

## CS 276 Week 5 Assignment – Records

**PART 1:**

- 1) Declare a programmer defined record based on the following table:

Name	Null	Type
SHIPPING_ID	NOT NULL	NUMBER(12)
PRICE		NUMBER(8,2)
PRODUCT_ORDERED		VARCHAR2(30)
AMOUNT_ORDERED		NUMBER(12)
FK_SHOPS_ID		NUMBER(12)

---Programmer Defined Record

```
DECLARE TYPE shipment_Record IS RECORD
```

```
(SHIPPING_ID NUMBER(12),
```

```
  PRICE NUMBER(8,2),
```

```
  PRODUCT_ORDERED VARCHAR2(30),
```

```
  AMOUNT_ORDERED NUMBER(12),
```

```
  FK_SHOPS_ID NUMBER(12)
```

```
);
```

```
BEGIN
```

```
  NULL;
```

```
END;
```

Worksheet	Query Builder
1	---Programmer Defined Record
2	DECLARE TYPE shipment_Record IS RECORD
3	(SHIPPING_ID NUMBER(12),
4	PRICE NUMBER(8,2),
5	PRODUCT_ORDERED VARCHAR2(30),
6	AMOUNT_ORDERED NUMBER(12),
7	FK_SHOPS_ID NUMBER(12)
8	);
9	BEGIN
10	NULL;
11	END;

Script Output	Query Result
Task completed in 0.016 seconds	
PL/SQL procedure successfully completed.	

2) Is the following a legal definition of a ROWTYPE attribute for a row?

```

CURSOR CLRTYP_CUR IS
    SELECT TYP_NU, TYPE_DS
    FROM CALLER_TYPE
    WHERE TYPE_DS LIKE NAME_INOUT || '%';
CLRTYP_REC CLRTYP_CUR%ROWTYPE;
NEXT_REC CLRTYP_CUR%ROWTYPE;

```

END;

```

CURSOR CLRTYP_CUR IS
    SELECT TYP_NU, TYPE_DS FROM CALLER_TYPE
    WHERE TYPE_DS LIKE NAME_INOUT || '%';
CLRTYP_REC CLRTYP_CUR%ROWTYPE;
NEXT_REC CLRTYP_CUR%ROWTYPE;

```

Green is valid

Red is invalid

3) Why won't the following code run?

```

1 DECLARE
2   -- Two identical TYPE declarations
3   TYPE DeptRec1 IS RECORD (dept_num NUMBER(2), dept_name VARCHAR2(14));
4   TYPE DeptRec2 IS RECORD (dept_num NUMBER(2), dept_name VARCHAR2(14));
5
6   dept1_info DeptRec1;
7   dept2_info DeptRec1;
8   dept3_info DeptRec2;
9
10 BEGIN
11   dept1_info := dept2_info;
12   dept2_info := dept3_info;
13
14 END;

```

### First problem

```

21 DECLARE
22   TYPE DeptRec1 IS RECORD (dept_num NUMBER(2), dept_name VARCHAR2(14));
23   TYPE DeptRec2 IS RECORD (dept_num NUMBER(2), dept_name VARCHAR2(14));
24
25   dept1_info DeptRec1;
26   dept2_info DeptRec1;
27   dept3_info DeptRec2;
28
29 BEGIN
30   dept1_info := dept2_info;
31   dept2_info := dept3_info;
32 END;

```

Script Output x Query Result x

Task completed in 0.028 seconds

### Error report -

```

ORA-06550: line 11, column 19:
PLS-00382: expression is of wrong type
ORA-06550: line 11, column 5:
PL/SQL: Statement ignored
06550. 00000 - "line %s, column %s:\n%s"
*Cause:      Usually a PL/SQL compilation error.
*Action:

```

### Second Problem

```

24
25 dept1_info DeptRec1;
26 dept2_info DeptRec1;
27 dept3_info DeptRec2;
28
29 BEGIN
30     dept1_info := dept2_info;
31     dept2_info := dept3_info;

```

Its trying to assign itself to itself, named something else. Attempt at locking? :)

## PART 2:

A **record** is a group of related data items stored in fields, each with its own name and datatype. You can think of a record as a variable that can hold a table row, or some columns from a table row. The fields correspond to table columns.

Records are composed of a group of fields, similar to the columns in a row. The **%ROWTYPE** attribute lets you declare a **PL/SQL record** that represents a row in a database table, without listing all the columns. Your code keeps working even after columns are added to the table. If you want to represent a subset of columns in a table, or columns from different tables, you can define a view or declare a cursor to select the right columns and do any necessary joins, and then apply **%ROWTYPE** to the view or cursor.

Run the Week 5 SQL file in the Moodle shell.

- 1) xxCreate an anonymous block that
  - a) xxCreates a cursor of the fields in the CHARGES table.
  - b) xxCreate a programmer defined record type that refers also refers to the columns in the table
  - c) xxLoop through the table, fetching the record and printing out the current balance for each customer code
  - d) After the loop, print out the number of late balances

The screenshot shows the Oracle SQL Developer interface. The top pane, titled 'Query Builder', contains a PL/SQL script. The script starts with a variable declaration for 'daysLate' and a 'BEGIN' block. It then enters a 'FOR' loop over a cursor named 'cur\_charges'. Inside the loop, it calculates 'daysLate' as the difference between the current date and the last payment date, outputs the current balance, and checks if the payment is late. If late, it increments a counter 'countLateBalances'. After the loop, it outputs the total number of late accounts and ends the procedure. The bottom pane, titled 'Script Output', shows the results of the script execution, including the balance for three accounts and the total number of late accounts, followed by a confirmation message.

```
37 daysLate number := 0;
38 BEGIN
39   FOR cur_record IN cur_charges
40     LOOP
41       daysLate := 0;
42       DBMS_OUTPUT.PUT_LINE(cur_record.customer_id || ' Balance: ' || cur_record.current_balance );
43
44       SELECT SysDate - cur_record.last_payment_date INTO daysLate FROM dual;
45       IF daysLate > cur_record.payment_terms
46         THEN
47           --DBMS_OUTPUT.PUT_LINE('LastPayment: ' || cur_record.last_payment_date || ' Late' );
48           countLateBalances := countLateBalances +1;
49         END IF;
50     END LOOP;
51     DBMS_OUTPUT.PUT_LINE('There are ' || countLateBalances || ' accounts late ');
52
53 END;
54
55 SELECT * FROM CHARGES;
```

8 Balance: 130  
9 Balance: 140  
10 Balance: 150  
There are 10 accounts late

PL/SQL procedure successfully completed.

SET serveroutput on;

DECLARE

CURSOR cur\_charges IS SELECT  
charge\_id,customer\_id,order\_id,last\_payment\_date,payment\_terms,current\_balance FROM charges;

TYPE charges\_record IS RECORD(charge\_id CHARGES.CHARGE\_ID%TYPE,  
customer\_id CHARGES.CUSTOMER\_ID%TYPE,  
order\_id CHARGES.ORDER\_ID%TYPE,  
last\_payment\_date CHARGES.LAST\_PAYMENT\_DATE%TYPE,  
payment\_terms CHARGES.PAYMENT\_TERMS%TYPE,  
current\_balance CHARGES.CURRENT\_BALANCE%TYPE);

countLateBalances number := 0;

daysLate number := 0;

BEGIN

FOR cur\_record IN cur\_charges

LOOP

daysLate := 0;

DBMS\_OUTPUT.PUT\_LINE(cur\_record.customer\_id || ' Balance: ' || cur\_record.current\_balance );

SELECT SysDate - cur\_record.last\_payment\_date INTO daysLate FROM dual;

IF daysLate > cur\_record.payment\_terms

THEN

--DBMS\_OUTPUT.PUT\_LINE('LastPayment: ' || cur\_record.last\_payment\_date || ' Late');

countLateBalances := countLateBalances +1;

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('There are ' || countLateBalances || ' accounts late');

END;

### **PART 3:**

#### **EXTRA CREDIT:**

Create a table SHOPPER which has the following fields:

- a) SHOPPER\_ID
- b) SHOPPER\_FIRST\_NAME
- c) SHOPPER\_LAST\_NAME

Create a table SHOPPING LIST which has the following fields:

For product\_size you would have a number, for product\_unit you would have "gallon","ounces", etc. so for a 5 gallon container of water PRODUCT\_SIZE would be 5 and PRODUCT\_UNIT would be "gallon").

- a) PRODUCT\_ID

- X**

xDefine a second record type SHOPPING\_LIST\_TYPE that has the same fields as the SHOPPING\_LIST table.

Define a fourth record type which is the compound record type of SHOPPER\_SHOPPING\_LIST.

xIn your execution section, assign values for all fields in the defined record types.

The screenshot displays the Oracle SQL Developer interface. The main window shows a PL/SQL procedure named `SL_INSERT_SHOPPER_LIST` in the `Query Builder` tab. The procedure is designed to insert data into the `SHOPPER_LIST` table based on a list of product units.

```

85  SL_PRODUCT_UNIT := 'gallon';
86  InsertShopperList(SL);
87
88  END;
89  --- Mega
90  DECLARE SP SUPER_SHOPPER_TYPE;
91  BEGIN
92  /*
93  OMA-86550 Couldn't get this to work...
94  SP_SHOPPER_ID := 1;
95  SP_SHOPPER_TYPE.SHOPPER_FIRST_NAME := 'bob';
96  SP_SHOPPER_TYPE.SHOPPER_LAST_NAME := 'builder';
97  SP_PRODUCT_ID := 1;
98  SP_FK_SHOPPER_ID := 2;
99  SP_PRODUCT_NAME := 'juice';
100 SP_PRODUCT_PRICE := 1.01;
101 SP_PRODUCT_SIZE := 5;
102 SP_PRODUCT_UNIT := 'gallon';
103 */ NULL;
104 END;
105
106 SELECT * FROM SHOPPER;
107
108 SELECT * FROM SHOPPING_LIST;

```

The procedure was executed successfully, as indicated by the status bar: "PL/SQL procedure successfully completed." The execution time was 0.034 seconds.

The `Statements - Log` window shows the following statements:

Seq#	Connection Name	Elapsed	SQL	Parameters
174	ibag	1	declare l_line varchar2(32767); l_line number; l_buffer varchar2(32767) := ''; l_lengthbuffer := 1000	
175	ibag	2	DECLARE --Define a record type SHOPPER_TYPE that has the same fields as the SHOPPER table. TY...	
176	ibag	1	declare l_line varchar2(32767); l_line number; l_buffer varchar2(32767) := ''; l_lengthbuffer := 1000	
177	ibag	2	DECLARE --Define a record type SHOPPER_TYPE that has the same fields as the SHOPPER table. TY...	
178	ibag	2	CREATE TABLE SHOPPER_LIST (SHOPPER_ID NUMBER(3) NOT NULL, SHOPPER_FIRST_NAME VARCHAR2(30) NOT...	
179	ibag	2	CREATE TABLE SHOPPING_LIST (PRODUCT_ID NUMBER, FK_SHOPPER_ID NUMBER, PRODUCT_NAME...	

The status bar at the bottom indicates: "Click on an identifier with the Control key down to perform 'Go to Declaration'". The interface also shows the file explorer on the left with various database objects like Tables, Views, and Sequences.

```
CREATE TABLE SHOPPER(  
    SHOPPER_ID NUMBER,  
    SHOPPER_FIRST_NAME VARCHAR2(30),
```

```
    SHOPPER_LAST_NAME VARCHAR2(30)
);
```

--For product\_size you would have a number, for product\_unit you would have "gallon","ounces", etc.

--so for a 5 gallon container of water PRODUCT\_SIZE would be 5 and PRODUCT\_UNIT would be "gallon").

```
CREATE TABLE SHOPPING_LIST(
PRODUCT_ID NUMBER,
FK_SHOPPER_ID NUMBER,
PRODUCT_NAME VARCHAR2(30),
PRODUCT_PRICE NUMBER(10,2),
PRODUCT_SIZE NUMBER,
PRODUCT_UNIT VARCHAR2(10)
);
```

DECLARE

--Define a record type SHOPPER\_TYPE that has the same fields as the SHOPPER table.

```
TYPE SHOPPER_TYPE IS RECORD (
    SHOPPER_ID NUMBER,
    SHOPPER_FIRST_NAME VARCHAR2(30),
    SHOPPER_LAST_NAME VARCHAR2(30));
```

--Define a second record type SHOPPING\_LIST\_TYPE that has the same fields as the SHOPPING\_LIST table.

```
TYPE SHOPPING_LIST_TYPE IS RECORD (PRODUCT_ID NUMBER,
    FK_SHOPPER_ID NUMBER,
    PRODUCT_NAME VARCHAR2(30),
    PRODUCT_PRICE NUMBER(10,2),
    PRODUCT_SIZE NUMBER,
    PRODUCT_UNIT VARCHAR2(10));
```

--Define a third record type of the two above defined record types – i.e. one of SHOPPER and one of SHOPPING\_LIST named SHOPPER\_SHOPPING\_LIST.



```
TYPE SUPER_SHOPPER_TYPE IS RECORD (  
  SHOPPER_ID SHOPPER_TYPE,  
  SHOPPER_FIRST_NAME# SHOPPER_TYPE,  
  SHOPPER_LAST_NAME# SHOPPER_TYPE,  
  PRODUCT_ID# SHOPPER_LIST_TYPE,  
  FK_SHOPPER_ID# SHOPPER_LIST_TYPE,  
  PRODUCT_NAME# SHOPPER_LIST_TYPE,  
  PRODUCT_PRICE# SHOPPER_LIST_TYPE,  
  PRODUCT_SIZE# SHOPPER_LIST_TYPE,  
  PRODUCT_UNIT# SHOPPER_LIST_TYPE);
```

```
PROCEDURE InsertShopper(ST IN SHOPPER_TYPE)
```

```
IS
```

```
BEGIN
```

```
  INSERT INTO SHOPPER VALUES ST;
```

```
END;
```

```
PROCEDURE InsertShopperList(SL IN SHOPPER_LIST_TYPE)
```

```
IS
```

```
BEGIN
```

```
  INSERT INTO SHOPPING_LIST VALUES SL;
```

```
END;
```

```
BEGIN
```

```
  -- Assign values to each type
```

```
  DECLARE ST SHOPPER_TYPE;
```

```
  BEGIN
```

```

ST.SHOPPER_ID := 1;
ST.SHOPPER_FIRST_NAME := 'bob';
ST.SHOPPER_LAST_NAME := 'builder';
InsertShopper(ST);

ST.SHOPPER_ID := 2;
ST.SHOPPER_FIRST_NAME := 'nancy';
ST.SHOPPER_LAST_NAME := 'drew';
InsertShopper(ST);
END;
-- Shopping List
DECLARE SL SHOPPER_LIST_TYPE;
BEGIN
    SL.PRODUCT_ID := 1;
    SL.FK_SHOPPER_ID := 2;
    SL.PRODUCT_NAME := 'juice';
    SL.PRODUCT_PRICE := 1.01;
    SL.PRODUCT_SIZE := 5;
    SL.PRODUCT_UNIT := 'gallon';
    InsertShopperList(SL);

    SL.PRODUCT_ID := 2;
    SL.FK_SHOPPER_ID := 2;
    SL.PRODUCT_NAME := 'juice';
    SL.PRODUCT_PRICE := 1.01;
    SL.PRODUCT_SIZE := 5;
    SL.PRODUCT_UNIT := 'gallon';
    InsertShopperList(SL);

```

```

END;

--- Mega

DECLARE SP SUPER_SHOPPER_TYPE;

BEGIN

/*

ORA-06550 Couldn't get this to work...

SP.SHOPPER_ID := 1;

SP.SHOPPER_TYPE.SHOPPER_FIRST_NAME := 'bob';

SP.SHOPPER_TYPE.SHOPPER_LAST_NAME := 'builder';

SP.PRODUCT_ID := 1;

SP.FK_SHOPPER_ID := 2;

SP.PRODUCT_NAME := 'juice';

SP.PRODUCT_PRICE := 1.01;

SP.PRODUCT_SIZE := 5;

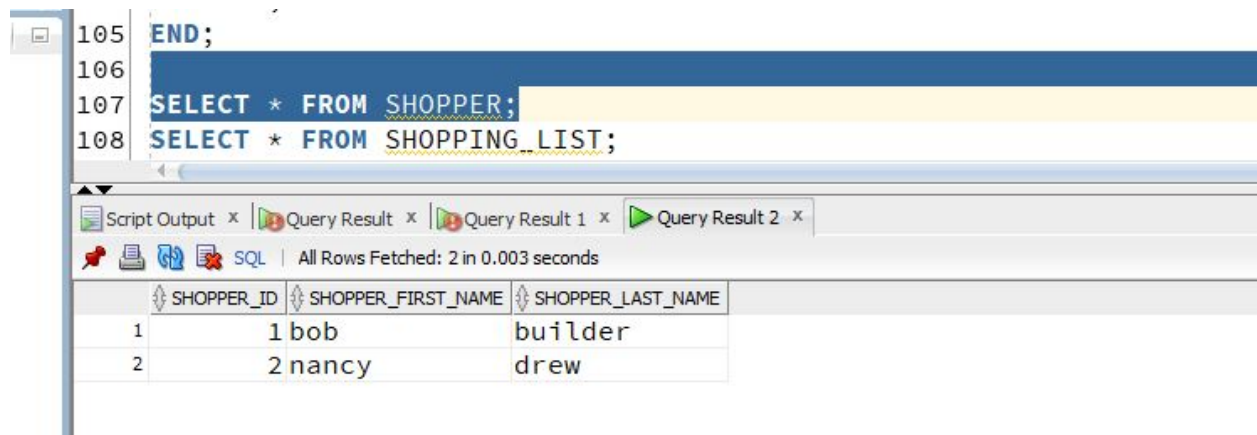
SP.PRODUCT_UNIT := 'gallon';

*/ NULL;

END;

END;

```



The screenshot shows an SQL IDE interface. The script editor contains the following SQL code:

```

105 END;
106
107 SELECT * FROM SHOPPER;
108 SELECT * FROM SHOPPING_LIST;

```

Below the script editor, there is a results window with the following tabs: "Script Output", "Query Result", "Query Result 1", and "Query Result 2". The "Query Result" tab is active, showing the results of the first query. The status bar indicates "All Rows Fetched: 2 in 0.003 seconds".

SHOPPER_ID	SHOPPER_FIRST_NAME	SHOPPER_LAST_NAME
1	bob	builder
2	nancy	drew

107 | SELECT \* FROM SHOPPER;  
108 | SELECT \* FROM SHOPPING\_LIST;

Script Output x Query Result x Query Result 1 x Query Result 2 x

SQL | All Rows Fetched: 2 in 0.003 seconds

	PRODUCT_ID	FK_SHOPPER_ID	PRODUCT_NAME	PRODUCT_PRICE	PRODUCT_SIZE	PRODUCT_UNIT
1	1	2	juice	1.01	5	gallon
2	2	2	juice	1.01	5	gallon

## PART 2:

- 1) How would I alter the above so that I am able to PASS values for my fields in the defined record types, such that I don't have to hard code them in the code itself? You will need to create a package that contains your procedures and variables.