operations when you have table partitioning, and it also allows you to store more information because now you can store more information since you can spread them across different partitions. So, in this example, for example, you knew that you could store information only in one table space and you could store letters, let's say it's a dictionary going from A to Z. But now you can spread this information and you put some information in one table space from A to C, another one from D to M, etc. And in terms of utilities, you can also perform these operations or these utilities independently, not on entire table, or you can also do it on the entire table as well.

### **Slide 21: Agenda**

OK, so moving on to the next section about views, indices and referential integrity. We will cover them very briefly because most of the concepts are the same as in other relational database management systems.

### **Slide 22: 3) Views**

So then we are going to move to views, and views are like a virtual table. It's really not much different from any other relational database management systems in terms of views. So you can create a view based on a select, and then you can do a **select \* from <the view>**, or from... you can do all the combinations. You can select different columns from the view. So you just create it as another table, but it's really a virtual table.

### **Slide 23: Agenda**

### **Slide 24: 4) Indexes**

Ok, so, with respect to indexes, we support, ascending or descending indexes, unique or non-unique, compound, cluster, etc. And this is a very simple example on how to create a unique index. Now, you could design your own indices, but with DB2 we have a tool called the "Design Advisor".

### **Slide 25: Launching the Design Advisor**

And I talked very briefly about the Design Advisor when we were talking about Tools.

### **Demo**

So let me very briefly show you the Design Advisor. I already started the Control Center here just to save some time. So you can invoke the Design Advisor by selecting a database and right-clicking and choosing Design Advisor. So let's start this Design Advisor. Now, the Design Advisor, it's like a wizard, so you will have to go through different steps. So, click Next and you can design several things, but we are just going to concentrate on indices, so we are not going to cover these other things that you may use also on other editions of DB2 that is not in DB2 Express-C. So then we just choose indices, we click "Next".

Now, here you have to specify a workload. So a workload is basically a bunch of SQL statements that have, like in this example, there's not only SQL statements, but you also specify the frequency of the SQL statements. So depending on the frequency, DB2 will make a decision as to what indices would be better for this workload, given the frequency for those SQL statements. So I'm going to, for example, quickly change a workload, and I could add more queries. So I can say "Add", and then I can say **select \* from dept**, and I can put another name for this. So I can say **sql3**. And this one it is executed only 10 times. Then I can click "OK". So now I have another SQL in my workload. And I can keep writing these SQL statements in my workload.

Based on these, the indices will be created. So now I click OK. And I'm going to click Next to move to the next section. Here, I'm just going to be running some statistics on my tables, because these would be used for better calculating the indices. So I'm just going to select all the tables, and click Next. And now how much disk space I want to use for calculating my tables, but I'm just going to click Next and take the default. Then it can calculate my indices. This may take several minutes to calculate the indices, but in this particular example, you know, I just had 3 SQL statements, and I was not expecting really to get any indices

recommended, but I wanted to show you that here, you will get a performance improvement.

In this particular case you get 0 disk space, in this case you also get 0 megs. But, if you had a real scenario where you had different SQL, you may get here a quick view as to what will be the improvement of using those indices, and how much space it will consume. If I click Next, well, you will get a chance to drop some of the indices that are not being used for this particular workload. And it's good to drop indices that are not being used because if you don't do that, whenever you are inserting or updating things in a table, it would take longer because it will also have to update these indices. And if you are not using the indices, then, it may be better just to drop them because otherwise it would affect the insert/update operations on your database. So now we click Next; you could decide whether you want to run now the suggestions. On this particular example there are no suggestions, but you can run it now, or you could create a task so you can run them later. Ok, so I'm just going to cancel from here.

### **Slide 25: Launching the Design Advisor**

### **Slide 26 Design Advisor**

So that was a very brief description of the Design Advisor.

### **Slide 27: Agenda**

And finally we are just going to cover the last section of this presentation, which is referential integrity.

### **Slide 28: Referential Integrity**

Referential integrity means that you basically want to establish some relationship between different tables. So here, for example, you want to establish a relationship between the department table, which is a parent table and the employee table, which is a dependent table, or a child table. And you can do this by issuing or using primary keys, foreign keys, and, again, there's not much difference in terms of the use of referential integrity in DB2 with other relational database management systems. So we are not going to talk about this in detail.

### **Slide 29 Referential Integrity Rules**

We have some insert and delete rules, so you know what happens on one table or on the parent table if you delete something on the child table, and what happens if you delete or insert something on the child table with a parent table. So these are some rules that are available as well in DB2, like in many other relational database management systems.

### **Slide 30: What's Next?**

So with this we have finished this Lesson 6, working with database objects. Congratulations, because you have completed this lesson. And as to what is next, we suggest you start working on Lesson 7, Database Movement. Thank you.