Agenda

In this session, we will talk about SQLcl, which is a new tool developed by the SQL Developer team at Oracle Database Development Tools Team. The purpose of this session is to introduce this great tool to you. Even though it is still in its early adopter version, it has some great new features that I thought developers and DBAs ought to know about so feel free to raise your hands if you have any questions during the presentation.

- We will talk about the old SQLPlus and what makes it so great.
- I will show you how to install SQLcl and some basic functionality.
- We will talk about some of the new features – there are a lot of them, so I picked only the ones I thought will be really-really cool and have some live demos of them
- Feel free to raise your hand with questions during the session but I promise we WILL also have time for Q&A in the end of the session.

[Audience interaction] Let me start with a question: please raise your hand if you are using SQLPlus. Is there anyone here who does not use it? I did not think so – and I do not think anybody here will find it as a great surprise. SQLPlus is one of oracle oldest tools. It has been around for at least 30 years – it came out in Oracle database version 5, which came out in 1985. Before that, Oracle command line tool was called UFI ("User Friendly Interface") which was anything but user friendly. That tool was in use in Oracle database version 2 through 4 (1979 to 1985).

The SQLPlus might sound like a simple tool but it is actually not. SQLPlus is one of the more complicated tools Oracle puts in their client package. It is the first tool to be updated when syntax changes, it interacts with the grid infrastructure tools seamlessly (for example – if you shut down a RAC instance using SQLPlus, the grid infrastructure will be aware this was intentional and will not try to restart the instance) .

As you are about to see soon, this tool looks very similar to the old SQLPlus. Most of the old features we duplicated but let me put it up front: there is no guarantee (as of now) that it will ever replace the SQLPlus completely it in the future. The guys at the development team made sure I state it, so no one at the SQLPlus development team get offended or something. Now, on with the show: let's start exploring the SQLcl.

Over the years, the guys in Oracle tried to add some new features to the SQLPlus. They added support to markup language so we could output our queries as html reports – some of the internal tools such as AWR and ASH are using it. They tried to introduce windows based SQLPlus called SQLPlusw and even had a web based version called iSQLPlus.

Oracle also recognized that SQLPlus is been used a lot of scripting and maintenance (they are using it too, themselves) so they also provided a lot of support of such things. For example the use of variables, prompt command, and user challenge like accept and so on.

As of Oracle 12cR1, SQLPlus is the only command line tool Oracle provides for SQL queries and database maintenance.

Over the years, this tool was updated a lot but we still used an old tool with not a lot of new features. The tool did get some bugs fixed and some new syntax updates but that was about it – there was little to no new functionality introduced to could make life easier on the end users.

About a year ago, the development team for SQL Developer, which is the not-so-new GUI interface that oracle provide free-of-charge to its customers, decided that they want to cover one of the areas that the GUI tool did not cover at all. That area was the command line interface area and automatic scripting. They started rewriting the functionality provided by SQLPlus into Java code and added some cool new features. Their goal was to make the new tool as similar to the SQLPlus as possible but still introduce some things that users had been complaining about for the last 30 years. The really exciting thing here is the team members are really open for suggestions. I know of at least couple of features that development team added to SQLcl just because someone twitted it to them, it sounded cool and not hard to implement…

Again, I would like to remind you that we are talking about an early adopter version. That means that the feature list is not final, some of the features are not fully implemented, and there are still some problems and bugs around the code so keep this in mind when you play around with the tool.

Okay – enough with the chitchat, let's get our hands dirty.

Installation

The first thing you will need to do in order to use SQLcl is download it from the SQL Developer site. You can find it under the SQL Developer 4.1 download page, at the bottom. It will be under SQLcl Early Adopter download link. While preparing for this session and building my presentation, there were at least three or four new versions that came out, so keep your eyes open for new releases – at least until they release a stable production version. If I understand correctly, that version is planned to be shipped out with Oracle 12cR2 which will be released sometime during 2016.

The actual footprint on the SQLcl tool is really small – it's only 11 or 12 Megabytes in size.

Like the SQL Developer, this is a Java based tool and it requires Java 1.7 or 1.8 to run. Unlike SQL Developer, it requires only the Java runtime environment (JRE) and not the Java development kit (JDK). SQLcl does not require anything else – for example, it does not require an Oracle Client at all but it does recognize it if you have one, and give you some extra features like thick client support. Since it Java based, it is not platform specific and can run on Windows, Linux or OS X but I didn't try it on anything else.

The installation process is very short and simple: download, unzip, and run. If you're using Linux you might need to change the sql file with the +x (execute flag) for making it an executable but that is about it. We are ready to run the SQLcl and use it.

Connection

Let's see how it looks when we're connecting for the first time. we said we don't need an Oracle Client so we will be using the EZConnect JDBC style connection string. I am using my local database for this presentation so we'll use the local IP address 192.168.56.101, my listener's port of 1521 and the database name for the connection.

The first new and cool feature I'd like to show you now is that I don't have to know where I want to connect. Let's say I want to connect to a database but I am unsure of its TNS connection name. I write connect oded@ and then the TAB key and I get a list of all available connection. The list is derived from my tnsnames.ora file and recent connections. If you remember a partial name of the tnsnames it will autocomplete it.

Once we're connected we can see the jdbc connection information by using the "show jdbc" command.

Intellisense

As you can see – one of the things they added is intellisense. Now we are connected to the database we can use it for our basic queries: select star from dual. Easy enough but we would like to query from our tables – how can we find now can use the auto complete again: sel [tab] * f [tab] [tab] [tab]. We see a list of tables we can query from so we will start the name and hit the [tab] key for intellisense. Did you notice that I could go back using the arrow keys back in the query? You can use the [CTRL] + [W] and [CTRL] + [S] to jump to the beginning and the end of the query. I can even go back to stand on the star to get a list of the columns in the query.

History

Okay, now we would like to go back to one of our previous queries or commands. We can go back to those queries using the [up] and [down] arrow keys. Once we find the command, we can edit it or use the [CTRL] + [S] keys to jump to the end run it. We can also print a full list of commands that we ran – this is not session persistent – we can even find command from a previous sessions of the tools so keep that in mind when you're using it to change passwords. We can use the "history full", "history script" and "history usage" to see the number of times a command has ran. We can use the history [number] to choose to load the command from the history to our command buffer.

DESC, INFO and INFO+

Now we know which table we want to query, but we're not really sure what is the table structure. We can use the good old describe command for that, but sometimes it is just not enough. When using the describe command, we can't tell if there are indexes on the table, what constraint the table follows - primary or foreign constraints, or what are the columns' default values. When using SQLPlus, this would be the time to start querying around the data dictionary to find all the relevant information. When using SQLcl all we need to do is run the information command (info for short) and we get a detailed screen with all relevant information. Here we can see the table columns, indexes, primary key, foreign key constraint, and default values. Pretty cool, right?

Even then, that is sometimes not enough. What if we want to know more details about the table statistics? And what about the column statistics and histograms? we can use the info+ command where we can see all that information.

Now we know the structure, we can return to that query and select the correct column. We can even use the intellisense to find the correct column name: s [tab] * f [tab] [tab] [tab] go back to the star (*) and hit [tab] – we get a list of column names to choose from.

Any questions so far?

Repeat

Let's continue to the next small and cool feature. Let's say you need to query a table over and over so see when it changes. For example, checking the status of the data guard of if you want to see if there are new user blocking locks and so on. Until now, we had to run it manually every few seconds. In other cases, we built scripts that will loop the query but we didn't have any good way to do it easily. Now we do. Once the query is in the buffer, we can use the repeat command to run it again and again. The repeat command runs the query from the buffer and accepts two parameters: The number of times we want to run the query and the interval in which we will ran it at. For example, if we want to run a query 5 times with 3 seconds interval we will use the "repeat 5 3". As I said, this is really useful if you're impatient as I am…

Jeff Smith from the development team had a great example for implementing this feature. He used it on Oracle 12c to run a live tail command on his alert log using a query. It looked something like this:

http://www.thatjeffsmith.com/archive/2015/04/tailing-the-alert-log-with-sqlcl/

DDL

One of the disadvantages of being a consultant is that sometimes you get to a customer who doesn't use any graphical tools and we have to get by with what we have. This is why low footprint tools are so important to us. When working with these customers, we sometimes need to extract a structure of an object – and that is a really hard work. Most of the DBAs I know use GUI tools to do it, but as a consultant, we don't always have this luxury.

SQLcl solves this problem by introducing a new short code command – the "ddl" command. This will extract the structure of an object in a single, simple command: ddl myobject to get a valid and formatted ddl command. This will work with most objects: table, view, synonym, sequence and even an index.

This feature uses the dbms_metadata package so we can be sure the command is valid but it also reformats it so we can be sure the command is readable.

Yes, I know that we can use the dbms_metadata manually, but for it's just hard work: we need to identify the type of the object, remember the syntax and write such a long command for each object… Even then, the formatting sometimes messes up and it gets the output all wrong and then we will need to go over the extracted code just to figure if it is a valid SQL and fix it if not.

Here is cool tip for you: since the ddl command uses the dbms_metadata.get_ddl command, we can use the dbms_metadata.session_transform command to modify its output. Let us say we extracted the ddl of a table, but we see that it extracted the segment attribution as well (the tablespace, size etc.) and we don't need it. We can modify it we using:

exec dbms_metadata.set_transform_param(dbms_metadata.session_transform, 'SEGMENT_ATTRIBUTES', false);

Now when we run the ddl command, it will not output the segment attributes. This is a really useful command, isn't it? The problem is that this is a connection specific. The next time we connect, we will use this setting, even if we did not close our SQLcl instance. How can we save this command for the next time we will need it?

Alias

Ah ha! SQLcl comes with an alias feature. The alias feature allows us, just like in unix or linux operating system, to name a command with an alias and use that when we want to run it. For example – if we create a new alias ddl_no_seg

> **alias** ddl_no_seg=exec dbms_metadata.set_transform_param(dbms_metadata.session_transform, 'SEGMENT_ATTRIBUTES', false);

And use the alias "ddl_no_seg" before using the ddl command. This will run the dbms_metadata.set_transform_param and we will get our neatly formatted create command. We can use the alias command to run simple scripts as well – for example, getting the size of the objects in my schema (tables2) or even send parameters to that alias: (locks zohar).

To see the list of aliases use the alias command. To see what it actually running use the alias list command.

ctas

Let's say we want to create a table as select but want to keep the original structure of the table we select. For example, if we want it to have the same partition structure or prmary key. In that case, we will need to extract a ddl, fix whatever need fixing and then run the command. SQLcl helps us with the ctas command: ctas old_table new_table will generate a command that builds the table with the same structure of the queried table and the select star from that table.

Bug: when using 'SEGMENT_ATTRIBUTES' false command generated is invalid.

Bug: when using ctas, the primary key name is not modified so it might fail when running the generated script.

SQLPATH and CD

The next thins is a little bit hard to explain but once you get the idea, you will wonder how we are getting around without it… Most of the database power users I know use command line interfaces and have a large directory with all the scripts they collected over the years. When they need to do something, they just pull up a script and run it. SQLcl, just like SQLPlus, allows us to run external scripts so no problem there.

So, what is the problem? The problem is that in SQLPlus, when running a script from a different directory than the one you started in, require us to write the full path for that script. Not a lot of people knows that we can set an environment variable called SQLPATH and that will add our directory to the search path of script. Now, if we want to change our running directory – with SQLPlus, we cannot. This is useful when running scripts for a specific version or if we want to change our directory to the rdbms/admin directory under the oracle home.

With SQLcl there is new feature which allow us to see what our sqlpath is during runtime: show sqlpath and modify our current script running directory: cd.

For example – if we want to run script in our sqlpath, we can just do it: @1 but when running a script outside of our path, we need it to be where we ran our command from.

With the CD command, we can change our current running location.

You should note that the search order is:

1. Current cd directory
2. Current running directory
3. The rest of the sqlpath

Bridge

This is kind of a weird feature but I'm sure most of us were waiting for it for ages. Let's say you need to transfer data between two databases. Most of us would use a database link right? However, what if we can't create a database link – for example, if we don't have permission or don't have a direct connection?

In SQLcl we have a "bridge" command. We can use our client that can connect to the two databases to create a temporary bridge between the tables. It is mainly used to script data move between two connections but we can use it for other things too. In order to create the bridge we use jdbc connection to the remote database and just query the data.

The following functionality is available

1. query tables in other connections
2. query tables in multiple connections in the same statement
3. insert data from one connection into another
4. create a table and insert data into it from another connection

Pretty output

The last feature I am going to present here today is one of my favorite and one of the main reason I really got to like this tool.

One of the main problems with the output that the sqlplus generate is that the original idea behind sqlplus was that we would like to see tables. Databases use tables and query generates tables so outputting tables sounds like a reasonable idea, right?

Well, no. Over the years, we found out that text based tables are great to explore data but sometimes we need more: we want to generate html tables, we want to generate insert command and sometimes we even want to output our data to other system. We sometime need out data in a csv, fixed or loader format. We sometimes want to output the data as XML or JSON… I guess this is something you're familiar with?

Some GUI tools gives us some solutions for those problems – but for some reason, the command line didn't. When we needed to output our data differently, we need to manipulate the standard output so it will fit our needs. We need to concatenate commas or command parts and this is really hard work for something that should be really simple.

SQLcl solves this problem for us. SQLcl introduces multiple ways we can output our data. Using the SET SQLFORMAT command we can set our output to be fixed size, sql loader format or even csv. We can generate xml and json outputs very simply.

We can even generate insert command if we want to – and all in a single and simple command (just watch out – the insert command is case sensitive).

Let's see some examples.

Set sqlformat csv

Select * from dwh;

Now, if we want to get it as insert commands:

Set sqlformat insert

Select * from dwh;

What about generating the data as JSON or XML?

Set sqlformat JSON

Select * from dwh;

Even the HTML output is different: the output is responsive, has smartphone and ipad support (small screens and landscape view) and so on. Just note that the old markup doesn't work yet.

The last thing in the pretty output is the ansiconsole format. The ansiconsole is useful if you want to use color in your output. This also allows us to use emoji (the small icons on you iphone and smartphone) but it has some limited support with windows 7 and linux (it works great on OS X and Windows 8.1).

Load

When we said we can output the data as csv, but what if we want to load data into our tablespace from CSV (excel files for example)?

We can do that as well: use the LOAD command with the table name and csv file it there you have it. In order for this to work, we need to make sure the file format is exactly as the format the sqlformat csv output it: the first line should be the list of columns and the data should be comma-delimited. It is autocommited every 50 records but this is something we can change if we need to.

Closing

We reached the end of the session. We talked about SQLcl, its new features and demonstrated a lot of them. At this point, I would like to remind you that this is a tool which is still in development. What we saw here today is an early adopter version, which will probably change in the future.

I would like to thank you again for inviting us. If there are any more questions of demos, feel free to contact me – my twitter, blog and email are on the screen.

Things I know still have bugs in

- You can use the host command to go to command shell yet. You can run single commands (host pwd)
- The DDL commands does not have terminator by defaults. Use this to work around:

Exec dbms_metadata.set_transform_param
(DBMS_METADATA.SESSION_TRANSFORM, 'SQLTERMINATOR', TRUE);

- When ctas when using 'SEGMENT_ATTRIBUTES' false the command generated is invalid.