INTRODUCTION

Object-oriented programming is widely used and has developed into an approach to database and application design that models the real world more closely. This approach combines data types and applicable operations (program units) into the data structure or object. Oracle’s object-type model is similar to the class mechanism in C++ and Java. With Oracle objects, entities such as a customer or an order are represented as an object containing attributes and methods. Attributes are similar to columns, and methods are similar to program units. The data structure and programming actions related to specific data are stored together as a unit called an “object type.” Many developers think this approach has advantages, such as standardization of data and tighter control of program units, as they’re associated with the data their actions are intended for.

This topic is extensive. However, as a PL/SQL developer, you should be aware of Oracle’s object technology features because they can make modeling business operations easier and make program development more efficient because objects can be reused and code design can be simplified. Object technology is gaining in popularity, so you’ll most likely encounter code using this approach.

NOTE

This appendix isn’t meant to be an introduction to object-oriented programming concepts, such as inheritance. It’s meant to give you an overview of object technology in Oracle, including object types, methods, and views. In other words, you learn about the mechanisms used to build an object-oriented application.
CREATING OBJECT TYPES

The Brewbean’s database stores a number of addresses, such as shipping, billing, and employee addresses. An object type can be created to contain address components and be used as the data type of each column in a database table storing address data. Creating a single object type to represent all address data ensures consistency for every address in the database. The following code creates an object type named ADDR_OT, which contains five attributes: street1, street2, city, state, and zip.

```sql
CREATE OR REPLACE TYPE addr_ot AS OBJECT
  (street1 VARCHAR2(25),
   street2 VARCHAR2(25),
   city VARCHAR2(25),
   state CHAR(2),
   zip NUMBER(9));
```

The ADDR_OT object type is used as the data type for the billing address and shipping address columns of the BB_ORDER table. Object types used as table columns are considered persistent because they’re stored in the database. Object types can also be used as data types for PL/SQL variables, which are considered transient because they exist only for the duration of the PL/SQL block.

The following code creates the BB_ORDER table, using the ADDR_OT object type for the billing address and shipping address columns:

```sql
CREATE TABLE bb_order
  (ord_id NUMBER(4),
   cust_id NUMBER(4),
   ord_date DATE,
   total NUMBER(6,2),
   bill_addr addr_ot,
   ship_addr addr_ot);
```

USING AN OBJECT TYPE

Now you have columns in the BB_ORDER table that actually contain five data elements. So how do you insert data into this structure? When an object type is created, a default constructor is created automatically. It’s a method for creating an object of the named type. For example, if you need to add an order to the BB_ORDER table, you would call the ADDR_OT constructor for the BILL_ADDR and SHIP_ADDR columns to match provided data values to each attribute in the object type.

Go ahead and create the object type in the following steps so that you can see how it’s used. Next, you’ll insert data in the object type by using the constructor.

1. In SQL Developer, use both code blocks in the previous section to create the ADDR_OT object type and the BB_ORDER table.
2. Now you can use the constructor to insert data in the object type columns. Enter the following code. The first two assignment statements use the constructor to create a variable containing data for each attribute of the ADDR_OT object type.
DECLARE
    lv_bill addr_ot;
    lv_ship addr_ot;
BEGIN
    lv_bill := addr_ot('11 Bush Dr',NULL,'Savannah','GA',346668229);
    lv_ship := addr_ot('812 Scott Lane','Apt #52','Savannah','GA',346668227);
    INSERT INTO bb_order
    VALUES (102,31,'11-NOV-2012',34.50,lv_bill,lv_ship);
END;

TIP
The constructor can be called directly in an INSERT statement; however, creating variables to hold these values makes the INSERT statement more readable.

3. Now that you have data in an object type column, how do you retrieve it? First, enter the SELECT statement shown in Figure F-1. Notice that the object type and a list of all attribute values for that column are presented as a group.

FIGURE F-1 A basic SELECT statement on an object type column

4. What if you want to retrieve a single value from the object type column? The query must reference the value with a table prefix and column prefix (see Figure F-2). The table name must be used even if there’s only one table in the query statement, or an error will result.

FIGURE F-2 Retrieving single attribute values correctly
5. In reviewing a table’s structure, what if you want to view its attributes? A basic `DESCRIBE` command lists only the column with the object data type, not the details of the object type column. You need to enter a `DESCRIBE` command on the object type, as shown in Figure F-3.

![Figure F-3 Using a DESCRIBE command](image)

**NOTE**
If you’re using SQL*Plus, you can enter a `DESCRIBE` command with the `DEPT ALL` option to view the full structure details.

**OBJECT METHODS**

Beyond data standardization, another reason for using object types is that it allows storing data-specific procedures and functions in the object type, which forms a bond with the data and associated program actions. For example, Brewbean’s wants a method associated with the address object type that formats addresses to print in a label style. The program units stored in an object type are called methods (even though the code might specify a function) and are invoked by using the object and method names in dot notation.

Adding a program unit to an object type is similar to setting up packages. First, you alter the object type to contain a specification for a function. Second, you create the body containing the function’s full code. Because the `ADDR_OT` object type already exists, you use the `ALTER TYPE` command to add a method or program unit to the object type. The `MEMBER` keyword is used to identify each method, followed by a program unit specification. In the following steps, alter the `ADDR_OT` object type you created earlier to add a method:

1. In SQL Developer, enter the following `ALTER TYPE` statement to add the `LBL_PRINT` method. The `CASCADE` option is used because the `ADDR_OT` type has dependent objects. (The `BB_ORDER` table uses this object type, too.)

```
ALTER TYPE addr_ot
ADD MEMBER FUNCTION lbl_print
RETURN VARCHAR2 CASCADE;
```
2. Next, enter a DESCRIBE command on the ADDR_OT object type to confirm that the method has been added.

3. To establish the function, use the CREATE TYPE BODY command shown in Figure F-4. (Notice that it's similar to the PACKAGE BODY code.)

```
CREATE TYPE BODY addr_ot AS OBJECT TYPE
  MEMBER FUNCTION lbl_print
    RETURN VARCHAR2;
  BEGIN
    lv_lbl VARCHAR2(100);
    BEGIN
      IF street2 IS NOT NULL THEN
        lv_lbl := street1||chr(10)||
          street2||chr(10)||
          city'||||state'||||'---'
          SUBSTR(zip,1,5)||'||SUBSTR(zip,6,4);
      ELSE
        lv_lbl := street1||chr(10)||
          city'||||state'||||'---'
          SUBSTR(zip,1,5)||'||SUBSTR(zip,6,4);
      END IF;
      RETURN lv_lbl;
    END;
  END;
```

The CREATE TYPE BODY command must reference an existing object type, just as a package specification must be referenced in the package body.

**FIGURE F-4**  The CREATE TYPE BODY command

4. Now that the object type includes a method, how do you use it? Try a simple SELECT statement on the SHIP_ADDR column, using the LBL_PRINT method as shown in Figure F-5. The method name is followed by a set of parentheses to instruct the system to look for a method with this name in the object.

```
SELECT o.ship_addr.lbl_print()
FROM bb_order o;
```

The SELECT statement must include the table alias, column name, and method in dot notation.

**FIGURE F-5**  Calling the LBL_PRINT method
OBJECT RELATIONS

Creating a database based on an object relational model typically involves using tables of object types. Object types are created to represent all the necessary data structures, and then tables are built based on these object types. Try approaching the Brewbean’s customer and order entities in this manner. First, you construct an object type to reflect the customer data and then create a table based on this object type. Next, you add the ORDER table; a one-to-many relationship between the customer and ORDER table based on customer ID needs to be established. In traditional relational database tables, you would use a foreign key constraint to enforce these relationships. Object types use REF variables to establish relationships, and these variables act more as pointers to rows in another table. To create an object relational design, follow these steps:

1. In SQL Developer, enter the following code to create an object type to represent customer data:

```sql
CREATE TYPE cust_ot AS OBJECT
    (cust_id NUMBER(4),
     first VARCHAR2(15),
     last VARCHAR2(20),
     email VARCHAR2(25),
     phone NUMBER(10));
```

2. Now enter the following code to create the customer table based on this object type. The PRIMARY KEY clause establishes a row uniqueness constraint based on an attribute of the object type.

```sql
CREATE TABLE bb_cust OF cust_ot
    (PRIMARY KEY (cust_id));
```

3. Enter the following code to create an object type for the order data that establishes a relationship with the customer data. The CUST_REF column is created with a REF data type, which references the CUST_OT object type. The customer table was created with the primary key CUST_ID, and the REF column looks for this primary key to establish the relationship.

```sql
CREATE TYPE ord_ot AS OBJECT
    (ord_id NUMBER(4),
     cust_ref REF cust_ot,
     ord_date DATE,
     total NUMBER(6,2));

CREATE TABLE bb_ord OF ord_ot
    (PRIMARY KEY (ord_id));
```

**NOTE**
As you know, Brewbean’s stores more data attributes for customers; however, these examples use only a few attributes to illustrate the concepts.
4. Next, test how the REF column works by entering some data in the customer and order tables with the following INSERTs. The REF operator needs a table alias that references the table you’re using to establish a relationship. The FROM clause identifies this table, and the WHERE clause tells the REF operator which row should be used for the relationship or correlation variable.

```sql
INSERT INTO bb_cust
VALUES
  (cust_ot(12,'Joe','Cool','jcool@yahoo.com',7773335555));

INSERT INTO bb_ord
SELECT ord_ot(1,REF(c),SYSDATE,24.50)
FROM bb_cust c
WHERE cust_id = 12;
COMMIT;
```

5. Query the order table, as shown in Figure F-6, to see how the REF column appears.

![FIGURE F-6](image)

6. You can query attributes of the customer object type by using the table name, REF column name, and attribute name in dot notation. The following code, for example, displays a customer’s last name:

```sql
SELECT ord_id, o.cust_ref.last
FROM bb_ord o;
```

**REF POINTERS VERSUS FOREIGN KEYS**

As you work with object types, keep in mind that REF pointers and foreign keys behave quite differently when managing relationships. If you create a foreign key constraint between the customer and order tables based on the customer ID, an error is raised if you try to delete a customer record referenced in the order table. However, a REF pointer works differently. To see how it works, experiment with a DELETE action in the following steps:

1. Delete the data in the BB_CUST table with the following statements. Notice that the DELETE action is accepted even though the BB_ORD table references the customer data.

```sql
DELETE FROM bb_cust;
COMMIT;
```
2. Query the BB_ORD table, as shown in Figure F-7, to confirm that a NULL value is now shown for the customer last name.

![Figure F-7 Confirming a deletion](image)

3. REF pointers do have a feature called a “dangling check” for showing when relationships are being broken. It specifies when the REF pointer has no associated row. To check whether the IS DANGLING condition is true for the row in the order table corresponding to the customer row you deleted, enter the query in Figure F-8.

![Figure F-8 Checking the CUST_REF column for a lost REF value](image)

**NOTE**

Figure F-8 shows the value 1 returned. The IS DANGLING condition returns a 0 to indicate FALSE or a 1 to indicate TRUE.

**TIP**

Even though the REF pointer allows deleting a parent record, this situation is usually prevented by using a BEFORE DELETE trigger to check whether child records exist.

**OBJECT VIEWS**

With Oracle’s object technology, you can use an object-oriented approach to designing a database. However, what about development shops that use the traditional Oracle relational database construction to support existing applications? For example, Brewbean’s has
already invested a lot of time to build the database with a traditional relational design. Can its developers take advantage of Oracle’s object technology features? Good question! They can, in fact, introduce object technology features into existing databases by using object views, which create a layer on top of the relational database so that the database can be viewed in terms of objects. Object views are the key to creating object-oriented applications without modifying existing relational database schemas.

Return to the original Brewbean’s database containing the BB_SHOPPER and BB_BASKET tables. These tables were created with the CREATE TABLE statement, which includes establishing a foreign key constraint based on the SHOPPER ID column. You can use an object view to represent these tables in an object structure. Figure F-9 shows the components involved in this process. This view maps to base tables (just like views you have created before); however, it also places data in object form by associating object types with the data being retrieved. Therefore, you need to create the object types before creating the object view.

The object view is based on the shopper and must reflect that each shopper can have many baskets because a foreign key established a one-to-many relationship between these tables. The CREATE VIEW statement uses the following operators:

- **OBJECT OID** indicates which column (typically the primary key column) in the base table provides a unique identifier for each object created.
- **CAST** allows a subquery to provide values for a nested table column in the view. The selected column list used in the CAST expression must match the attribute list of the object type used for this column.
- **MULTISET** enables the CREATE VIEW statement to handle multiple rows returned from the subquery in the CAST expression instead of just a single row.
Take a look at the following script that establishes the object view for the BB_SHOPPER and BB_BASKET tables. The first statements create object types to represent the table data in the object view. A table of the BASK_OT object type is created to use as an attribute in the SHOP_OT object type to reflect that many baskets can be associated with a single shopper.

```sql
CREATE TYPE bask_ot AS OBJECT
   (idBasket NUMBER(5),
    total NUMBER(7,2))
/
CREATE TYPE bask_tab AS TABLE OF bask_ot
/
CREATE TYPE shop_ot AS OBJECT
   (idShopper NUMBER(4),
    last_name VARCHAR2(20),
    city VARCHAR2(20),
    idBasket bask_tab)
/
CREATE VIEW shop_vu OF shop_ot
   WITH OBJECT OID(idShopper)
   AS
   SELECT s.idShopper, s.lastname, s.city,
          CAST(MULTISET(SELECT idBasket, total
                       FROM bb_basket b
                       WHERE b.idShopper = s.idShopper)
          S bask_tab)
       FROM bb_shopper s;
```

The OF SHOP_OT portion in the CREATE VIEW statement associates an object type with the view. Each object's unique identifier is the idShopper value, as specified in the OBJECT OID expression. The CAST and MULTISET expressions provide the values for the last attribute of the SHOP_OT object type, which is the table of baskets. All the operations on the data can now be handled through the object view to treat the database in an object-oriented fashion. Figure F-10 shows a simple SELECT statement on the object view to display the values returned for an object. Notice that shopper 21 has two baskets.

![FIGURE F-10 Querying an object view](image)
SORTING AND COMPARING OBJECT TYPE COLUMNS

Sorting and comparing simple data, such as scalar data types, is fairly straightforward. However, these tasks are a bit more challenging when more complex data types, such as object types, are introduced. Keep in mind that an object type can contain a variety of attributes. Figure F-11 shows what happens when you try to use an ORDER BY clause on an object type column in a query.

Two special methods of object types—ORDER and MAP—are available to sort and compare object type columns. An object type can contain only one of these methods. (Only the MAP method is described in this section to introduce you to this concept.) Follow these steps to use the MAP method to sort object type columns:

1. First, enter the following code to add data to the BB_ORDER table so that you have two rows to be sorted:

   DECLARE
   lv_bill addr_ot;
   lv_ship addr_ot;
   BEGIN
   lv_bill := addr_ot('55 Ulger Dr',NULL,'Chesapeake','VA',444668229);
   lv_ship := addr_ot('55 Ulger Dr',NULL,'Chesapeake','VA',444668229);
   INSERT INTO bb_order
   VALUES (103,38,'11-NOV-12',24.50,lv_bill,lv_ship);
   END;

2. Verify the data in the BB_ORDER table by using the query in Figure F-12.

FIGURE F-11 A sorting error on an ADDR_OT object type column
3. Next, alter the ADDR_OT object type to add a MAP method specification, as shown in the following ALTER TYPE statement:

```
ALTER TYPE addr_ot
ADD MAP MEMBER FUNCTION mapping RETURN VARCHAR2 CASCADE;
```

4. Add the MAP method to the TYPE BODY, as shown in Figure F-13, by using a CREATE OR REPLACE TYPE BODY statement. In this case, Brewbean’s decided to sort address fields based on state and city.
5. Enter the query shown in Figure F-14 to confirm that the ORDER BY clause is successful because of adding the MAP method to the object type.

```
SELECT ord_id, o.ship_addr.state, o.ship_addr.city
FROM bb_order o
ORDER BY o.ship_addr DESC;
```

**FIGURE F-14** Sorting an object type column

**SUMMARY**

Object types give you a mechanism to knit the data structure and associated program units into a single object. Incorporating object types, methods, and REF pointers enables you to use an object technology design approach in Oracle applications.