openSAP
ABAP Development for SAP HANA

WEEK 3, UNIT 1

00:00:13 Hello. Welcome back to week 3.

00:00:15 Wow, week 3 already. That means that you have successfully finished half of the openSAP course

00:00:22 and I hope you earned a lot of points in your last week's weekly assignment.

00:00:26 Okay, today's unit is about recent Open SQL enhancements and part 1 already indicates that there will also be a part 2,

00:00:34 but I would just say ABAP Code: Ready, Set, Optimize.

00:00:38 Yes, before we start optimize, we maybe should first take a look at the data model which we will be working on and do our optimizations.

00:00:46 This is Enterprise Procurement data model which is available with every NetWeaver 7.4 system.

00:00:53 You can also generate data as much as you want to try out the Code Pushdown paradigm and so on

00:01:00 but it is important to know for your trial systems we have already generated the data.

00:01:05 So, let's take a look at the tables or start with an SNWD and then we have sales order tables with a classical header item

00:01:14 and we have invoice tables, which are basically the same but holds every invoice information, also with header items.

00:01:22 Both of them are connected to a business partner on a header level where you can find the business partner of the sales order respective to the invoice

00:01:30 and of course the product of each item position. Now we can really start with the optimization information.

00:01:40 Yes, okay. Of course we will just go back to our classical slide here and on the first part you know that we have now finished with the Detect phase

00:01:51 and now we are finally on the Optimize phase.

00:01:54 Thank you Jens. Optimize encrypts the content of the five units in this week.

00:02:01 We will start with Open SQL enhancements and that is really the first stopping point or the first point where you can start with the optimization using Open SQL.

00:02:12 And in units 3, 4, and 5, we will come to Core Data Services and what that is and what is in there for you. We will talk about that.

00:02:22 Okay, so Open SQL. You might ask, what is Open SQL? Or you might say, okay I know what Open SQL is.

00:02:30 That certainly is true. I think 99% of you have already worked with Open SQL.

00:02:36 So Open SQL in our ABAP Application Server is the database abstraction layer calling an SQL-like syntax

00:02:44 but what might not be known to you is that Open SQL is the database abstraction layer and actually the only database abstraction layer—

00:02:55 that's a crazy word, database abstraction layer—that has a common semantic for all of SAP-supported databases.

00:03:03 That is important for you as we've talked about migration to SAP HANA or database migration in general.

00:03:10 There is no problem in migrating Open SQL code from one database to another because it has the same semantics on all databases and you can just use it as before.

00:03:21 The only thing that we had as an exception was database-specific hints. Of course, they are not respected if you are migrating to another database.

00:03:30 So it's not just a subset of SQL commands, but it has a common semantics. So, the question is, I know already Open SQL

00:03:39 and I used it before. What's in there for me in ABAP 7.4 SP2 and beyond?

00:03:44 Yeah, let's take a look at it. As you can see in the diagram, there is classical Open SQL, should define the amount of features,

00:03:53 and as you can see there are more features available. What those are in detail we will show you here in this session later

00:04:02 and of course, in part 2. But first of all a look at this SQL92 standard. So SQL92 standard was created when?

00:04:11 92. So quite a while ago. This was a standard where the most common databases...

00:04:20 Sorry, let me rephrase the sentence.

00:04:23 So this was a standard when the most common databases used to have a feature set which is available on all of them.

00:04:30 So, now why isn't Open SQL in that standard? This is the case because we had to do a semantic layer for all of them

00:04:39 and with 7.4 we could enhance it and what's also important is that Open SQL will really help you doing the code pushdown with a very easy way.

00:04:50 Follow this paradigm. Let's take a look at the feature. I think this is more impressive than talking about slides.

00:04:59 Yeah, thank you, Jens. So what I will be showing you in this unit is the syntax enhancements. That is not interesting performance-wise

00:05:09 or for the Code-to-Data paradigm but there is a new syntax

00:05:15 and it had been necessary for us in order to incorporate the new features.

00:05:20 So I would like to make you aware of these syntax enhancements especially about escaping of host variable and the comma-separated SELECT list.

00:05:28 Then we will dig deeper into the SELECT list enhancements, so what is new there.

00:05:34 I will be showing aggregation functions, literals, and arithmetical expressions

00:05:40 and there will be even more features in the next unit, so the part 2 unit on Open SQL.

00:05:46 Let me just jump to the system. I have prepared a couple of reports for you and the very first one is just on syntax.

00:05:54 You see, we've talked about the Enterprise Procurement Model. Here is the sales order table and I'm just selecting the sales order ID,

00:06:03 the gross amount, and the corresponding currency code. Let me just transform that for you to the new Open SQL syntax.

00:06:12 I will just copy it. I talked about comma-separated field lists and that still gives me an error message

00:06:20 and that is because I told you about escaping of host variables, so I need to put the @ sign here.

00:06:28 Those of you familiar with SQL on the database level know that they also have escaping of host variables that is the colon (:).

00:06:38 However, colon in the ABAP was already in use.

00:08:42 So those are two important things concerning statements: comma-separated field list and the @ sign.

00:06:51 But it is also very important for you to know that you don't need to change your whole report now to the new SQL statement

00:06:59 or all the reports. The old style will stay intact so you will still be able to use it

00:07:05 and at the new one or change a classical statement just on a statement level.

00:07:10 As you can see here in the report, we have on top the classical syntax and down beneath it we have the new syntax

00:07:18 and maybe you can just now activate this report and you see that it will work without any problems.

00:07:24 So just to make your life easier, you can switch a SQL statement in a report without having to change the rest of the SQL statements.

00:07:33 Okay and one other feature which might not be that important for productive use cases

00:07:40 but to showcase things for you in the next couple of units, I will also use the INTO TABLE @DATA so here the internal table lt_so_amount

00:07:53 I have defined with a type statement and a data declaration, but I can use the type inference functionality in Open SQL.

00:08:02 So I just say INTO @DATA and the lt_result, the internal table, will be inferred from the SELECT list given here.

00:08:12 Okay, that's just so that you have seen this type of inference be used in the example reports.

00:08:19 Okay, enough about syntax. Coming to aggregations.

00:08:24 So we've been talking about Code-to-Data paradigm and it is important that if you have data-intensive coding, you really let the database do that for you.

00:08:35 The very first or very simple example is an aggregation, for example, because you only need to transfer the aggregated result set.

00:08:44 So here I'm selecting the business partner and company name—company name of, of course, that business partner—

00:08:51 and the gross amount and the corresponding currency. And I summed that up and of course I have to do a GROUP BY.

00:08:59 And in order not to you only the syntax but also to show you the result, let me...

00:09:06 and because you don't want to see me type all of the time, I have again a tiny little template on that. I will just execute that report and display the results.

00:09:16 We have about 45 business partners or customers in the system and you see this is nicely displayed and it works fine.

00:09:26 And fast.

00:09:28 And fast!

00:09:30 Because it was using Code-to-Data paradigm. Okay, what is also in there, in Open SQL enhancements, are literals in the SELECT list.

00:09:41 So here I just use a SELECT and I am selecting X, or abap_true as you might know, or for example, an integer literal.

00:09:52 I can just do that. Of course, that does not make too much sense concerning the Code-to-Data paradigm

00:09:57 because I already have that X and the 42 on my application layer, but it might come in handy, for example, in the existence check.

00:10:05 So if you remember when we talked about detecting performance optimization potential, here we had also the syntax...

00:10:15 sorry not the syntax check. We also had the existence check as a very prominent part of those things and we have seen really bad coding there,

00:10:25 for example, if we do a SELECT * just to see if there is an entry in that table.

00:10:31 And we can do this better, especially now with that literal X which is, by chance, the abap_true value.

00:10:40 So let's take a look at a new check there, how you can do an existence check in the most performant way from an ABAP perspective

00:10:49 and that here is the SELECT SINGLE @abap_true. Of course, you could write the X, but this is much more meaningful

00:10:57 so you know what you are doing here from the table and to one variable.

00:11:02 And afterwards, you can easily do an IF statement with the abap_true where you can still code when the line exists, and elsewhere something else.

00:11:12 It is especially important. I mean, of course you could do a SELECT SINGLE on one single column,

00:11:19 which is from a performance perspective much better than the SELECT * or the SELECT SINGLE * statement.

00:11:25 However, that is really a generic check, so you don't need to know which columns that table includes. So just a generic check.

00:11:36 Okay and last but not least, I would like to show you some arithmetical expressions.

00:11:41 So here, for example you sum 1 + 1.

00:11:47 That is certainly not that important for Code-to-Data paradigm

00:11:53 because it is just a code to database because Application Server, of course, can sum 1 + 1 in the application layer.

00:12:01 Just to showcase for you what, for example, is possible.

00:12:07 Something else for example, I could use the gross amount and do a reduce gross amount and just multiply that with a given host variable,

00:12:16 here a discount rate. I put that to 80%. Or, for example, I could use functions like CEIL

00:12:25 and CEIL actually is for me much better concerning salary, maybe gross amount, not if you are the seller of the product.

00:12:33 Or yes, of course if you're the seller of the product, but not the buyer of the product.

00:12:39 Okay, a lot more functionality is available and have a look at the next unit and of course at the documentation.

00:12:50 And with that, I would like to go back to the slides and say see you for the next unit.

00:12:57 Yes, and just a short summary, remember we have enhanced Open SQL to enable you to follow the Code-to-Data paradigm

00:13:07 and Open SQL is the first and easiest way to follow this paradigm.

00:13:11 So thank you and see you in the next unit where we take a look at more Open SQL enhancement.

00:13:19 Good-bye.

WEEK 3, UNIT 2

00:00:13 Welcome to the second part of Recent Open SQL Enhancements. You're in week 3, unit 2.

00:00:19 In the next unit, we will also be joined by Jens again.

00:00:23 Okay, so what is the topic of today's unit?

00:00:27 We have seen in the previous unit what Open SQL recently learned, in particular what has been done concerning syntax

00:00:36 and some features in the SELECT list that we have already seen in the demo.

00:00:43 In this unit, I will be showing even more new functionality in Open SQL.

00:00:48 I will show you, for example, conditional expressions in a SELECT list, GROUP BY, HAVING clauses, and also JOIN statements and client handling.

00:00:58 It is important to remember that everything that I am showing you is because with Open SQL, we would like to do code to data.

00:01:07 We would like to follow the Code-to-Data paradigm. So it's not just showing you these features because they are there

00:01:13 and because we can say yes we can, but it's important that you really should apply them for your Code-to-Data paradigm.

00:01:22 Okay, so let me show you in the system directly these SELECT list enhancements that I already mentioned

00:01:30 and expressions in GROUP BY and HAVING and JOIN and client handling. Jumping into the system again, we have prepared a couple of reports for you.

00:01:40 Let me start with a CASE statement. There are two methods, or two kinds, of CASE statements in Open SQL:

00:01:50 the so-called simple CASE statement and search CASE statements.

00:01:54 Simple CASE is just CASE column WHEN construct,

00:01:59 while the search CASE is a CASE WHEN construct. But let me start with the simple CASE.

00:02:05 Here I have a small example again on our sales order table and I'm asking myself, the delivery status ' ' or 'D' what does that mean?

00:02:15 And in order to give it a more human readable name, I will say ' ' (space) is equal to OPEN and 'D' is DELIVERED.

00:02:25 So a search CASE statement can do more complex things. Again on the sales order table, here I am asking myself,

00:02:34 I have a certain gross amount and if that gross amount is above a certain value, I say okay, that is a high volume sales order.

00:02:42 Else I just give it a space.

00:02:46 Of course, there could be even more complex things and you can even have CASE inside CASE, so nested CASE statements.

00:02:55 And I have a little bit more advanced of course it could be even more advanced things in your

coding.

00:03:01 I'm a little bit more advanced. I'm asking which company name is in there

00:03:09 and if the company name is equal to a given value, I give for those company names a discount on the gross amount.

00:03:19 So, you can now already imagine what you can do with a CASE statement in your ABAP coding.

00:03:27 In addition to the CASE statement, we have another conditional expression provided for you. That is the so-called COALESCE expression.

00:03:36 COALESCE is a function that is taking two arguments and it either returns the first argument, if it is not NULL,

00:03:47 or the other. And now there is something very special for ABAP developers because NULL is typically not there for you in the ABAP.

00:03:56 You are certainly more familiar with initial values. However, with, for example, LEFT OUTER JOIN statements,

00:04:06 you could create a NULL value. Here is a LEFT OUTER JOIN on the sales order table and joined to the invoice header table.

00:04:15 LEFT OUTER in that example is if a sales order has not yet been invoiced

00:04:22 there is no sales order invoice and in that case, of course, also the gross amount is NULL.

00:04:27 So, I have the COALESCE function on this invoice gross amount, and in case that is NULL, I just return the sales order gross amount.

00:04:37 So much about conditional expressions.

00:04:43 I could talk for hours on that new functionality in the SELECT list; however, I think that it is good only to give you a couple of hints what's in there

00:04:53 and you go to your trial system and try all of the things out on your own.

00:04:57 Have a look at the documentation, have a look at SCN showing you new functionalities. All of these example programs are there in your trial system.

00:05:06 Try them out, enhance them, do a lot of things just try all the things out on your own and have a look how it works.

00:05:15 Okay, in addition to the SELECT list enhancements, Open SQL also learned new things in the GROUP BY clauses and HAVING clauses.

00:05:24 Let me start with GROUP BY. Closing them in order not to be confused. GROUP BY I need if I have for example, aggregations.

00:05:34 In that example, I have an aggregation on the gross amount again, and I have a CASE statement.

00:05:41 So for those sales orders below a certain gross amount, I say that is low volume sales order. And for all others, I just give it a space.

00:05:52 I could, of course, also have put here the abap_true. If I do that CASE statement in the SELECT list

00:06:03 and do an aggregation, of course, I have to mention again the CASE statement in the GROUP BY and that's exactly what this example is about.

00:06:11 So I have to repeat in the GROUP BY also the CASE statement. Something which is now available in Open SQL.

00:06:19 So it's just the CASE statement repeated from above here.

00:06:24 In order not to show you only these examples, but to show you also that they work, let me just execute that

00:06:32 and you can see here I have for each company name now two entries,

00:06:39 so for each business partner, and that's because I have a GROUP BY also on that CASE statement.

00:06:46 You see I have one entry with a certain total gross amount, where the sales order is below the certain value

00:06:55 and of course the rest with a SPACE statement, with the value space here.

00:07:01 Okay. In addition to GROUP BY, I mentioned already HAVING. That becomes very handy here, for example,

00:07:12 when I do an aggregation again on the gross amount but I would only like to take into account

00:07:19 those aggregated gross amounts of each business partner if they are above a certain value.

00:07:26 So here with that HAVING clause and the aggregation again in the HAVING clause,

00:07:33 I can filter only on that total gross amount which are very high volume.

00:07:40 Okay, again I execute it in order not to only show you the details which you can, of course, have a look at on your own in the system.

00:07:48 Here you see I have 45 business partners and they are reduced to the 5 business partners which have a total amount above this certain amount.

00:08:02 So much about GROUP BY and HAVING clauses. And I have mentioned that I will also show you something on JOIN statements and on client handling.

00:08:12 You are well aware of JOIN I think, about the INNER JOIN that has been in the Open SQL for some years already.

00:08:20 What we have newly provided for you are the RIGHT OUTER JOIN statement and we have removed a couple of limitations, for example, in the ON condition.

00:08:31 Let me show you that in an example program. So here, for example, I do a RIGHT OUTER JOIN

00:08:39 and I'm joining the sales order with the business partner table. That allows me, with a RIGHT OUTER JOIN, to also have the business partners

00:08:48 which don't have any sales orders in the system.

00:08:53 And as I mentioned, we have also improved the ON conditions, or enhanced the ON condition possibilities.

00:09:00 So here in the ON condition, I can have the sales order gross amount which is above a certain value.

00:09:09 So not only the equal sign is allowed now, but also greater or less than.

00:09:17 And last but not least, I would like to say a couple of words on client handling.

00:09:23 Client handling, of course, is familiar to you. A very prominent feature of Open SQL.

00:09:30 I think you are well aware that each Open SQL statement on a client-dependent table, which is the majority in your system,

00:09:40 is enhanced, so if I do a SELECT on a table, your SELECT statement is enhanced with a WHERE clause on the logon client.

00:09:49 And in order to overrule this client handling which is automatically done,

00:09:56 in order to overrule that, you can use now, with our new Open SQL features, the so-called USING CLIENT.

00:10:06 In the old days or in the classical Open SQL, you would have used the CLIENT SPECIFY and then mentioned again in the WHERE condition

00:10:15 the client field. With USING CLIENT, this becomes even more easy. In that example, I do a SELECT statement, doesn't matter what it does,

00:10:25 and I'm overruling the standard client handling of Open SQL,

00:10:31 which would add a WHERE CLIENT equal to 001 with the USING CLIENT.

00:10:38 A small tip: There is no such client in the system and of course, then also no data in there.

00:10:44 So what do you think the Open SQL SELECT will return? Exactly. Nothing.

00:10:51 And that's exactly what you see, an empty result set, because there is no data in that specific client.

00:11:01 Okay. This has been a lot of features

00:11:07 and this is, of course, a unit where you only get a hint on how many new features you have in Open SQL.

00:11:17 So if you would like to learn more about all of these new features, go to the Help Portal or go to this valuable SCN blog by Horst Keller

00:11:27 or, even better, go to the system and try all of the things out on your own.

00:11:33 And with that, I would like to conclude the two units on Open SQL.

00:11:37 We have talked about Open SQL. That it is the only one database abstraction layer

00:11:44 which has a common semantic on all SAP-supported databases. That this is very important for you if you migrate from one database to the other.

00:11:55 We have a new syntax that has been necessary in order to incorporate the new features.

00:12:01 Remember, for example, the comma-separated SELECT list or the escaping of host variables with the @ sign.

00:12:08 And I have talked—or Jens and me, in the previous unit—

00:12:13 have talked a lot about these new features provided to you in order to follow the Code-to-Data paradigm.

00:12:20 And that is really also important. Open SQL is your first point where you can start with the optimization of your ABAP coding

00:12:30 concerning database-oriented programming.

00:12:35 With that, I would like to say thank you. Stay tuned for the next unit, when we will be talking about the Core Data Services.

00:12:44 See you.

WEEK 3, UNIT 3

00:00:13 Welcome. It's great to have you back for week 3, unit 3: Core Data Services: What's in It for You?

00:00:20 Hi and welcome back. Let me first outline this unit for you.

00:00:24 The first part of the unit will show you what the overall concept of Core Data Services is

00:00:31 and how we use Core Data Services in the Application Server ABAP

00:00:38 and of course, how you can consume those Core Data Services in your ABAP programming environment.

00:00:47 Let's switch to this very well-known slide. The Core Data Services, CDS, is here to help you optimize

00:00:56 your coding for SAP HANA. But Jasmin, let's see what's the general concept of Core Data Services.

00:01:04 Okay, thank you Jens.

00:01:07 Core Data Services — Core Data Services are a collection of the main specific languages and services

00:01:15 for defining and consuming semantically enriched data models.

00:01:20 Whoa, what a sentence!

00:01:23 So Core Data Services simplify and harmonize the definition and consumption of semantically enriched data models

00:01:32 and the best thing in that, that is independent of the consumer technology.

00:01:36 We have seen a similar concept—that is the abstraction between application and user interface.

00:01:44 In unit 1 of the first week, we have seen the end-to-end application demo.

00:01:51 We had, of course, an Application Server ABAP, on SAP HANA as database

00:01:58 and we had a user interface abstraction in terms of an OData model

00:02:06 in order to abstract the application from the user interface, which has been just a browser.

00:02:12 The same concept applies here for Core Data Services. You have the database, the SAP HANA database, for example,

00:02:21 and you have this Core Data Services as a layer between the database and the application data.

00:02:29 And I said, already, this is independent of the consumer technology, so it doesn't matter whether you are in the Application Server ABAP

00:02:39 or you're an SAP HANA XS developer, or you're in the Java stack.

00:02:44 So what is Core Data Services technically? It's an extension of SQL

00:02:51 and really, SQL not Open SQL but of SQL.

00:02:55 It extends SQL with expressions and that already gives you an idea where the Code-to-Data

paradigm comes into the game.

00:03:04 Besides the expressions, there are domain-specific metadata and associations.

00:03:10 And what all of that is, we will hear a bit in that unit but more in the upcoming units of this week.

00:03:17 Core Data Services includes several languages, as mentioned.

00:03:24 The first part, and the most important one for today's unit, is the Data Definition Language, abbreviated by DDL.

00:03:32 We have a query language and the Data Manipulation Language (DML), which is really taken care for you by the Data Dictionary or the Data Control Language (DCL).

00:03:44 Okay, that is the general concept of Core Data Services.

00:03:49 You might ask, Me as an ABAP developer, why is that important for me?

00:03:53 Of course, we would like, as Jens mentioned, to support you by following the Code-to-Data paradigm.

00:03:59 Typically you have already the persistencies on database. For example, the MARA table is already there for you.

00:04:09 So if you would like optimize, you would typically start with view definition or with advanced view-building capabilities.

00:04:20 Therefore, we have introduced in the ABAP new ABAP objects or ABAP artifacts, the so-called view entities

00:04:30 and they're enclosed in the so-called R3TR DDLS object.

00:04:36 DDL you know already. It stands for Data Definition Language and the “S” is just for source. So a Data Definition Language source.

00:04:44 You can therein define semantic and enriched data models. You will see that in a second.

00:04:50 More about the concept. The good thing with these new artifacts is they are fully integrated in the ABAP infrastructure.

00:04:58 That starts from the definition and maintaining these artifacts

00:05:04 and once you have defined a DDL source and you have defined the view in that DDL source,

00:05:10 you activate it using the Data Dictionary. And at that moment, the view gets deployed by the ABAP Dictionary on the database.

00:05:19 So, for example, an SAP HANA view is created.

00:05:23 Using that approach with creating the view during activation and the ABAP Dictionary

00:05:30 allows you to use the standard CTS, Change and Transport System, in the ABAP.

00:05:35 So you only have to transport the dictionary object which is this DDLS object to the target system

00:05:44 and once activated in the target system, the view gets deployed.

00:05:48 One word on defining and maintaining. You cannot do that with a standard workbench. You have to use the SAP Development Tools in Eclipse,

00:05:57 in particular, the ABAP Development Tools that we have met in the first week, unit 5.

00:06:04 Last but not least, of course, you would like to consume it. And of course you can do that with standard Open SQL.

00:06:11 But enough of the theory. Let's jump to the system and create such a view.

00:06:15 Okay, for this let me switch to the mentioned development tools,

00:06:23 which you might already, hopefully, know from your trial systems. And what we want to do now is just to create a new DDL source file

00:06:32 and create such a view. So just relax. This will not be a big feature party or anything. It will just show you how you create such a DDL source.

00:06:42 For that, or course, we select New. Then we don't find something which seems to be right and we choose Other.

00:06:51 And we remember the name DDL, and for that we find the DDL source here.

00:06:58 We click Next and we of course need to specify a name for this source file, and by chance we have something here already.

00:07:07 We give it a name, Simple CS View, for the description, and now we see what Jasmin has mentioned in the last slide—

00:07:18 that it is just included in a classical transport request which I will now create: open SAP and click on Finish, and that's all.

00:07:30 It is empty.

00:07:32 It's empty. It's a source file, and source files are classically empty.

00:07:38 This is because CDS is also a general concept. As an ABAP developer you are quite used to that we have kind of skeletons ready,

00:07:47 but in this case, don't panic. We have syntax help available because what do we want to do?

00:07:54 I would like to define a view.

00:07:56 Exactly. So for that we just try to type “D”. Ignore that one.

00:08:02 And we have defined view syntax help available.

00:08:05 Press the top key and we have the defined view and of course, a name would be very handy now for the view.

00:08:13 For that we choose ZCDS here. V for view, SIMPLE, and now, of course, we want to know from where do we want to select.

00:08:25 So, define view as select from. And now you might ask,

00:08:34 why is here the “from”? Now only I do my protection list because you will see when I create the projection that it's much more handy

00:08:42 and useful if you specify the table from where you select first, so here we use classical EPM table snwd,

00:08:52 sales order table. That's the right one. And now we do the protection list.

00:09:01 And now since you have already specified the table name, we can help you with the content

assist with the columns of that table.

00:09:09 I think sales order ID, there are primary keys, always a good idea, so we add the sales order ID,

00:09:19 we give it a name, order_id.

00:09:25 Gross amount is also kind of very useful to know of my sales order and if I have a gross amount,

00:09:34 currency code is always a good idea.

00:09:39 Basically I've now refined my view, but I still have this light bulb bubble here

00:09:47 telling me the SQL view is missing an annotation or catalog name. And that refers back to the Data Dictionary

00:09:55 that I have to specify the name of this view in the Data Dictionary. That it's always available there as any classical open view.

00:10:05 So and of course we help you here also, so we click on the light bubble and we have already a suggestion here. Let's just use that one and we are good to go.

00:10:15 Just activate the view and everything is done.

00:10:20 But of course, this is not so impressive because maybe we also want to know what's inside the view

00:10:27 and for that we have introduced a new data preview in the ABAP Development Tools here

00:10:33 and here are our DDL sources, just ignore those ones. You will get in touch with them later.

00:10:38 And right-click the view and choose Open Data Preview.

00:10:45 Here I have the result of this view and I can sort here, for example, by gross amount, different orders. I can have filters, number of entries, and so on.

00:10:56 Just play around with it on your trial system. This is really handy when it comes to developing those views.

00:11:03 Okay, but we can also consume it in an ABAP report. So we have prepared something for you.

00:11:13 This is exactly just a simple report defining the data type with the entity name of this view.

00:11:21 You remember the view name. The structure is already available in the Data Dictionary and we can do a SELECT *

00:11:29 from that entity name with open SQL into the defined data variable.

00:11:35 Yes, and you might ask, he is using SELECT *, typically they tell me I should not use that.

00:11:41 Yes, but in that case you have created the view

00:11:44 and you have already specified only the fields that you want to use, so a SELECT * here is no problem at all.

00:11:52 Just remember if you did not define the view, check if you need all of the fields or not. If not, specify those columns.

00:12:00 But that's basically all. You now know how to create a CDS view and we will get in touch with all of the new features in the next unit.

00:12:09 Thanks for demo and before we close this session, you might ask, hey I have now seen Open SQL and I have seen CDS,

00:12:20 so question is, are they competitors? And the answer is, not at all.

00:12:27 We have a couple of guidelines for you. Of course, one guideline is if you need a feature which is only available in Open SQL

00:12:36 or only available in CDS, you would have to choose one over the other.

00:12:40 However, there are more general guidelines, so if you really have only one query that needs to be executed once—

00:12:50 and once means in one specific coding part—use Open SQL.

00:12:55 And if you have a real reuse case, choose CDS over Open SQL.

00:13:02 Or if you need specific features or concepts which are only available in CDS, which you will hear about in the upcoming units,

00:13:12 you would of course, also use Core Data Services.

00:13:16 And with that I would say thank you and hand over to Jens for the conclusion.

00:13:22 Just a short recap. What have we learned today? Core Data Service is a bigger concept and only in the Application Server ABAP

00:13:31 but we have started adopting this Core Data Services with the first set, the ABAP view building,

00:13:37 because this is really what helps you most in developing cool applications on ABAP for SAP HANA

00:13:44 Next we will take a look at what is more inside those few building things.

00:13:50 Okay, thank you and good-bye. See you in the next unit. Bye!

WEEK 3, UNIT 4

00:00:12 Hello and welcome back to week 3, unit 4, Core Data Services: View Definition.

00:00:19 Let me outline the unit we are in.

00:00:23 In the last unit we've heard about Core Data Services. What that is in the general concept and what it means for ABAP developers.

00:00:33 In this unit, we will concentrate more on the features provided for you in order to follow the Code-to-Data paradigm.

00:00:41 So let me show you the outline of today's unit. We will start with the capability that the CDS view definition has, so the view definition features.

00:00:51 In particular, what you can do with the projection list, what aliases are, and then we will come to the view-on-view concept.

00:01:00 I don't want to say something now, just have a look at the features.

00:01:07 After the view-on-view concept, we will introduce the extension concept, how can you extend CDS views, and last but not least, we will come to CDS views with input parameters.

00:01:18 Please buckle up. This is going to be a long session because there are a lot of features provided for you to follow the Code-to-Data paradigm and to optimize your ABAP coding.

00:01:28 The good thing is, we will show it all live in the system so we won't bother you with any slides.

00:01:35 True. So let's jump directly to the system, open the CDS views we have already created for you, and show you what's in there for you.

00:01:45 Okay, we will start with projection list features. So in principle, they are very similar to the things that we have already seen in the Open SQL unit,

00:01:56 the second unit of this week. So if you would like to do a Code-to-Data paradigm, you would come up with aggregations.

00:02:06 For that, you have for, example, the SUM function to do an aggregation

00:02:11 here in that view on SO, and SO is our well-known sales order table.

00:02:18 In addition to that...no, sorry one step back. You would have an aggregation. We will of course have a GROUP BY statement,

00:02:28 so we just repeat the fields on what we have to aggregate, except for literals. I'll come to that in a second.

00:02:36 So I repeat the business partner ID, the company name, as well as the currency code here in the GROUP BY clause in order to have the summed gross amount.

00:02:48 In addition to that, we have the GROUP BY, we can have as well as in the Open SQL case, a HAVING clause.

00:02:56 And here again we can have, for example, a summed gross amount, the total gross amount, and we can, for example, filter on those sales orders which are above a certain value.

00:03:08 You have seen it already, in the view we can have literal values. You remember this 1+ 1 example in the Open SQL case

00:03:18 or integer literals, so again the 42. And like Jens has shown you in the previous unit, we can

have a data preview on that

00:03:28 in order to see what the result of a query on that CDS view would be.

00:03:35 So just go there and you see there the string literal called Literal or the integer literal 42.

00:03:41 We have company names, customer IDs, and total gross amount, the summed gross amount, of our business partner.

00:03:50 Again, filter would be possible and so on and so forth. We have already mentioned that in the previous unit too.

00:04:00 You can have a semantic information on the business partner ID here. In that case, I'm calling the key, or the key attribute,

00:04:09 and we can have alias names like your customer IDs or, for example, for the summed gross amount. We also have to give that an alias name.

00:04:19 Okay, those are features which you are used to also from Open SQL now. In addition to this basic features,

00:04:29 there are things like arithmetical function or cast expressions. I just opened that arithmetical function here, for example.

00:04:37 I am subtracting the net amount from the gross amount, giving me the tax amount. Or if I could, for example, multiply the gross amount

00:04:47 with a certain number here like the discount that I've typically used for the Open SQL examples.

00:04:55 And if I would like to do that I could, for example, cast a gross amount to an ABAP flow time variable. And again, give it an alias name.

00:05:04 And last but not least for basic features, I have here some conditional expressions. That was also part of the Open SQL unit.

00:05:15 We had as conditional expressions the simple CASE, the searched CASE and the COALESCE function

00:05:21 and all of that is also available here in the CDS view in the DDL sources.

00:05:29 So I can have a simple CASE, CASE on delivery status, delivery status of a sales order.

00:05:35 I can have the searched CASE where I have this case when, some value above a certain value that is a high volume sales order

00:05:44 and I can have the COALESCE function, if you remember that from Open SQL. This either gives me the first argument if it is not NULL,

00:05:53 really a NULL value—not initial value but NULL value—or here the Not yet invoiced string if no invoice header exists.

00:06:03 Okay, that was very, very fast. Please go to your trial system and try that out on your own.

00:06:11 Have a look at the documentation and you will find plenty of these new features. But coming to something very cool and new.

00:06:20 Yes, so let me take over this one and switch to another view. So the first new thing is:

00:06:28 we will start with the view which is not so cool. You'll remember maybe there is a just a

buyer_guid from the SIMPLE view

00:06:36 but the really cool thing in this is that you can now have views based on existing views.

00:06:42 So what does this mean? It is very easy. So let's switch to the second one. What you have here is define view on this base view,

00:06:52 so you can have stacks of views over each other and this really helps you creating complex calculations

00:07:00 and at the end, having it all pushed down to the database

00:07:05 because when you execute such view-on-view concepts, it's always as the database calculates it as one view in total,

00:07:14 with all the joins and it is just executed in one statement and therefore also optimized like that.

00:07:21 So that's very important. If you do view stacking, this in on compiled design-time level.

00:07:27 During run times this is all calculated to one big view in the most cases. There are some exceptions, but basically it's a better choice

00:07:37 then writing classical loop coding just to have view-on-view concepts.

00:07:43 Okay, so what we have here, just as an example, you see we have this base view,

00:07:50 we join it with a business partner on the base view buyer_guid on the node_key. And if I switch back to the base view which I can navigate

00:08:00 with Ctrl + left-click and you see I'm back on that view that we have already opened.

00:08:08 Then I add the gross amount and the currency code with the joined company name. What does this look like?

00:08:16 We can just execute it in the Data Preview and you see here you have the business partner ID, company name and also the gross amount and the currency code.

00:08:27 Okay, so just a simple example. This can get complex, but it really helps you creating reuse views and also complex calculations.

00:08:40 Another very cool feature and also very important for you as a customer or a partner is that you can extend CDS views.

00:08:49 So what does this mean? Let's go back to the base view and you see here this little sign here on the left,

00:08:57 and it tells you that this view is already extended with an extension and just by left-clicking, we can take a look at it.

00:09:05 Here you see, you don't write define view; you write extend view and name the base view

00:09:16 and then you give it an extension name and you just add those fields which you want to add.

00:09:22 For the current release you are working with (SP8), extensions are only available on the protection list

00:09:29 and only if the base view does not have an aggregation or GROUP BY statement

00:09:39 but these are features that are coming. We will show you that in the outlook at week 4.

00:09:44 Okay, do you think we should execute it?

00:09:48 Yeah, why not.

00:09:51 Oh, that is not an extension.

00:09:56 Of course, I already thought I made a mistake but of course extensions cannot be previewed. You should always preview the base view.

00:10:06 Yeah and here you have full fields on the base view. So in principle, you would think you would come up now with a data preview on those four columns.

00:10:16 Yep, but there are some more and by chance they are exactly the ones

00:10:24 we have extended with created_by billing status and so on. We see it here.

00:10:31 Exactly. And that is because the technique that is used here is just the append semantic,

00:10:36 which you are certainly familiar with when extending, for example, a Data Dictionary table.

00:10:43 Just an append. Okay, so much about views-on-views

00:10:52 and view extensions and now comes a new concept.

00:10:59 Up to this point, you are completely database-independent, I have to say,

00:11:05 so all of that will work on any SAP-supported database. And what I will show you now is the so-called input parameters.

00:11:14 We have provided for you...let me close all of the windows, in order not get into trouble with all these windows open.

00:11:22 We have provided a possibility for you to give input, scalar input parameters to a CDS view.

00:11:31 The syntax is just that you define the view as you are now familiar with and you say with parameters.

00:11:40 I have only one parameter, the customer name here. It could be several. You just have a comma-separated list of parameters

00:11:48 and this parameter is given—for example, when I consume it in another view or by Open SQL—are given to the input parameter can then, for example,

00:11:59 be used in the ON condition of joins here as a $parameters in the projection list. I can also have that in more complex function,

00:12:11 like, for example, in this CASE expression I can compare the company name of the business partner with my input parameter.

00:12:20 And of course, I can also use that in the WHERE clause.

00:12:25 I could now, in principle, show you that again with the open data preview; however, I would like to choose the other approach.

00:12:33 I have here a consumption view, so nothing else than just a view-on-view like the one Jens has just shown you.

00:12:42 So I'm using a SELECT FROM. As databases, I'm using the view with_input_parameter.

00:12:49 I named it like that in order to make it a bit more readable. And I pass the parameter.

00:12:54 Okay, so I will just use the consumption view to display that. It is not going to be very interesting

00:13:04 because the only thing that I'm consuming here is from that view. I use param_customer_name

00:13:12 and if I go back to the parameter, you see that is just the parameter repeated in the projection list.

00:13:21 So I will see a lot of SAP entries and as many as we have in the original databases as entries.

00:13:32 So let me just consume that, not that interesting. You see I'm just passing the parameter and get that as a column.

00:13:43 Okay, so that was view-on-view consuming the input parameter.

00:13:54 Of course, you are interested also in how can I consume that with Open SQL. For that we have prepared again a small little report.

00:14:02 In the consumption here, let me enlarge that a bit. I mentioned already that input parameter is database-dependent

00:14:12 so it really depends on which database system you're executing that on.

00:14:18 So in principle, I would just write the SELECT FROM the view with input_parameter

00:14:24 and it's a very similar syntax like the one that you have seen in the CDS view.

00:14:28 But in order to have that running only on one system, I want of course to check that.

00:14:35 And for that, I would get a syntax warning that this is only available on certain databases

00:14:44 and not only on SAP HANA, but on certain databases. That is a pragma removing the syntax warning for me

00:14:54 but of course if I remove that with a pragma I also have to do something.

00:14:58 For that, the ABAP database feature class provides you with certain help functions.

00:15:08 So I have to say which feature I would like to use in that case. I would like to use the views with parameter feature.

00:15:15 And I can ask the cl_abap_dbfeatures class if that functionality is available.

00:15:23 So here is the use_features. I can even do a double-click on that so it imports the connection.

00:15:32 I can have it using the standard and which feature I would like to request. And that returns me if the feature is supported or not.

00:15:40 In my case, of course, it is supported. I have here the information given back from that class imported into the this lv_feature_supported.

00:15:50 I check that in an IF...ELSE construct and if the feature is supported I can perform the SELECT.

00:15:57 Otherwise I can do some alternative coding here or some meaningful fallback solution.

00:16:05 Okay, so a lot of new features provided for you with CDS

00:16:11 and with that I would like to hand over for Jens for the key takeaways in order to show you what's really necessary to take with you.

00:16:21 Okay, just to sum it up in what you have just seen in those demo example CDS views.

00:16:28 It's important for you that we have a rich feature set to follow the Code-to-Data paradigm

available in our CDS views.

00:16:38 And those CDS views, as I've just shown you, can be stacked into different views so that you can consume a created view.

00:16:46 So you can create, for example, a data model for yourself.

00:16:51 By the way, it is also important for you to know we are of course using those CDS views in our SAP Business Suite already

00:16:58 and will probably be available soon to you.

00:17:01 And of course, what is very important, and we know this, you can extend those views to add fields you need to have on your UI or your program code

00:17:15 And of course, you can have input parameters, of course, scalar input parameters

00:17:21 and keep in mind to wrap those with an IF around and check the function, if this function is supported by your database.

00:17:30 And if you can imagine, there are some more to come up in the next service releases of SAP NetWeaver 7.4.

00:17:36 Okay, so thank you very much and see you in the next week.

00:17:45 No, next unit, we are talking about. The week is not yet over, there is one more unit to come

00:17:52 and we will talking about Core Data Services, what associations are, and what's in there for you.

00:17:59 See you in the next unit. Bye-bye.

WEEK 3, UNIT 5

00:00:12 Welcome to the last unit of week 3, Core Data Services: Associations.

00:00:18 Hi there. In this unit we will show you how you can use, define, and, of course, consume associations in the Core Data Services.

00:00:29 But before we start with the concept of those associations, let's first take a look at union and join support in CDS views.

00:00:38 Thank you Jens. Let me directly jump to the system. We have created a small Core Data Services ZCDS view for you

00:00:47 and that one has a union inside. So what does it do? I have a SELECT statement.

00:00:56 I'm selecting from the sales order table. I have a projection list and here in the projection list,

00:01:03 I have the HAVING, no sorry, that is not in the projection list. In that SELECT statement, I have a HAVING clause.

00:01:11 I'm aggregating thesales order gross amounts and if it's below a certain value, I put as a category the 'small' inside here as a string literal.

00:01:20 I will union, that SELECT statement with another SELECT statement. That means I'm just combining the result set of those two SELECT statements.

00:01:31 and here the SELECT statement below is on the same table. That is not a mandatory prerequisite, but pretty handy for my example.

00:01:41 In that case, I have the same SELECT statement except that the HAVING clause is above a certain value.

00:01:48 And if it's above a certain value of the total gross amount, I put the category as 'large'.

00:01:54 So there are two types of union statements, UNION and UNION ALL, and they differ only in the distinct semantic.

00:02:02 So UNION applies the distinct semantic on the result set of these two JOIN statements,

00:02:07 while UNION ALL just combines the result sets of the two unions, of the two SELECT statements.

00:02:14 The prerequisites for UNION and UNION ALL is that they have the same amount of columns, same number of columns, and they have to have compatible types.

00:02:24 As we have the business partners having low and high amount of sales order gross amounts,

00:02:33 I expect there will low and high, large and small, category sales gross amounts.

00:02:39 Just have a look at the data preview and you can see here that is the case.

00:02:44 So I have those having categories small, if that is below the certain value, and large for the others.

00:02:50 So much about union. Let me know come to join support in CDS.

00:02:57 We have already had some examples using JOIN statements and now I would like to say a couple of words on what is there.

00:03:06 So for Open SQL we have seen there are INNER JOINS, LEFT OUTER JOINS, and RIGHT

OUTER JOINS, and the same things are available in CDS views.

00:03:17 You could again have, in the ON condition, comparison values here. In that example, I only have the comparison with the equals (=) sign.

00:03:27 Here in that example—and I would like to say a bit more about that example because it will be reused when we are talking about association, in a second—

00:03:36 so I'm doing a JOIN between a sales order in the business partner table. That is an INNER JOIN because each sales order should have a corresponding business partner.

00:03:46 I could additionally have a LEFT OUTER JOIN between the sales order and the sales order invoice header, for example,

00:03:54 if the sales order has not yet been invoiced, there will not be an invoice, an invoice header, so LEFT OUTER JOIN is the join of choice here.

00:04:04 Okay, with that I would like to hand back to Jens to talk about associations now.

00:04:12 Okay, what are associations? So, let's just think about a data model which you have here,

00:04:21 our example model, sales order, business partner, invoice, product. And every time you want to consume those artifacts

00:04:29 you have to create the JOIN statements. In our example, it's quite easy also to write the ON condition always

00:04:37 because it is not key to some other GUID identifier,

00:04:46 but this is hard work if you think about the Business Suite models with the MARA table,

00:04:52 where you have quite huge ON conditions on the JOIN statement.

00:04:56 Now how would it be if you could just write instead of all the time the whole join condition, just an alias association.

00:05:05 So, for example, sales order to business partner. Let's call it “to business partner”, so this would be quite easy.

00:05:15 And that's exactly what associations are all about.

00:05:19 So, basically they replace the join condition with a simple path expression and that's all.

00:05:27 If you say that's kind of the whole concept of the association, but it brings you so much benefit and for example, it makes it really easy

00:05:36 to consume data models later on. But basically, let's just jump into the system and see how you can define associations

00:05:46 and also how you can consume those associations. Okay.

00:05:53 Let's go to the example. And remember this view here with the two join conditions to the business partner and to the invoice header from the sales order.

00:06:03 This is an example where we have already created an association, replacing the LEFT OUTER JOIN to the invoice header

00:06:13 It's the 021 association, as Jasmin has just stated, replacing the LEFT OUTER JOIN.

00:06:19 There can still be a sales order without an invoice, but hopefully for not a long time

00:06:26 And you have the INNER JOIN here to the business partner and this is exactly the alias that I've been talking about in the example beforehand

00:06:35 with to business partner or business partners. Here it's invoice headers and this association here is used to get the payment status, the currency code,

00:06:45 and of course, used also in the GROUP BY or HAVING statement. and now we could also refactor this to an association.

00:06:55 For that I just write association. Syntax help is available. I define the cardinality,

00:07:04 that's also a feature available with associations, that you can say I I know the cardinality and I can make it,

00:07:14 enrich the meta model of the CDS view, that's exactly what we have talked about in unit 3 of this week,

00:07:22 that you enhance SQL with new concepts and that's all in CDS.

00:07:28 Now I just need the 2 and that's basically all. I can reuse this ON condition.

00:07:35 And this business_partners association, that's the name now, is reused here and the view is fully functional.

00:07:44 So that's the way how you could define an association.

00:07:48 But there is still more to come because this one could also be written with a JOIN statement and you don't gain much of this model.

00:07:58 But Jasmin will show more on that.

00:08:01 Yes, thank you Jens for showing us how to define and also already how to consume an association

00:08:08 Let me give you a bit more insight about association types that are available.

00:08:14 In principle, all you had in here are so-called ad-hoc associations.

00:08:19 I have created a small CDS view showing you the difference between ad-hoc associations,

00:08:25 the one that you have already seen right now, and so-called exposed associations.

00:08:30 We had the sales order to sales order invoice header association and if you directly consume the association defined in a view

00:08:40 in the projection list, like the one here, you follow the path to the payment status in the invoice header.

00:08:47 You have an ad-hoc association. And important to know here is that using this path and following this path

00:08:56 really constitutes a join on the database. So you really execute the JOIN statement.

00:09:03 On the other hand side, we have exposed association. The prerequisite for an exposed association in order to,

00:09:12 for example ,consume the association in a view on that view, in a view-on-view scenario,

00:09:19 you would expose it just giving the association for further usage. So here I have the association business partner and I would like to expose it.

00:09:29 The prerequisite in order to do so is that I have the field used in the ON condition,

00:09:35 so the buyer_guid here in the projection list, a so-called managed association. And that exposing of the association does not constitute a join.

00:09:47 We call it the join-on-demand mechanism, so only those where you really write down the path expression,

00:09:54 follow the path expression is executed or constitutes a join on the database.

00:10:00 Okay, I have exposed it now, so the question is, how can I consume it? I mentioned already that you can consume it in another view.

00:10:09 So on that zcdsv_assoc_types, I put a view on that view, a second layer. I have created that already.

00:10:19 I have prepared the define view, the consumption now on that base view.

00:10:25 So what I can do now is use auto-completion.

00:10:29 I can use the association which is exposed to business partner, given here, so association,

00:10:38 and I can follow that the path, for example, to business partner ID.

00:10:46 Of course, I could have already shown you the consumption with the the Open Data Preview.

00:10:53 I will now, not use the Consumption view but back to the association definition.

00:11:00 So let me open that with the Open Data Preview and selecting, for example, one order, arbitrarily, I can right-click

00:11:10 and I can follow the association, for example here now, to business partner. And in principle, there could not only be only one.

00:11:17 There could be various, really depending on how complex your model is. But as you have seen, I don't have to write down any ON conditions.

00:11:27 I don't have to write very complex things. I'm just using this very simple human-readable path expression.

00:11:36 That's where the power of the associations really starts to shine. So just a short example, if you have a bigger model it gets more and more handy.

00:11:45 Just try it out.

00:11:47 Yes, and one last word before I hand over to Jens.

00:11:53 There are of course more things in there. For example, you can filter on the associations. You can have 1 to n associations.

00:12:02 You can filter the cardinality if you have a certain filter condition, and so on and so forth. But we would like to really stick to the basic concept, to bring you the concept,

00:12:12 we will have that as additional material. Just have a look at your trial system. There are various examples in there. Also, in the additional material.

00:12:24 Thank you Jasmin. So, let me just do a short recap of what you have just learned

00:12:29 because as we have stated, we want to let you get the concept of the association.

00:12:34 And the basic idea, as mentioned in the first slide of the associations, it's really replacing joins

with simple path expressions. That's it.

00:12:44 It's then an easy-to-consume model which you have available and don't need to write the joins.

00:12:51 You have easy refactor and you have quite a good reuse views, basic views where you can build up your UI models. for example.

00:13:00 And of course, the join-on-demand as Jasmin has stated, those path expressions are only materialized when you use them. That's all.

00:13:10 So the only thing left is now: What have we learned in this week?

00:13:17 Yeah, thank you Jens. So we started the week, or everything inside this week, was about optimizing your ABAP coding

00:13:27 especially when working with SAP HANA.

00:13:30 And in order to do so or in order to follow the Code-to-Data paradigm,

00:13:36 we have shown you what Open SQL is giving you as tools at hand,

00:13:47 how we enhanced Open SQL, what the features are, and so on so forth.

00:03:53 And besides Open SQL, we have also now the Core Data Services. You have learned what Core Data Services is in general.

00:14:03 You have also learned what's in there for ABAP developers.

00:14:07 You have seen the long list of new features, of course, because the whole concept is new, of features provided to you in terms of

00:14:16 CDS view definition capabilities, and last but not least, in this unit we have heard about associations and how you can use them.

00:14:26 So we will now leave the path of optimizing and we will start to dig deeper into the HANA business.

00:14:34 So we will see what you can do if you really need to use natively the SAP HANA features

00:14:42 and how to consume and use that as ABAP developers.

00:14:46 We will start with that in the next week, in unit 1, with the native SAP HANA usage in ABAP.

00:14:54 See you next week.

00:14:55 Stay tuned, good-bye.

00:14:57 And last but not least good luck for your weekly assignment!

www.sap.com