*Edson Sanchez* ITI-3400 Database Programming

**export\_JL**

CREATE OR REPLACE PROCEDURE export\_jl\_sp (

file\_name VARCHAR2 -- user creates file name

)

AUTHID current\_user

AS

jl utl\_file.file\_type;

/\* CURSOR TO GET DATA FROM ALL ROWS \*/

CURSOR cur\_cust IS SELECT

lastname,

firstname,

address,

city,

state,

zip,

email

FROM

customers;

/\* LOCAL VARIABLES \*/

lv\_line\_txt VARCHAR2(200);

lv\_line\_cnt NUMBER(5) := 0;

BEGIN

/\* CREATING AND OPENING FILE TO WRITE\*/

jl := utl\_file.fopen('DATA\_PUMP\_DIR',file\_name,'W');

/\* USING IF STATMENT FOR ERROR CHECK\*/

IF

utl\_file.is\_open(jl)

THEN

/\* USING CURSOR TO LOOK THROUGH ALL ROWS\*/

FOR rec\_cust IN cur\_cust LOOP

/\* PUTTING ROW DATA INTO VARIABLE\*/

lv\_line\_txt := rec\_cust.firstname

|| ', '

|| rec\_cust.lastname

|| ', '

|| rec\_cust.address

|| ', '

|| rec\_cust.city

|| ', '

|| rec\_cust.state

|| ', '

|| rec\_cust.zip

|| ', '

|| rec\_cust.email;

/\* WRITING VARIABLE ONTO FILE\*/

utl\_file.put\_line(jl,lv\_line\_txt);

/\* COUNTER FOR LINES WRITTEN\*/

lv\_line\_cnt := lv\_line\_cnt + 1;

END LOOP;

/\* NO ERRORS\*/

utl\_file.fclose(jl);

dbms\_output.put\_line('Processed '

|| lv\_line\_cnt

|| ' lines.');

ELSE

dbms\_output.put\_line('Error opening file.');

END IF;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Error SQL exception');

dbms\_output.put\_line(sqlerrm);

END;

**export\_BB**

CREATE OR REPLACE PROCEDURE export\_bb\_sp (

file\_name IN VARCHAR2 –- user creates file name

)

AUTHID CURRENT\_USER

AS

bb utl\_file.file\_type;

/\* CURSOR TO GET DATA FROM ALL ROWS \*/

CURSOR cur\_shopper IS SELECT

firstname,

lastname,

address,

city,

state,

zipcode,

email

FROM

bb\_shopper;

/\* LOCAL VARIABLES \*/

lv\_line\_txt VARCHAR2(200);

lv\_line\_cnt NUMBER(5) := 0;

BEGIN

/\* CREATING AND OPENING FILE TO WRITE\*/

bb := utl\_file.fopen('DATA\_PUMP\_DIR',file\_name,'W');

/\* USING IF STATMENT FOR ERROR CHECK\*/

IF

utl\_file.is\_open(bb)

THEN

/\* USING CURSOR TO LOOK THROUGH ALL ROWS\*/

FOR rec\_shopper IN cur\_shopper LOOP

/\* PUTTING ROW DATA INTO VARIABLE\*/

lv\_line\_txt := rec\_shopper.firstname

|| ', '

|| rec\_shopper.lastname

|| ', '

|| rec\_shopper.address

|| ', '

|| rec\_shopper.city

|| ', '

|| rec\_shopper.state

|| ', '

|| rec\_shopper.zipcode

|| ', '

|| rec\_shopper.email;

/\* WRITING VARIABLE ONTO FILE\*/

utl\_file.put\_line(bb,lv\_line\_txt);

/\* COUNTER FOR LINES WRITTEN\*/

lv\_line\_cnt := lv\_line\_cnt + 1;

END LOOP;

/\* NO ERRORS\*/

utl\_file.fclose(bb);

dbms\_output.put\_line('Processed '

|| lv\_line\_cnt

|| ' lines.');

ELSE

dbms\_output.put\_line('Error opening file.');

END IF;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Error SQL exception');

dbms\_output.put\_line(sqlerrm);

END;

**import\_JL**

CREATE OR REPLACE PROCEDURE import\_jl\_sp (

file\_name VARCHAR2 -- variable to insert is the file name from BB export procedure

)

AUTHID current\_user

AS

fh utl\_file.file\_type; --file header

ftxt VARCHAR2(2000); --variable to hold file text

fname VARCHAR2(15);

lname VARCHAR2(20);

address VARCHAR2(40);

city VARCHAR2(20);

state CHAR(2);

zipcode VARCHAR2(15);

email VARCHAR2(30);

lv\_line\_cnt NUMBER(5) := 0;

BEGIN

fh := utl\_file.fopen('DATA\_PUMP\_DIR',file\_name,'R'); --opening file to read

IF

utl\_file.is\_open(fh) --making sure the file opens correctly

THEN

LOOP --loop to get each row

BEGIN

utl\_file.get\_line(fh,ftxt); --putting the line into the text variable to then parse

/\*Using SUBSTR() and INSTR() functions to separate values\*/

SELECT

substr(ftxt/\*text file\*/,1/\*starting position\*/,instr(ftxt,',',1,1) - 1/\*Length\*/),--fname

substr(ftxt/\*tect file\*/,instr(ftxt,',',1,1) + 1/\*starting position\*/,instr(ftxt,',',1,2) - instr(ftxt,',',1,1) - 1/\*Length\*/),--lname

substr(ftxt/\*text file\*/,instr(ftxt,',',1,2) + 1/\*starting position\*/,instr(ftxt,',',1,3) - instr(ftxt,',',1,2) - 1/\*Length\*/),--address

substr(ftxt/\*text file\*/,instr(ftxt,',',1,3) + 1/\*starting position\*/,instr(ftxt,',',1,4) - instr(ftxt,',',1,3) - 1/\*Length\*/),--city

substr(ftxt/\*text file\*/,instr(ftxt,',',1,4) + 1/\*starting position\*/,instr(ftxt,',',1,5) - instr(ftxt,',',1,4) - 1/\*Length\*/),--state

substr(ftxt/\*text file\*/,instr(ftxt,',',1,5) + 1/\*starting position\*/,instr(ftxt,',',1,6) - instr(ftxt,',',1,5) - 1/\*Length\*/),--zipcode

substr(ftxt/\*text file\*/,instr(ftxt,',',1,6) + 1/\*starting position\*/)--email, no length because it goes to the end

INTO

fname,

lname,

address,

city,

state,

zipcode,

email

FROM

dual;

EXCEPTION

WHEN no\_data\_found THEN --exiting loop when there are no more rows

EXIT;

END;

INSERT INTO customers (

customer#,

firstname,

lastname,

address,

city,

state,

zip,

email

) VALUES (

customers\_customer#\_seq.NEXTVAL,

fname,

lname,

address,

city,

state,

zipcode,

email

);

lv\_line\_cnt := lv\_line\_cnt + 1; --keeping track of how many lines

END LOOP;

COMMIT;

dbms\_output.put\_line('Number of rows inserted: ' || lv\_line\_cnt); --displaying how many rows were inserted

ELSE

dbms\_output.put\_line('Error opening file.'); --if the file was not opened correctly

END IF;

utl\_file.fclose(fh); -- closing file

EXCEPTION --catching any other errors

WHEN OTHERS THEN

dbms\_output.put\_line('Error SQL exception');

dbms\_output.put\_line(sqlerrm);

END;

**import\_BB**

CREATE OR REPLACE PROCEDURE import\_bb\_sp (

file\_name VARCHAR2 -- variable to insert is the file name from JL export procedure

)

AUTHID current\_user

AS

fh utl\_file.file\_type; -- File Header

ftxt VARCHAR2(2000); -- variable to hold file text

fname VARCHAR2(15);

lname VARCHAR2(20);

address VARCHAR2(40);

city VARCHAR2(20);

state CHAR(2);

zipcode VARCHAR2(15);

email VARCHAR2(30);

lv\_line\_cnt NUMBER(5) := 0;

BEGIN

fh := utl\_file.fopen('DATA\_PUMP\_DIR',file\_name,'R'); -- opening the file to read

IF

utl\_file.is\_open(fh) --checking to see that it opened correctly

THEN

LOOP -- loop to get each line in the file

BEGIN

utl\_file.get\_line(fh,ftxt); -- putting the line into the variable to then parse

/\*Using the SUBSTR() and INSTR() functions to separate the values\*/

SELECT

substr(ftxt/\*text file\*/,1/\*starting position\*/,instr(ftxt,',',1,1) - 1/\*Length\*/),--fname

substr(ftxt/\*tect file\*/,instr(ftxt,',',1,1) + 1/\*starting position\*/,instr(ftxt,',',1,2) - instr(ftxt,',',1,1) - 1/\*Length\*/),--lname

substr(ftxt/\*text file\*/,instr(ftxt,',',1,2) + 1/\*starting position\*/,instr(ftxt,',',1,3) - instr(ftxt,',',1,2) - 1/\*Length\*/),--address

substr(ftxt/\*text file\*/,instr(ftxt,',',1,3) + 1/\*starting position\*/,instr(ftxt,',',1,4) - instr(ftxt,',',1,3) - 1/\*Length\*/),--city

substr(ftxt/\*text file\*/,instr(ftxt,',',1,4) + 1/\*starting position\*/,instr(ftxt,',',1,5) - instr(ftxt,',',1,4) - 1/\*Length\*/),--state

substr(ftxt/\*text file\*/,instr(ftxt,',',1,5) + 1/\*starting position\*/,instr(ftxt,',',1,6) - instr(ftxt,',',1,5) - 1/\*Length\*/),--zipcode

substr(ftxt/\*text file\*/,instr(ftxt,',',1,6) + 1/\*starting position\*/)--email, no length because it goes to the end

INTO

fname,

lname,

address,

city,

state,

zipcode,

email

FROM

dual;

EXCEPTION

WHEN no\_data\_found THEN

EXIT; -- exiting once there are no more lines in the file

END;

INSERT INTO bb\_shopper (

idshopper,

firstname,

lastname,

address,

city,

state,

zipcode,

email,

dtentered --this column did not need to be in the parsing section since it will use SYSDATE

) VALUES (

bb\_shopper\_seq.NEXTVAL,

fname,

lname,

address,

city,

state,

zipcode,

email,

SYSDATE

);

lv\_line\_cnt := lv\_line\_cnt + 1; --keeping track of the number of lines

END LOOP;

COMMIT;

dbms\_output.put\_line('Number of rows inserted: ' || lv\_line\_cnt);

ELSE -- in case file does not open correctly

dbms\_output.put\_line('Error opening file');

END IF;

utl\_file.fclose(fh);

EXCEPTION -- catching any other errors

WHEN OTHERS THEN

dbms\_output.put\_line('Error SQL exception');

dbms\_output.put\_line(sqlerrm);

END;

**MGNT – What management must know about the migration**

What management on both sides needs to know is that the new customers being migrated into their database will have some missing information.

For the JustLee side of things, the customers being migrated in will be missing a Region value. This could be important as they may send out coupons/promos or even just gather data on buying habits based on region but will not have that for the new customers unless someone goes in manually and finds the region based on the customers address. Additionally, the Referred column will also be empty but that is not a problem as this is only used when other customers are referred by others to give the them deals and isn’t necessary for anything.

On the BrewBeans side of things there are a couple more columns that will be NULL. The Phone and Fax columns will have no values, and those could be of importance when trying to reach a customer. Province and Country will also not have values and just like above may come in handy, but these can also be added manually by looking at the addresses. Cookie and Promo are the next to have no value but, in my opinion, these are not as important to have straight away. Lastly is probably the most important of the NULL columns. The Username and Password columns are going to be NULL for the coming in customers. This will need to be addressed as they will need these if the customers are going to be logging in to but things and should be done so as soon as possible after the import is done.

**TECH – What a technician must know about the migration**

The first thing that I did for this project was look at both tables and figure out which columns were shared by both tables and thus be able to be moved to the other tables. The values I came up with were: Firstname, Lastname, Address, City, State, Zipcode and Email. Next step was changing variable values to match in both tables. Those changes are as follows:

In the JL Customers table:

Firstname increase to varchar2(15)

Lastname increase to varchar2(20)

Address increase to varchar2(40)

City increase to varchar2(20)

State changed from varchar2(2) to char(2)

Zip increase to varchar2(15)

In the BB Shopper table:

Email increase to varchar2(30)

There is also a CHECK constraint on the Region that I thought would maybe cause some issues but after researching, I found out then NULL values are acceptable in CHECK constraints, so no change needs to be made there.

Next looking at the procedures:

For the two **export procedures** I made the parameter to be put in be the file name that way the user can create it and know what file to use for the import procedure. The import procedures work by creating a cursor that retrieves the necessary columns from the respective table. The procedure then opens a file to write the column values from the cursor. It does so with a FOR loop that puts the values into a local variable that is concatenated to have commas in between all the values. After that it writes that line into the file and the loop goes again for every customer/shopper. There is also a counter in these procedures that counts how many lines are processed and outputs it when they are run so that the user can see and confirm the number.

For the **import procedures** the same parameter of the file name was given to it. Again, this is so the user can choose what the file is called in the export and then know which file to use for this import procedure. The procedure works by using the SUBSTR() and INSTR() functions to parse the given text string. I use the SUBSTR() to get out the information needed and the INSRT() to find the position of the commas. The easiest way to explain how this works would be to talk about each parameter in the SUBSTR() function. The parameters are as follows:

SUBSTR(string, starting position, length)

**String**: The string parameter is the same through the whole code as it will always be reading from the same file.

**Starting Position**: The starting position for the first variable is 1 since it needs to start at the beginning of the file. For the second variable, however, the starting position is the position of the second comma (found using INSTR) plus 1 since we need to start looking from right after the comma. The rest of the variables follow the same formula for finding the starting position. They are just the next position of the next incremental comma plus 1.

**Length**: The length of the first string to be taken out for the first variable is determined by finding the position of the first comma (again using INSTR) and subtracting 1 since we only want what is in between the commas. For the second variable we take the position of the second comma and subtract it from the position of the first comma and subtract one from that to again only get what is in between the commas. This continues for the rest of the variables with the positions of the comma being incremented each time. When you get to the last variable, however, you no longer need the length parameter as when it is left out, it will read to the end of the string. And since there are no commas after the last variable, this is what we need.

After the parsing section of the two import procedures comes the inserting of the data for the two procedures. This consists of just a basic INSERT statement that takes the variables we got from the parsing and puts them into the corresponding table column. The first variable is one that wasn’t parsed however since we use sequences to assign the customer# and idshopper column values. The only difference in both import procedures is that in the BB\_SHOPPER table there is a DTENTERED column that holds the date that a customer was entered into the database. So, I needed to add that into the importBB() procedure. This value did not need to be parsed though since it will just use the sysdate of the system that it is run on.

The line after the insert statement is just a counter so that when the procedures are run, they can output exactly how many rows were inserted into the tables. After that, the rest of the code is mainly just error handling.