The Oracle Magician

March, 2004 *Volume III, number 1*

From the Editor

Welcome to the fifth issue of the magazine, The *Oracle Magician*! This occasional newsletter focuses on various "tricks of the trade" in the Oracle world--from DBAs, architects, developers, designers, and report writers.

Thank you to the many notes and suggestions from readers. They are most appreciated!

In this issue we present an overview of the recent Oracle feature, "Object Types." I suspect that many DBAs are confused by Object Types; I hope this article helps clear things up.

As noted in this issue, the Apress edition of my book, "The Art and Science of Oracle Performance Tuning," is now available in bookstores.

We are also looking for writers to submit articles that explore interesting ideas of use to DBAs or designers.

Please send all ideas to Editor@OracleMagician.com

Chris Lawson

Editor

Oracle Object Types: How to Navigate through the Maze

By Chris Lawson

In recent years, Oracle has begun to include more object-oriented features in each new database release. One of the most significant new features is called "object type." You have probably heard about object types, but what good are they?

An object type is quite different than the usual sort of things we typically see in a relational database. In this article we provide an introduction to this new set of features.

Introduction to Oracle Object types

An object type is not at all like a table or an index. The new "thing," or data construct, that we are creating is a *type*—which really is a *definition*, not something like a table or index that has storage. Just as the data types *number* or *varchar2* don't need any storage, our object types don't need any storage. So, even though the syntax used to create an object type may look like we're creating a table, we won't be specifying any sort of storage information.

Each object type has attributes that define the structure of our new object type. Additionally, we usually include an associated procedure or function, which is called a "method." The idea is to bring the data and the program together, so that we manipulate the data only via the methods.

A Simple Example: Apartment-Renting

Let's begin with a simple example. In order to illustrate object types, let's model the business of apartment

Continued on page 2

Table of Contents

Continued from page 1

rentals. We'll assume that the manager of the apartment building, having just returned from an Oracle class, wants to model his business in an object-oriented fashion using Oracle's object types. Let's begin by creating a new object type for the person who rents an apartment:

```
Create or replace type Renter AS
OBJECT (
First_Name        Varchar2(30),
Middle_Name       Varchar2(30),
Last_Name         Varchar2(30),
Drivers_License   Varchar2(20),
Phone             Varchar2(10));
/
```

An object type is quite different than the usual sort of things we typically see in a relational database

We call the parts of the object type *attributes*. So our new Renter object type has five attributes. Notice also the slash mark at the end of the code above. You cannot simply put the usual semicolon at the end, as you do when creating a table. We won't show the slash from now on, but remember that you'll need to include it.

When we run the above code, we are really creating an object *template*. We simply store a definition in the database, without any storage information. If you query the view User_Objects, you will see the object Renter listed. It will have an object type of "TYPE." On the other hand, if you query User_Segments, there will be no entries for this new object type, since no storage is required.

Some more object types

The apartment manager quickly points out that we have forgotten to model the most important thing—the apartments themselves! Clearly, we need another object type. Let's call it simply Apt:

```
Create Or Replace Type Apt As
Object (
Apt_Number      Number,
Bedrooms        Number,
Rental_Price    Number,
Lease_Expire    Date );
```

So far, we haven't defined any "methods"—the functions or procedures used to access or modify the data in our model. Our chats with the landlord revealed that he often needs to know when the apartment will be available for a new renter. The landlord tells us that he needs 30 days following lease expiration. Let's modify our Apt definition to include a method to retrieve that information. Our method will actually be a function:

```
Create Or Replace Type Apt As
Object (
Apt_Number      Number,
Bedrooms        Number,
Rental_Price    Number,
Lease_Expire    Date,
Member Function Apt_Ready return
Date);
```

Now let's define the actual body for our method. It's very similar to the steps for creating a regular PL/SQL function. Our sample method just finds the lease expiration date and adds 30 days:

```
Create Or Replace Type Body Apt As
  Member Function Apt_Ready Return
Date is
  Ready date;
  Begin
  Ready  := Lease_Expire + 30;
  Return Ready ;
  End Apt_Ready;
End;
```

Continued on page 3

Continued from page 2

Well, we have succeeded in creating a few object types, but what do we do with them? As they exist now, we can reference Apt or Renter, but we can't really store any useful information.

Although some applications may build object types just for modeling purposes, our apartment renting business really needs to store some data. This brings us to a new type of object, an *object table*.

Making data persistent with object tables

One easy way to store data using our object types is to build a new object called an *Object Table.* An object table is a table that matches a certain object type. It will hold the objects, and will show up as a "real" table, having storage. Let's build an object table based on our Apt object type:

Create table Apt_Info of Apt;

If we describe our new table, we see that the column definitions match perfectly the attributes of our object type Apt. **See Figure 1, below.**

We can insert rows just like a regular table, and query just like a regular table. Let's go ahead and insert data for apartments 201 through 203. **See Figure 2, below.**

If we wished, we could now query the Apt_Info object table just like a regular table. Let's try a query using our new function called "Apt_Ready." Remember that this method is supposed to add 30 days to the lease expiration.

To use a method, you have to use and reference the table alias, like this:

Select A.Apt_Number,
A.Apt_Ready() READY
from Apt_Info A;

Continued on page 4

```
SQL>  desc Apt_Info

 Name                                      Null?    Type
 ----------------------------------------- -------- -------
 APT_NUMBER                                         NUMBER
 BEDROOMS                                           NUMBER
 RENTAL_PRICE                                       NUMBER
 LEASE_EXPIRE                                       DATE
```

Figure 1: Sample Object Table based on Apt

```
Insert into Apt_Info values (201, 2, 795.00,to_date('01-FEB-2004', 'DD-MON-YYYY'));
Insert into Apt_Info values (202, 2, 895.00,to_date('01-JAN-2004', 'DD-MON-YYYY'));
Insert into Apt_Info values (203, 1, 695.00,to_date('01-JUN-2004', 'DD-MON-YYYY'));
```

Figure 2: Inserting data into our new Object Table

Continued from page 3

APT_NUMBER	READY
201	02-MAR-04
202	31-JAN-04
203	01-JUL-04

Besides the regular way of querying an object table, Oracle provides an alternate method. Remember that an object table is really based on an object type, which just happens to have multiple values. Thus, Oracle provides an alternative way to view these tables, in which the results look like a single column. This alternate method is shown in **Figure 3**, below.

Note that in this alternative method of querying the object table, the results are formatted as though the table had just a single column.

Nested Object Types

Just as we assign a data type to a column in a table, we can assign an object type to a column. This is called an nested object type. Let's create a new table, and include both the Renter and Apartment information in this table:

```
Create table Apt_Business (
Line_Item Number,
Apt_Data      Apt,
Renter_Data   Renter)
```

If we describe this new table, we can see our new object types. See **Figure 4,** below.

Observe that our two object types are indeed the "data types" for the apartment and renter columns. So the column *Apt_Data* is not just a number or a string of characters; instead, it contains a lot of information, as defined by the Apt object type. Similarly, the column

Continued on page 5

```
Select Value (A)  from Apt_Info A
Where Apt_Number = 202;

VALUE(A)(APT_NUMBER, BEDROOMS, RENTAL_PRICE, LEASE_EXPIRE)
-----------------------------------------------------------
APT(202, 2, 895, '01-MAR-04')
```

Figure 3: Alternate Method of Selecting from an Object Table

```
Desc Apt_Business

 Name                                      Null?    Type
 ----------------------------------------- -------- -----------
 LINE_ITEM                                          NUMBER
 APT_DATA                                           APT
 RENTER_DATA                                        RENTER
```

Figure 4: New Table References Object Types

Continued from page 4

```
Insert into Apt_Business values
(1,  APT(202, 2, 895,  '01-FEB-2004'),
Renter('Joe', 'X', 'Smith', 'CDL92569800', '4158889900'));
```

Figure 5: Inserts using two Constructors

Renter_Data, which is based on our object type Renter, contains the renter's firstname, lastname, phone, etc.

Using Constructors

When we access or manipulate data in objects, we will use a "constructor." The constructor has the same name as the object type. The parameters that we use will match the attributes of the object type.

To see how this works, let's put some data into our new *Apt_Business* table. We will use the number 1 for the first field. The second "column" is really the Apt object type. The third column is the Renter object type. In order to insert data into the special "columns," we will use the constructors *APT* and *RENTER*. Figure 5, above, shows how it looks:

Wrap Up

With these examples, we have only touched on the basics of Oracle object types. There are many more features available—some of them truly bewildering in their complexity. You will probably run into Oracle object types soon, so you might as well get used to them!

The Oracle Magician

The OracleMagician Newsletter

Editor: Chris Lawson

New edition of The Art & Science of Oracle Performance Tuning!

The Apress edition of *The Art and Science of Oracle Performance Tuning* is now in book-stores.

Thanks to Publisher Apress for the nice work on this edition, shown below.