Table of Contents

[CHAPTER 20: ADVANCED OBJECT TYPE USAGE 2](#_Toc415679425)

[Theory 2](#_Toc415679426)

[AIM 4](#_Toc415679427)

[Lab Exercise 20: ADVANCED OBJECT TYPE USAGE 5](#_Toc415679428)

[1. Object Collection 6](#_Toc415679429)

[2. Abstract Object 9](#_Toc415679430)

[3. User-Defined Aggregate Function 17](#_Toc415679431)

[4. Hierarchy Storage 23](#_Toc415679432)

[SUMMARY 32](#_Toc415679433)

[REFERENCES 33](#_Toc415679434)

[INDEX 34](#_Toc415679435)

# CHAPTER 20: ADVANCED OBJECT TYPE USAGE

## Theory

In this chapter, you will examine some advanced Object Type usage. In specific, you will learn how to deal with object hierarchy in which a parent object is just a placeholder to its children, abstract object. You will also learn about how to use Object Type in advanced PL/SQL code such as BULK COLLECT syntax. User-defined Object Type can serve you to build and advanced function such as a custom aggregate function. Finally, you will learn about how Oracle stores Object Type in a relational table, so you know what is really going on in the back scene.

Oracle supports using Object Types with some advanced features in PL/SQL programming, such as BULK COLLECT clauses. Thus, you may use this features to enhance your code performance in collecting and manipulating objects stored in Object Table of Column Object.

In some cases, you business logic requires to be built as a hierarchy so the parent object represents an abstract object, not to create an instance of it but to reserve a common node for different objects. For example, if you want to build a graphical application which allows the user to use some shapes from the list. In such application, you may think to create a "SHAPE" object type to be a parent of all of your "TRIANGLE", "RECTANGLE", and other shapes. However, there is no really object named "SHAPE" that the user may drag and use! It is just a placeholder or a common node for other objects. In such cases, you create an abstract object, in Oracle terminology NOT INSTANTIABLE object, to represent this node.

Oracle has many useful aggregate functions such as MAX, MIN, AVG and others. These functions allow you to compute a single value for a set or rows. In most cases, these functions are sufficient; but in other cases, you may find it useful if you can create your own aggregate function. For example, if you want to create a "BEST" aggregate function, which takes a set of "PRODUCT" Object Types and return the best one depending on some pre-defined criteria. Fortunately, Oracle makes it possible to create your own aggregate function. All what you need is to create an Object Type which implements a set of functions which are:

* **ODCIAggregateInitialize**: A static function that is used to create new instance of the same object and to initializes the Object attributes, if necessary.
* **ODCIAggregateIterate**: A function that is executed for each row. Use this function to do your aggregate works. The function gets a "value" parameter as input, which represents the value you intend to aggregate from each row.
* **ODCIAggregateMerge**: Oracle supports multithreading and parallel execution. Thus, when you submit one SELECT statement, Oracle may create multiple threads to execute your query. In the aggregation function perspective, you code may split across multiple threads. In such case, you of course need to merge them back together before doing your final calculation. This function serves you for this purpose. The function gets the context of the other thread and combines it with its own.
* **ODCIAggregateTerminate**: Use this function to do the final calculation before returning the result to the user.

After creating the object type, you need to create an aggregate function and mention your Object Type as implementer of it.

Oracle stores your hierarchal object structure in a regular relational table. For example, if you have the following hierarchy:

When you created an Object Table to store this hierarchy using "Vehicle" object as a base, Oracle automatically creates columns to store the vehicle's attributes AND hidden columns to hold the children's attributes. Actually, this table will store all Object Types. However, you can only handle the children's attributes if you explicitly ask the PL/SQL parser to get "car" or "motorcycle" objects.

## AIM

The AIM of the following exercise is to explore some advanced Object Type usage.

The steps involved will include:

* Object Collection
* Abstract Object
* User-Defined Aggregate Function
* Hierarchy Storage

In general, lab exercises are done in sequential order. Thus, it is assumed that you successfully completed the previous labs. However, not all previous labs are required. Please be sure to run the following lab before proceeding:

* Installing Oracle Database 12c.
* Object Types
* Object Type Storage
* Object Type Manipulation

Estimated Completion Time:

30 minutes

# Lab Exercise 20: ADVANCED OBJECT TYPE USAGE

|  |
| --- |
|  |

## Object Collection

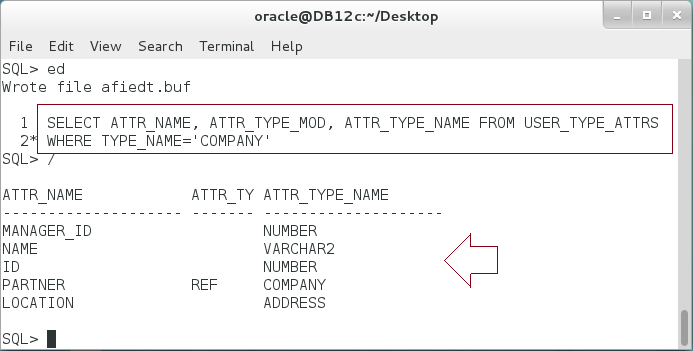
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

**Step 2:** We are going to use one of the previously created object, "COMPANY". Execute the following query:

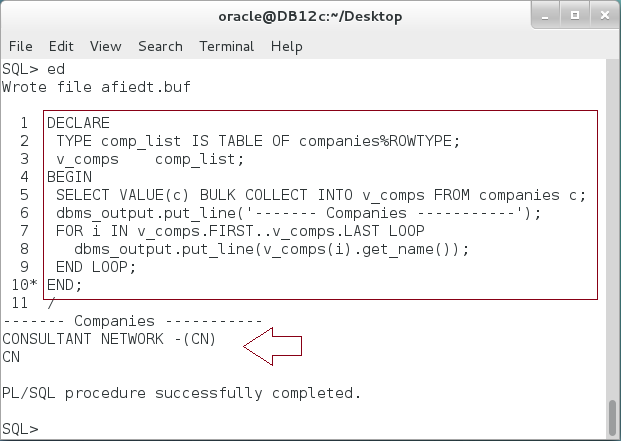
|  |  |
| --- | --- |
| Command | Description |
| SELECT ATTR\_NAME, ATTR\_TYPE\_MOD, ATTR\_TYPE\_NAME FROM USER\_TYPE\_ATTRS |  |
| WHERE TYPE\_NAME=**'COMPANY'** |
| / |

****

**Please:** Refer to Chapter 18 to create the object if cannot find it.

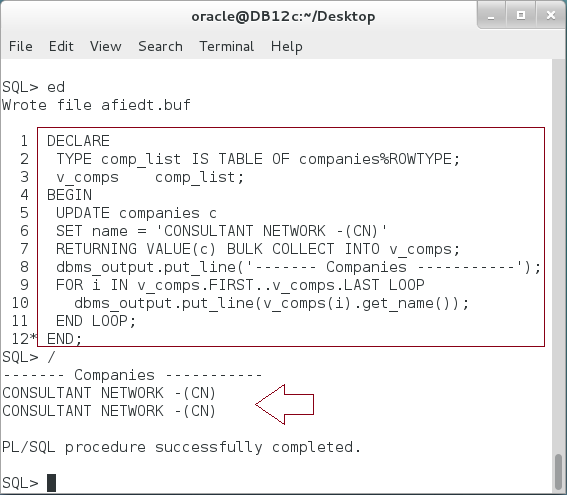
**Step 3:** You can define a nested table of Object Type and use it to bulk collect the result set of the query as shown below:

|  |
| --- |
| Command |
| DECLARE |
| **TYPE** comp\_list **IS TABLE OF** **companies%ROWTYPE**; |
| v\_comps comp\_list; |
| BEGIN |
| SELECT **VALUE(c) BULK COLLECT INTO** v\_comps FROM companies c; |
| dbms\_output.put\_line('------- Companies -----------'); |
| FOR i IN v\_comps.FIRST..v\_comps.LAST LOOP |
| dbms\_output.put\_line(v\_comps(i).get\_name()); |
| END LOOP; |
| END; |
| / |



**Step 4:** You also able to use RETURNING BULK COLLECT clause as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE comp\_list IS TABLE OF companies%ROWTYPE; |  |
| v\_comps comp\_list; |  |
| BEGIN |  |
| **UPDATE** companies c | Use Returning Bulk collect clause. |
| SET name = 'CONSULTANT NETWORK -(CN)' |
| **RETURNING VALUE(c) BULK COLLECT INTO** v\_comps; |
| dbms\_output.put\_line('------- Companies -----------'); |
| FOR i IN v\_comps.FIRST..v\_comps.LAST LOOP |
| dbms\_output.put\_line(v\_comps(i).get\_name()); |
| END LOOP; |  |
| END; |  |
| / |  |

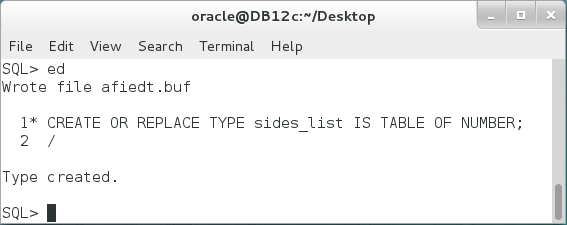


**Please note:** That were some examples of object collections. Explore more expression by yourself.

## Abstract Object

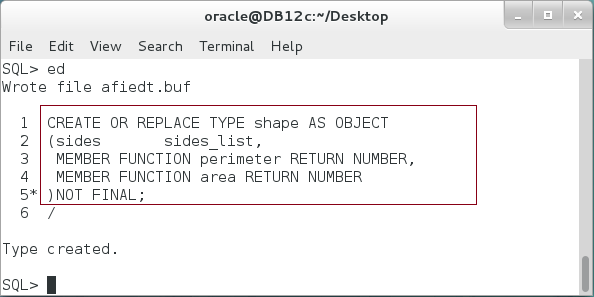
**Step 1:** We are going to create new Object Type "SHAPE". The "SHAPE" object depends on a nested table "SIDES\_LIST". Create the nested Table first as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TYPE sides\_list IS TABLE OF NUMBER; |  |
| / |  |

****

**Step 2:** Create the "SHAPE" object type as shown below:

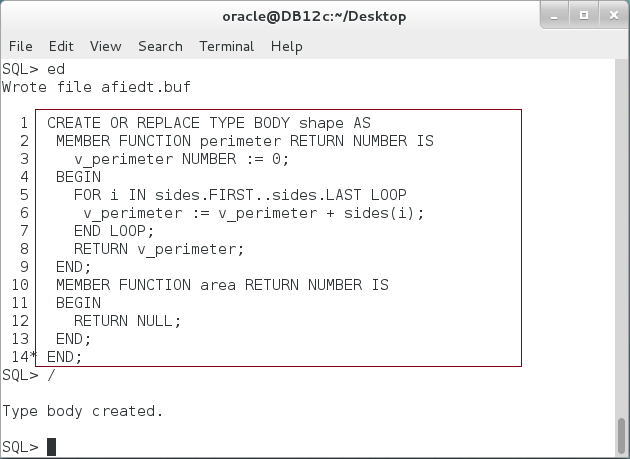
|  |  |
| --- | --- |
| Line | Description |
| **CREATE OR REPLACE TYPE** shape **AS OBJECT** |  |
| (sides sides\_list, | This Object should serve as parent. |
| MEMBER FUNCTION perimeter RETURN NUMBER, |
| MEMBER FUNCTION area RETURN NUMBER |
| )**NOT FINAL;** |
| / |



**Please note:** The SHAPE object has two member functions, PERIMETER and AREA. This seems logical so far since all of our shapes have perimeters and areas.

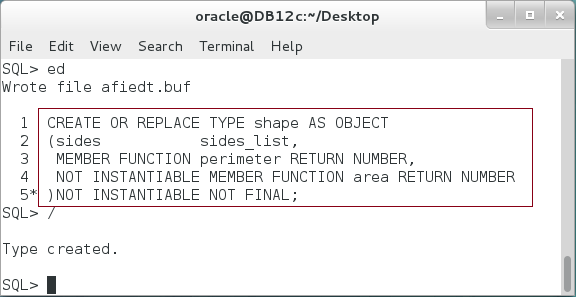
**Step 3:** Implement the SHAPE object body as shown below:

|  |  |
| --- | --- |
| Line |  |
| **CREATE OR REPLACE TYPE BODY** shape AS |  |
| **MEMBER FUNCTION perimeter** RETURN NUMBER IS |  |
| v\_perimeter NUMBER := 0; |  |
| BEGIN |  |
| FOR i IN sides.FIRST..sides.LAST LOOP |  |
| v\_perimeter := v\_perimeter + sides(i); |  |
| END LOOP; |  |
| RETURN v\_perimeter; |  |
| END; |  |
| **MEMBER FUNCTION area** RETURN NUMBER IS | We can't implement "area" function. |
| BEGIN |
| **RETURN NULL;** |
| END; |
| END; |
| / |  |

**Please note:** Although it seemed logical for the first time, we actually cannot implement "area" function unless we know the child type, triangle or rectangle.

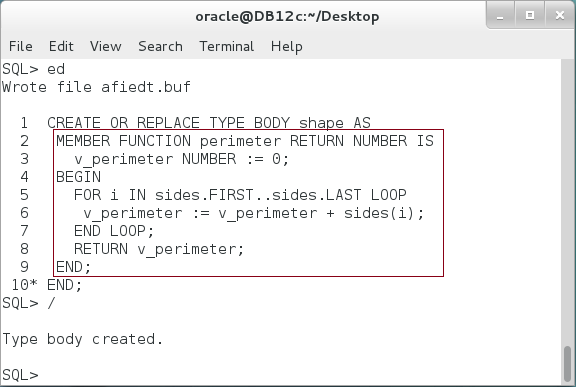
**Step 4:** In the previous case, you should use the NOT INSTANTIABLE object and AREA method. Re-Create the Object specification as shown below:

|  |  |
| --- | --- |
| Line |  |
| CREATE OR REPLACE TYPE shape AS OBJECT |  |
| (sides sides\_list, | Not instantiable method and Object. |
| MEMBER FUNCTION perimeter RETURN NUMBER, |
| **NOT INSTANTIABLE** MEMBER FUNCTION area RETURN NUMBER |
| )**NOT INSTANTIABLE** NOT FINAL; |
| / |



**Step 5:** Implement the SHAPE object body as shown below:

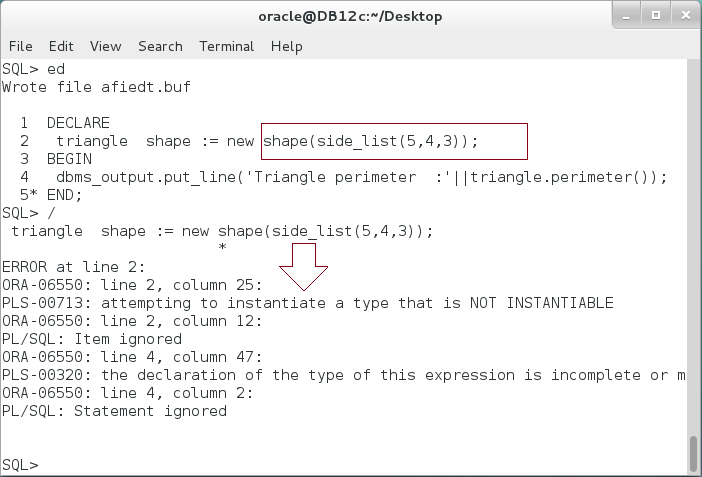
|  |  |
| --- | --- |
| Line |  |
| CREATE OR REPLACE TYPE BODY shape AS |  |
| **MEMBER FUNCTION perimeter** RETURN NUMBER IS | Only member function appears on the type body. |
| v\_perimeter NUMBER := 0; |
| BEGIN |
| FOR i IN sides.FIRST..sides.LAST LOOP |
| v\_perimeter := v\_perimeter + sides(i); |
| END LOOP; |
| RETURN v\_perimeter; |
| END; |  |
| END; |  |
| / |  |



**Please note:** NOT INSTANTIABLE method is not implemented in the body. Only regular (member, static, and constructor) methods should be implemented in the type body.

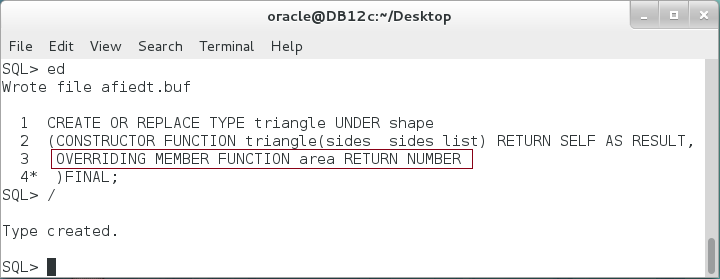
**Step 6:** Try to create an object instance of NOT INSTANTIABLE object as shown below:

|  |  |
| --- | --- |
| Line |  |
| DECLARE |  |
| **triangle shape := new shape(side\_list(5,4,3));** | Failed because it is not instantiable. |
| BEGIN |
| dbms\_output.put\_line |
| ('Triangle perimeter :'||triangle.perimeter()); |
| END; |  |
| / |  |



**Step 7:** Create "TRIANGLE" object type as a child of "SHAPE" object as shown:

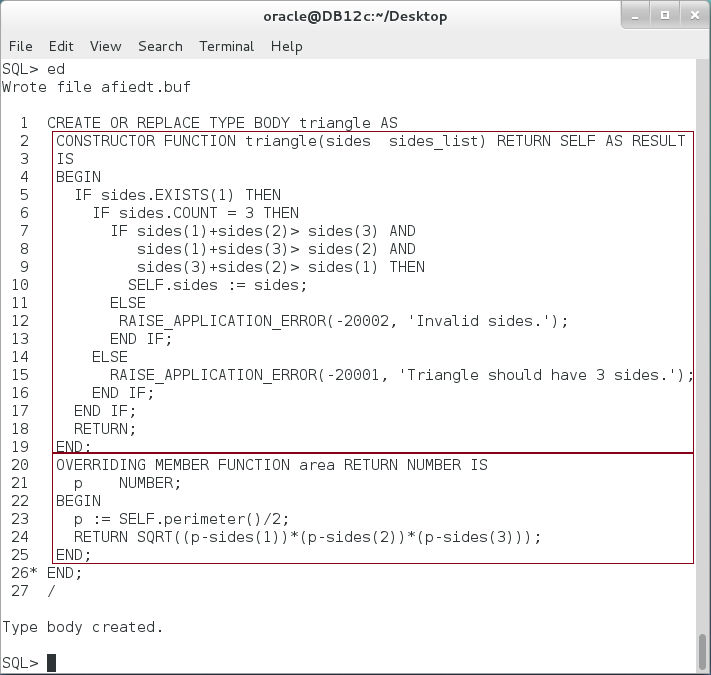
|  |
| --- |
| Line |
| CREATE OR REPLACE **TYPE triangle UNDER shape** |
| (CONSTRUCTOR FUNCTION triangle(sides sides\_list) RETURN SELF AS RESULT, |
| **OVERRIDING MEMBER FUNCTION area RETURN NUMBER** |
| )FINAL; |
| / |



**Please note:** TRIANGLE object should implement "AREA" method.

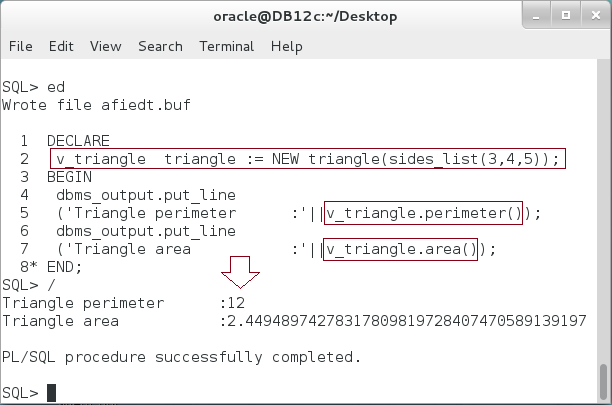
**Step 8:** Implement TRIANGLE object body as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE BODY triangle AS |  |
| CONSTRUCTOR FUNCTION triangle(sides sides\_list) RETURN SELF AS RESULT | Use constructor to check if the use enter a valid triangle's sides. |
| IS |
| BEGIN |
| IF sides.EXISTS(1) THEN |
| IF sides.COUNT = 3 THEN |
| IF sides(1)+sides(2)> sides(3) AND |
| sides(1)+sides(3)> sides(2) AND |
| sides(3)+sides(2)> sides(1) THEN |
| SELF.sides := sides; |
| ELSE |
| RAISE\_APPLICATION\_ERROR(-20002, 'Invalid sides.'); |
| END IF; |
| ELSE |
| RAISE\_APPLICATION\_ERROR(-20001, 'Triangle should have 3 sides.'); |
| END IF; |  |
| END IF; |  |
|  |  |
| RETURN; |  |
| END; |  |
| **OVERRIDING MEMBER FUNCTION area RETURN NUMBER IS** | Now we are able to implement the "**AREA**" method. |
| p NUMBER; |
| BEGIN |
| p := SELF.perimeter()/2; |
| RETURN SQRT((p-sides(1))\*(p-sides(2))\*(p-sides(3))); |
| END; |
| END; |  |
| / |  |



**Step 9:** Test TRIANGLE object using the following PL/SQL block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **v\_triangle triangle := NEW triangle(sides\_list(3,4,5));** |  |
| BEGIN |  |
| dbms\_output.put\_line |  |
| ('Triangle perimeter :'||**v\_triangle.perimeter()**); |  |
| dbms\_output.put\_line |  |
| ('Triangle area :'||**v\_triangle.area()**); |  |
| END; |  |
| / |  |



**Step 10:** Create your own **SQUARE** and **RECTANGLE** objects under SHAPE object.

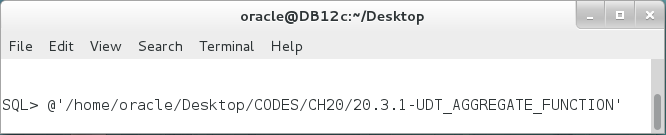
## User-Defined Aggregate Function

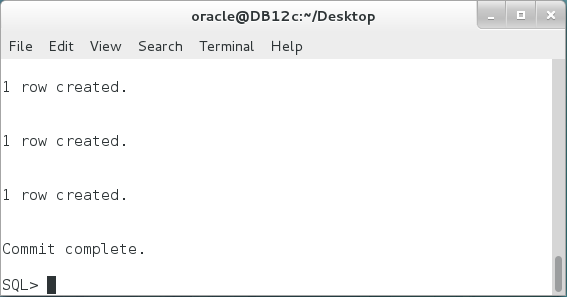
**Step 1:** You are going to create a GPA aggregate function in which the student average is calculated so that:

* Any mark under or equal to 50 should be considered as 50, "Academic Zero".
* Subject mark "P" means this subject is not required in the study plan and should not calculated.
* Any NULL mark represents ongoing subject and therefore is not calculated.
* GPA is a sum of all calculated marks divided by the subject counts.

Ask your trainer to give you a file "**20.3.1-UDT\_AGGREGATE\_FUNCTION**" to create "**STUDENTS\_MARKS**" table and fill it in with some student's mark as shown below:

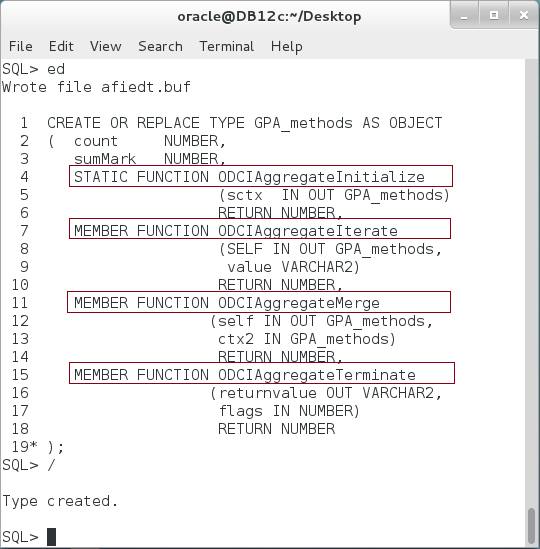
|  |
| --- |
| Line |
| **@**'/home/oracle/Desktop/CODES/CH20/**20.3.1-UDT\_AGGREGATE\_FUNCTION'** |





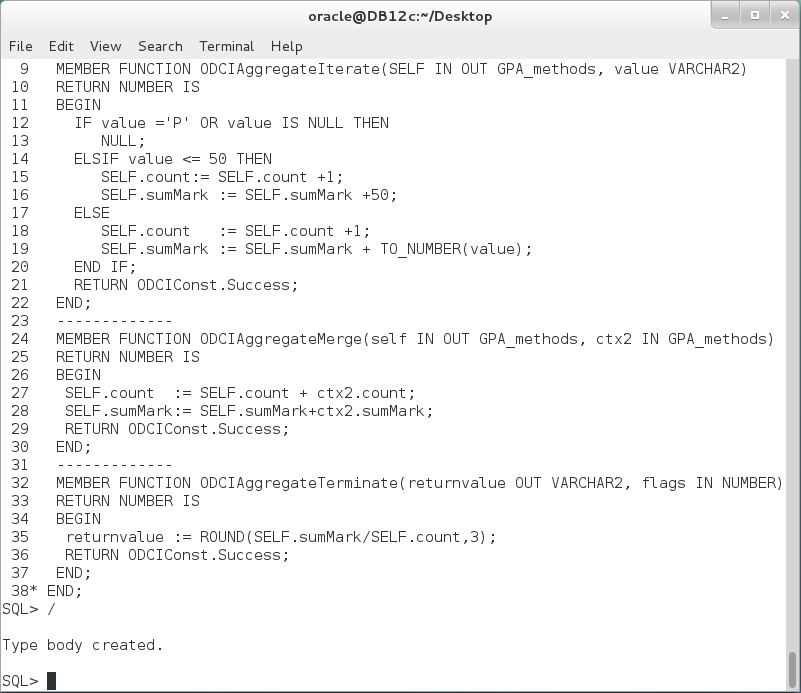
**Step 2:** Create "**GPA\_METHODS**" object type as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE GPA\_methods AS OBJECT |  |
| ( count NUMBER, | To hold the number of subject |
| sumMark NUMBER, | To hold the accumulated sum |
| **STATIC FUNCTION ODCIAggregateInitialize** | Static function gets Object type instance as input and initializes it. |
| (sctx IN OUT GPA\_methods) |
| RETURN NUMBER, |
| **MEMBER FUNCTION ODCIAggregateIterate** | Get "value" parameter which represents the mark in the row. |
| (SELF IN OUT GPA\_methods, |
| value VARCHAR2) |
| RETURN NUMBER, |
| **MEMBER FUNCTION ODCIAggregateMerge** | It gets the "ctx2" parameter which represents the instance of the second thread. |
| (self IN OUT GPA\_methods, |
| ctx2 IN GPA\_methods) |
| RETURN NUMBER, |
| **MEMBER FUNCTION ODCIAggregateTerminate** | The function has "returnvalue" OUT parameter which represents the final result. |
| (returnvalue OUT VARCHAR2, |
| flags IN NUMBER) |
| RETURN NUMBER |
| ); |  |
| / |  |

****

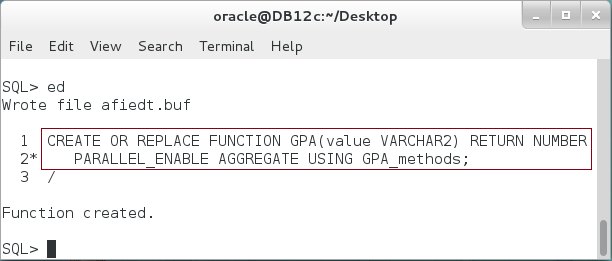
**Step 3:** Implement the Object Type as shown:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE BODY GPA\_methods AS |  |
| **STATIC FUNCTION ODCIAggregateInitialize**  (sctx IN OUT GPA\_methods) | Create new object with parameters "**count**" and "**sumMark**" equal to **"0"**. |
| RETURN NUMBER IS |
| BEGIN |
| sctx := NEW GPA\_methods(0,0); |
| RETURN ODCIConst.Success; |
| END; |
| ------------- |
| **MEMBER FUNCTION ODCIAggregateIterate** (SELF IN OUT GPA\_methods, value VARCHAR2) | For each iteration, get the check the mark "value" and apply the rules of counting as mentioned previously. |
| RETURN NUMBER IS |
| BEGIN |
| IF value ='P' OR value IS NULL THEN |
| NULL; |
| ELSIF value <= 50 THEN |
| SELF.count:= SELF.count +1; |
| SELF.sumMark := SELF.sumMark +50; |
| ELSE |
| SELF.count := SELF.count +1; |
| SELF.sumMark := SELF.sumMark + TO\_NUMBER(value); |
| END IF; |
| RETURN ODCIConst.Success; |
| END; |
| ------------- |  |
| **MEMBER FUNCTION ODCIAggregateMerge**  (self IN OUT GPA\_methods, ctx2 IN GPA\_methods) | Merge the result found so far on the second thread "**ctx2**" to this object instance "**SELF**". |
| RETURN NUMBER IS |
| BEGIN |
| SELF.count := SELF.count + ctx2.count; |
| SELF.sumMark:= SELF.sumMark+ctx2.sumMark; |
| RETURN ODCIConst.Success; |
| END; |
| ------------- |  |
| **MEMBER FUNCTION ODCIAggregateTerminate**  (returnvalue OUT VARCHAR2, flags IN NUMBER) | Calculate student GPA so it represents the sum of all calculated marks divided by their count. |
| RETURN NUMBER IS |
| BEGIN |
| returnvalue := ROUND(SELF.sumMark/SELF.count,3); |
| RETURN ODCIConst.Success; |
| END; |
| END; |  |
| / |  |

****

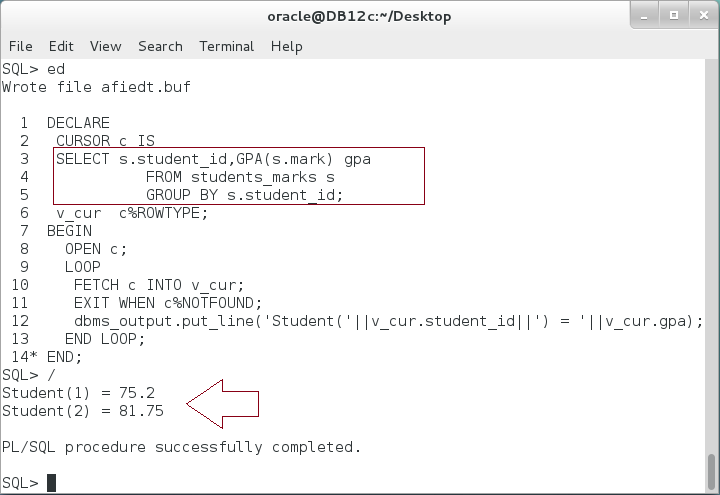
**Step 4:** Create the aggregate function that use "GPA\_MARKS" object type as shown:

|  |
| --- |
| Line |
| CREATE OR REPLACE FUNCTION **GPA(value VARCHAR2)** RETURN NUMBER |
| PARALLEL\_ENABLE AGGREGATE **USING GPA\_methods;** |
| / |



**Step 5:** Use "GPA" aggregate function as any other Oracle native aggregate function. Use the following block as example:

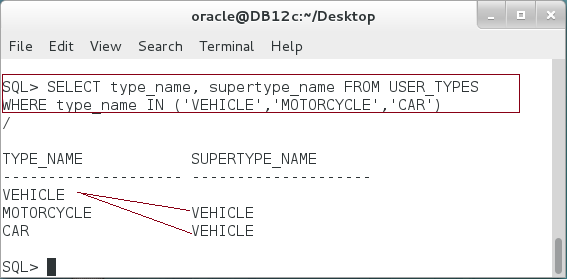
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| CURSOR c IS | **GPA** function with **GROUP BY** expression. |
| **SELECT s.student\_id,GPA(s.mark) gpa** |
| **FROM students\_marks s** |
| **GROUP BY s.student\_id;** |
| v\_cur c%ROWTYPE; |
| BEGIN |  |
| OPEN c; |  |
| LOOP |  |
| FETCH c INTO v\_cur; |  |
| EXIT WHEN c%NOTFOUND; |  |
| dbms\_output.put\_line |  |
| ('Student('||v\_cur.student\_id||') = '||v\_cur.gpa); |  |
| END LOOP; |  |
| END; |  |
| / |  |



## Hierarchy Storage

**Step 1:** We are going to use one of our previous Object hierarchy created in the previous chapters, "VEHICLE", "CAR", and "MOTORCYLE". Execute the following query to view the types' details:

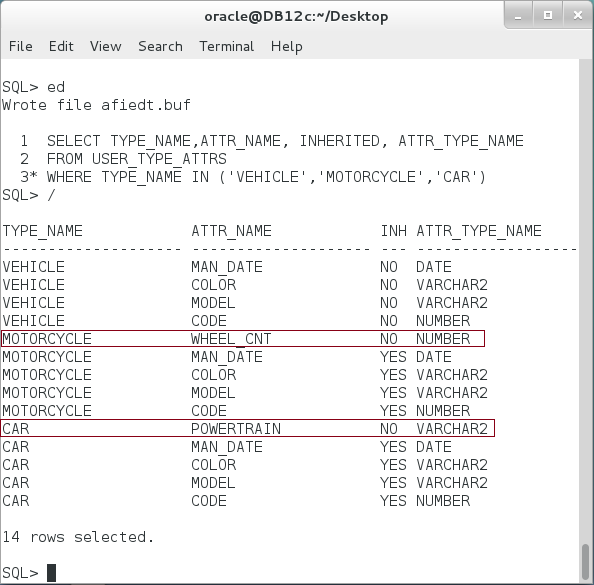
|  |  |
| --- | --- |
| Line | Description |
| SELECT type\_name, supertype\_name FROM USER\_TYPES |  |
| WHERE type\_name IN ('VEHICLE','MOTORCYCLE','CAR') |  |
| / |  |



**Please note:** if you cannot find the objects, please refer to chapter 19.

**Step 2:** Review the attributes of these objects as shown below:

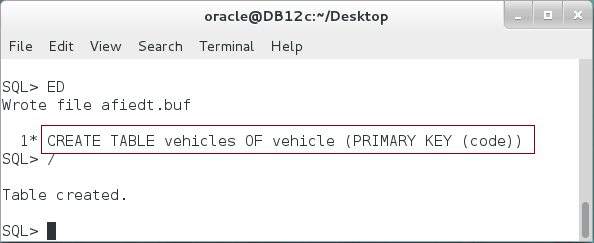
|  |  |
| --- | --- |
| Line | Description |
| SELECT TYPE\_NAME,ATTR\_NAME, INHERITED, ATTR\_TYPE\_NAME |  |
| FROM USER\_TYPE\_ATTRS |  |
| WHERE TYPE\_NAME IN ('VEHICLE','MOTORCYCLE','CAR') |  |
| / |  |



**Please note:** MOTORCYCLE and CAR object inherit all of their attributes from VEHICLE except for "WHEEL\_CNT" and "POWERTRAIN" attributes respectively.

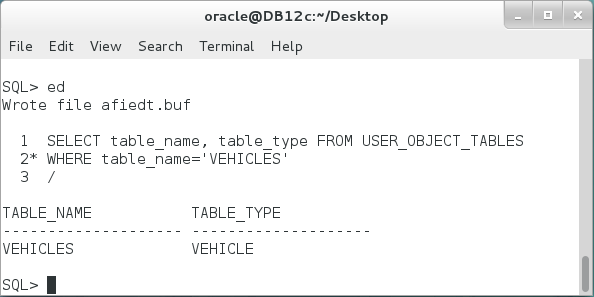
**Step 3:** Create an Object Table based on "VEHICLE" object as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE TABLE vehicles OF vehicle (PRIMARY KEY (code)) |  |
| / |  |

****

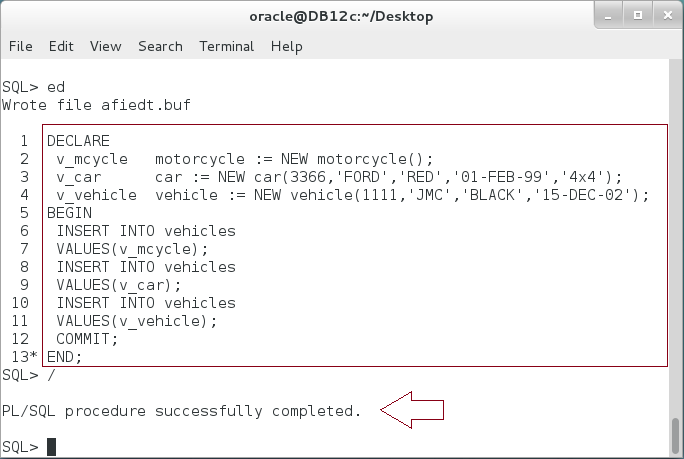
**Step 4:** You can query your Object Tables from USER\_OBJECT\_TABLES view as shown:

|  |  |
| --- | --- |
| Line | Description |
| SELECT table\_name, table\_type FROM USER\_OBJECT\_TABLES |  |
| WHERE table\_name='VEHICLES' |  |
| / |  |

****

**Step 5:** The question here, how we can store our all types in this object table? Execute the following PL/SQL block:

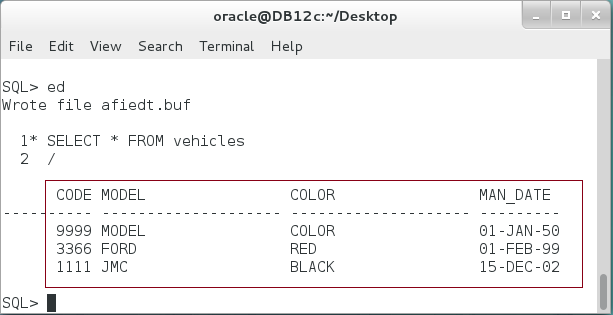
|  |  |
| --- | --- |
| Line | Description |
| DECLARE | Create different Object Type of the hierarchy |
| v\_mcycle **motorcycle** := NEW motorcycle(); |
| v\_car **car** := NEW car(3366,'FORD','RED','01-FEB-99','4x4'); |
| v\_vehicle **vehicle** := NEW vehicle(1111,'JMC','BLACK','15-DEC-02'); |
| BEGIN |
| INSERT INTO vehicles | Insert **MOTORCYCLE** |
| VALUES(**v\_mcycle**); |
| INSERT INTO vehicles | Insert **CAR** |
| VALUES(**v\_car**); |
| INSERT INTO vehicles | Insert **VEHICLE** |
| VALUES(**v\_vehicle**); |
| COMMIT; |  |
| END; |  |
| / |  |

****

**Please note:** The block succeeded! We create the Object Table to hold a VEHICLE type not to hold all the hierarchy under it. **Explain it!**

**Step 6:** Query the "VEHICLES" table as shown below:

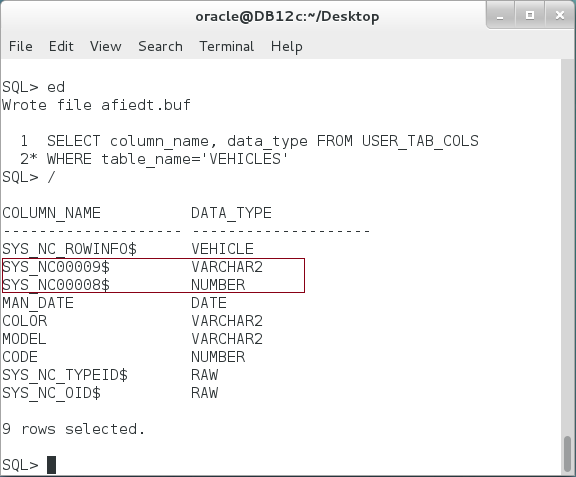
|  |  |
| --- | --- |
| Line | Description |
| SELECT \* FROM vehicles |  |
| / |  |

****

**Please note:** All object instances appear in the list, **but where** is "WHEEL\_CNT" attribute of "MOTORCYCLE" object and where is "POWERTRAIN" attribute of "CAR" object?

**Step 7:** Execute the following query to view the Object Table's column:

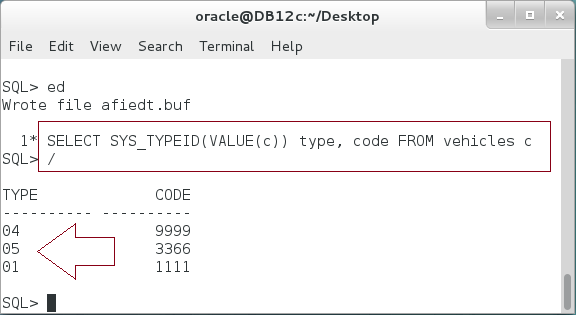
|  |  |
| --- | --- |
| Line | Description |
| SELECT column\_name, data\_type FROM USER\_TAB\_COLS |  |
| WHERE table\_name='VEHICLES' |  |
| / |  |

****

**Did you notice the hidden columns?** These columns are used to store the attributes of all Object Types under the parent object, "**VEHICLE**".

**Step 8:** The question now is: How can we distinguish between different Object Types stored in the Object Table? Execute the following query:

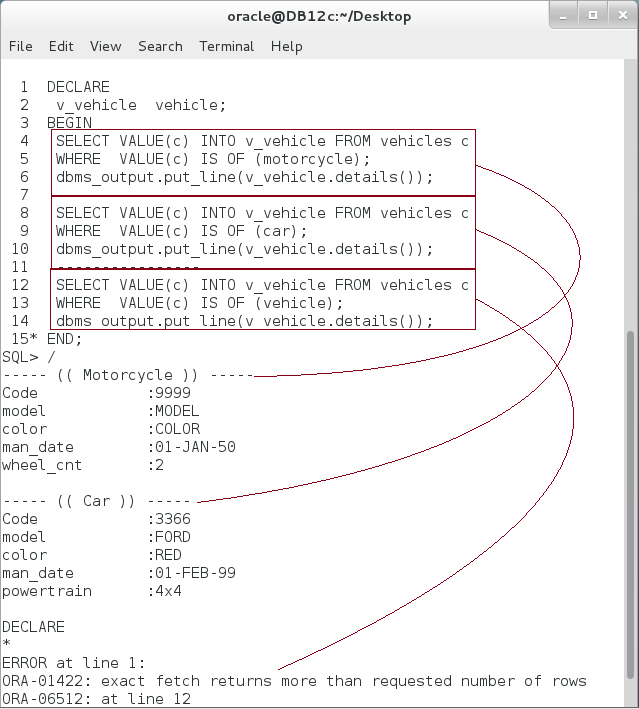
|  |  |
| --- | --- |
| Line | Description |
| SELECT **SYS\_TYPEID(**VALUE(c)**)** type, code FROM vehicles c |  |
| / |  |

****

**Please note:** SYS\_TYPEID function is used to distinguish between different Object Types stored in the same table.

**Step 9:** Finally, use "**IS OF (object\_type)**" expression to retrieve a specific Object Type out from the Object Table:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_vehicle vehicle; |  |
| BEGIN | Get MOTORCYCLE objects |
| SELECT VALUE(c) INTO v\_vehicle FROM vehicles c |
| WHERE **VALUE(c) IS OF (motorcycle);** |
| dbms\_output.put\_line(v\_vehicle.details()); |
| ---------------- |  |
| SELECT VALUE(c) INTO v\_vehicle FROM vehicles c | Get CAR objects |
| WHERE **VALUE(c) IS OF (car);** |
| dbms\_output.put\_line(v\_vehicle.details()); |
| ---------------- |  |
| SELECT VALUE(c) INTO v\_vehicle FROM vehicles c | Get VEHICLE objects. |
| WHERE **VALUE(c) IS OF (vehicle);** |
| dbms\_output.put\_line(v\_vehicle.details()); |
| END; |  |
| / |  |

 **Please note:** The last query failed. **Why?**

# SUMMARY

Object Type can very useful to represent a coherent and robust business layer in database layer. Some application developers see it as a replacement of table-relational database. Other used Object-relational database in conjunction with Table-Relational database. In this chapter, you examined using an Object Type to hold the implementation of a user-defined aggregation function. You also explored using Object Type in bulk SQL and bulk binding. Oracle supports using the concept of "Abstract Class" as other OOP languages do. Finally, you investigated how Oracle stored Object hierarchy in a regular table. Since Oracle stores the parent's attributes in columns and child's attributes in hidden columns.

After completing this lab exercise, you should be able to Object Type in specialized situation; Bulk SQL, Abstract Object, and User-Defined Aggregation Funciton .

# REFERENCES

* https://docs.oracle.com/database/121/ADOBJ/adobjint.htm#ADOBJ001
* http://docs.oracle.com/cd/B28359\_01/appdev.111/b28425/aggr\_functions.htm
* http://www.oracle-developer.net/display.php?id=215
* http://www.codeproject.com/Articles/170713/Custom-Aggregates-in-Oracle
* https://docs.oracle.com/database/121/ADOBJ/adobjadv.htm#ADOBJ7390

# INDEX

Abstract Object 1, 4, 9, 32

User-Defined Aggregation Funciton 32