Introduction:

Oracle8 introduced Object-Relational Database (ORDB) Management System as extension of purely relational database (RDB) system of Oracle7. In Oracle 10g objects use have been further enhanced. As the name implies, in this implementation the user designs the database using the relational model but incorporates several object-oriented features into the design. The object type definition is accomplished through the `create type` command. Two collection objects varying arrays (allowing storing multiple values for an attribute within a row) and nested relations (tables within tables) are introduced in the following.

Example: The mail-order database (see the attached database). Assume that the address attribute is present in both the `customers` and `employees` tables. A good way to model this situation is to create an object type and include attributes of this type in the `customers` and `employees` table.

In procedural languages one uses system-defined data types and creates new data structures (struct). In Object-oriented programming, the user defines classes, which defines data types and methods. In Oracle, the `create type` command allows object type (class) definition.

1. **Step 1**: create `address_type` object as follows:

   ```sql
   create type address_type as object |
   struct varchar2(30),
   city varchar2(30),
   state varchar2(30),
   zip number(9); |
   / 
   ```

   DBMS_OUTPUT:
   ------------
   0 record(s) affected

   [Executed: 2/8/2015 10:10:21 AM] [Execution: 302ms]

2. **Step 2**: create `phones_varray_type` as collection type, as follows:

   ```sql
   create type phones_varray_type as varray(5) of char(12); |
   / 
   ```

   DBMS_OUTPUT:
   ------------
   0 record(s) affected

   [Executed: 2/8/2015 10:10:21 AM] [Execution: 40ms]
3. **Step 3**: create person_type object as follows:

```sql
create type person_type as object {
    name varchar2(30),
    address address_type,
    phones phones_varray_type);
```

---

Creating Relations: The `o_employees` is created as follows:

```sql
create table o_employees {
    eno number(4) not null primary key,
    person person_type,
    hdate date
};
```

---

Create the `o_customers` and `o_parts`, in a similar fashion as show:

```sql
create table o_customers {
    cno number(5) not null primary key,
    person person_type
};
create table o_parts {
    pno number(5) not null primary key,
    pname varchar2(30),
    qoh integer check(qoh >= 0),
    price number(6,2) check(price >= 0.0),
    olevel integer
};
```
Inserting Tuples:

The first set of insertions are performed on the relation `o-employees`:

```sql
insert into o_employees values (1000,
    person_type('Jones'),
    address_type('123 Main St.', 'Wichita', 'KS', 67226),
    phones_varray_type('316-555-1212', '316-555-2323', null, null),
    '20-NO7-00')
```

```
DBMS_OUTPUT:
--------
1 record(s) affected
[Executed: 2/3/2015 11:21:32 AM] [Execution: 60ms]
```

Continue with the insertions and check the inserted as shown here:

```sql
insert into o_employees values(1001,
    'Smith',
    '100 Elm St.', 'Fort Dodge', 'IA', 50500,
    phones_varray_type('316-666-1212', '316-666-2323', null, null),
    '11-OC7-00')
```

```
/  
insert into o_employees values(1001,
    'Smith',
    '100 Elm St.', 'Fort Dodge', 'IA', 50500,
    phones_varray_type('316-666-1212', '316-666-2323', null, null),
    '11-OC7-00')
```

```
/ 
select * from o_employees
```

```
/ 
drop table o_employees
/
select * from o_employees
/ 
```

```
DBMS_OUTPUT:
--------
3 record(s) selected [Fetchaison One] [Fetchaison One]
```

This is followed by insertions into the relation `o_customers`, as follows:

```sql
insert into o_customers values
(1111, 'Charles', '123 Main St.', 'Wichita', 'KS', 67226,phones_varray_type('316-699-5555', '316-699-5555', null, null))

insert into o_customers values

insert into o_customers values
(3333, 'Barbara', '111 Inwood St', 'Fort Dodge', 'IA', 50504,phones_varray_type('316-111-1234', '316-111-1234', null, null))
```

Performing Queries:

The following two queries demonstrate performing such search operations in ORDB systems:

```sql
Find the names and street addresses for employees hired after February 01, 2000.

```sql
from o_employees e
where e.hiredate > '01-FEB-00'
```
Here, `e.person.address.street` refers to the `street` attribute defined within the `address` attribute, which is defined within the `person` attribute of the `o_employees` table. The dot notation is used to access the inner-most basic attributes of a complex object structure. Note that table aliases are required when table definitions are based on object types.

**Find the names of employees and customers who live in the same zip code.**

```sql
select e.person.name EMPLOYEE, c.person.name CUSTOMER
from o_employees e, o_customers c
where e.person.address.zip = c.person.address.zip
```

---

**DEMS_OUTPUT:**

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>Charles</td>
</tr>
<tr>
<td>Jones</td>
<td>Bertran</td>
</tr>
<tr>
<td>Smith</td>
<td>Barbara</td>
</tr>
</tbody>
</table>

3 record(s) selected [Fetch MetaData: 0ms] [Fetch Data: 0ms]

[Executed: 2/8/2015 1:30:18 PM] [Execution: 30ms]

This is join operation of the two tables `o_employees` and `o_customers` based on the `zip` attribute which lies deep in the object structure.

**Find the pairs of customer number values of customers having the same zip code.**

```sql
select cl1.cno CUSTOMER#1, cl2.cno CUSTOMER#2
from o_customers cl1, o_customers cl2
where cl1.person.address.zip = cl2.person.address.zip
```

---

**DEMS_OUTPUT:**

<table>
<thead>
<tr>
<th>CUSTOMER#1</th>
<th>CUSTOMER#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2222</td>
<td>1111</td>
</tr>
<tr>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>2222</td>
<td>2222</td>
</tr>
<tr>
<td>1111</td>
<td>2222</td>
</tr>
<tr>
<td>3333</td>
<td>3333</td>
</tr>
</tbody>
</table>

5 record(s) selected [Fetch MetaData: 0ms] [Fetch Data: 0ms]

[Executed: 2/8/2015 1:30:50 PM] [Execution: 3ms]
What seems to be wrong with this query? How would you modify it? Here is the answer:

```
SELECT cl.cno CUSTOMER#1, c2.cno CUSTOMER#2
FROM o_customers cl, o_customers c2
WHERE cl.person.address.zip = c2.person.address.zip AND
cl.person.name != c2.person.name
```

Uploading & Deleting:

**Update** the street address of the employee named Green to ‘111 Elm St’.

**Delete** a particular employee tuple from the employees relation with the name being ‘Green’ or ‘Greek’.

Questions:

a. How do you know what objects have been created?
b. How do you know what collection types have been created?
User-defined methods:

When an object type is created, Oracle automatically creates methods to construct and access objects of that type. In addition to these system-defined methods, the user may specify other methods for an object type. The user-defined methods are created by specifying the method prototype in the create type statement and by completing the method implementation in the create type body statement.

Consider the order details information in the mail order database. The following creates an object with two attributes, pno and qty, along with a method, called cost. See ODETAILS object added to the browser list.
The last two lines define the method `cost` for this object type. The first line defines the member function prototype, a method with no parameters. The method could potentially modify the database state if it is invoked, which is neither desirable nor allowed if this method is to be used in an SQL query. Therefore, a `pragma` statement must be specified which restricts the method from updating the database state. Here are the available restrictions used in pragma statement:

**WNDS**: Write No Database State (no updating of database state allowed)
**RNDS**: Read No Database State (no query allowed)
**WNPS**: Write No Package State (no values of package variables are modified)
**RNPS**: Read No Package State (no package variables are referenced)

The body of this method is created through a PL/SQL file named `Order_details.sql`, and is prepared and executed as follows:

```sql
create or replace type body odetails_type as
member function cost return number is
  p o_parts.price%type;
begin
  select price into p from o_parts where pno = :p;
  return p * self.qty;
end;
end;
/
```

Notice the use of the `self` keyword, which allows the method to refer to the attributes of the object on which the method is applied.

Let’s insert a few tuples into `o_parts` relation, as follows:

```sql
select * from o_parts
```

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>QOH</th>
<th>PRICE</th>
<th>OLEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10506</td>
<td>Land Before Time I</td>
<td>200</td>
<td>19.99</td>
<td>20</td>
</tr>
<tr>
<td>10507</td>
<td>Land Before Time II</td>
<td>156</td>
<td>19.99</td>
<td>20</td>
</tr>
<tr>
<td>10508</td>
<td>Land Before Time III</td>
<td>190</td>
<td>19.99</td>
<td>20</td>
</tr>
</tbody>
</table>
PL/SQL Access to Oracle Objects

Declaring and Initializing Objects:

Objects can be declared and used in PL/SQL based on object types defined in the database. The structure of these objects can be traversed using the dot notation, objects can be modified and, if needed, the Database can be updated using the modified objects. In the following example, p is declared (and initialized) as a person_type object variable, which uses two already declared data types objects (address_type, and phones_varray_type). The following PL/SQL anonymous block creates a person_type object, initializes it, updates, updates its attributes, and calls a procedure to display the object.

```plsql
declare
  p person_type := person_type('aaa',
    address_type('sss', 'ccc', 'sss', 11111),
    phones_varray_type(null, null, null));
procedure display_person(p in person_type) is
begin
  if p is not null then
    dbms_output.put_line('Name: ' || p.name);
    dbms_output.put_line('Street: ' || p.address.street);
    dbms_output.put_line('City: ' || p.address.city);
    dbms_output.put_line('State: ' || p.address.state);
    dbms_output.put_line('Zip: ' || p.address.zip);
    if p.phones(1) is not null then
      dbms_output.put_line('Phone 1: ' || p.phones(1));
    end if;
    if p.phones(2) is not null then
      dbms_output.put_line('Phone 2: ' || p.phones(2));
    end if;
    if p.phones(3) is not null then
      dbms_output.put_line('Phone 3: ' || p.phones(3));
    end if;
  end if;
begin
  p.name := 'D. Maul';
p.address.street := '123 Kenobi St.';
p.address.city := 'Any City';
p.address.state := 'Any State';
p.address.zip := '65665';
p.phones(1) := '111-1234';
p.phones(2) := '111-1235';
p.phones(3) := null;
display_person(p);
end;
show errors
```
Object Type Method Implementation:

Methods associated with object types are specified in PL/SQL

```sql
create or replace type body o_details_type as
  member function cost return number is
    p o_parts.price + type;
  begin
    select price into p from o_parts where pno = self.pno;
    return p * self.qty;
  end;
end;
/
```

Another object Type Method Implementation:

```sql
create or replace type body o_order_type as
  member function total_cost return number is
    item o_details_type;
  total number := 0;
  item_cost number;
  begin
    for i in 1 .. self.odetails.count loop
      item := self.odetails(i);
      item_cost := item.cost();
      total := total + item_cost;
    end loop;
    return total;
  end;
end;
show errors
/
```
Nested Tables:

The second type of collection object supported by Oracle8 and used since then with enhancements, is the *nested table*. In contrast to varying arrays, which have a fixed maximum number of entries, nested tables can support an unlimited number of entries per row. The general syntax of the `create type` statement to create a nested table is:

```
CREATE TYPE type_name AS TABLE OF datatype;
```

Where *type_name* is the name given to the nested table and *datatype* is any valid Oracle8 basic data type or an object type.

In the mail order database example, a table object type consisting of many *odetails_type* objects:

```sql
SQL> desc odetails_type;
Name       Null? Type
---------- -------- ----
PNO                     NUMBER(5)
QTY                     NUMBER(38)
METHOD
-------
MEMBER FUNCTION COST RETURNS NUMBER
```

can be defined as follows:

```sql
SQL> create type odetails_nstable_type as
table of odetails_type;
Type created.
```

An attribute of this type can then be included in another object type, called *o_order_type*, which corresponds to the *orders* in the mail order database. This way, the order details for a particular order are logically associated with the order information itself rather than in a separate table as was the case in the relational design of the mail order database.

```sql
create type o_order_type as object{
  ono number(5),
  odetails odetails_nstable_type,
  cno number(5),
  eno number(4),
  received date,
  shipped date,
  member function total_cost return number,
  pragma restrict_references(total_cost,WNDS));
} /
```
The following shows the method implementation for total_cost.

```
create or replace type body o_order_type as
member function total_cost return number is
  i               integer;
  item            odetails_type;
  total           number :=0;
  item_cost       number;
begin
  for i in 1..self.odetails.count loop
    item := self.odetails(i);
    item_cost :=item.cost();
    total := total + item_cost;
  end loop;
  return total;
end;
end;
/
show errors
```

```
SQL> @total_cost.sql;
Type body created.
No errors.
```

```
SQL> desc o_order_type;
Name                            Null?    Type
-------------------------------  --------  ----
ONO                                      NUMBER(5)
ODETAILS                                  ODETAILS_NTABLE_TYPE
CNO                                      NUMBER(5)
ENO                                      NUMBER(4)
RECEIVED                                 DATE
SHIPPED                                  DATE

METHOD
-----
MEMBER FUNCTION TOTAL_COST RETURNS NUMBER
```

In the next module examples of implementation of the methods will be provided.
**Triggers:**

Like a stored procedure, a trigger is a named PL/SQL unit that is stored in the database and can be invoked repeatedly. Unlike a stored procedure, you can enable and disable a trigger, but you cannot explicitly invoke it. While a trigger is enabled, the database automatically invokes it—that is, the trigger fires—whenever its triggering event occurs. While a trigger is disabled, it does not fire.

Syntax for creating triggers:

```
create_trigger ::= 
```

![Description of the illustration create_trigger.gif](create_trigger.gif)

```
plsql_trigger_source ::= 
```

![Description of the illustration plsql_trigger_source.gif](plsql_trigger_source.gif)

Examples of triggers and their implementation in the next module.

**Advantages of PL/SQL:**

These are the Advantages of PL/SQL:

- **Block Structures:** PL/SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.

- **Procedural Language Capability:** PL/SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).

- **Better Performance:** PL/SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
Mail-Order Database:

<table>
<thead>
<tr>
<th>ENO</th>
<th>ENAME</th>
<th>ZIP</th>
<th>HDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Jones</td>
<td>67226</td>
<td>12-DEC-95</td>
</tr>
<tr>
<td>1001</td>
<td>Smith</td>
<td>69606</td>
<td>01-JAN-91</td>
</tr>
<tr>
<td>1002</td>
<td>Brown</td>
<td>50302</td>
<td>01-SEP-94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>QOH</th>
<th>PRICE</th>
<th>LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10506</td>
<td>Land Before Time I</td>
<td>200</td>
<td>19.99</td>
<td>~20</td>
</tr>
<tr>
<td>10507</td>
<td>Land Before Time II</td>
<td>156</td>
<td>19.99</td>
<td>~20</td>
</tr>
<tr>
<td>10508</td>
<td>Land Before Time III</td>
<td>100</td>
<td>19.99</td>
<td>~20</td>
</tr>
<tr>
<td>10509</td>
<td>Land Before Time IV</td>
<td>50</td>
<td>19.99</td>
<td>~20</td>
</tr>
<tr>
<td>10601</td>
<td>Sleeping Beauty</td>
<td>300</td>
<td>24.99</td>
<td>~20</td>
</tr>
<tr>
<td>10701</td>
<td>When Harry Met Sally</td>
<td>120</td>
<td>19.99</td>
<td>~30</td>
</tr>
<tr>
<td>10800</td>
<td>Dirty Harry</td>
<td>140</td>
<td>14.99</td>
<td>~30</td>
</tr>
<tr>
<td>10900</td>
<td>D. Zhivago</td>
<td>100</td>
<td>24.99</td>
<td>~30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CNO</th>
<th>CNAME</th>
<th>STREET</th>
<th>ZIP</th>
<th>PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Charles</td>
<td>123 Main St.</td>
<td>67226</td>
<td>316-636-5555</td>
</tr>
<tr>
<td>2222</td>
<td>Bertram</td>
<td>237 Ash Ave.</td>
<td>67226</td>
<td>316-689-5555</td>
</tr>
<tr>
<td>3333</td>
<td>Barbara</td>
<td>111 Inwood St.</td>
<td>68606</td>
<td>316-111-1234</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ONO</th>
<th>CNO</th>
<th>ENO</th>
<th>RECEIVED</th>
<th>SHIPPED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1120</td>
<td>1111</td>
<td>1000</td>
<td>10-DEC-94</td>
<td>12-DEC-94</td>
</tr>
<tr>
<td>1121</td>
<td>1111</td>
<td>1000</td>
<td>12-JAN-95</td>
<td>15-JAN-95</td>
</tr>
<tr>
<td>1122</td>
<td>2222</td>
<td>1001</td>
<td>13-FEB-95</td>
<td>20-FEB-95</td>
</tr>
<tr>
<td>1123</td>
<td>3333</td>
<td>1000</td>
<td>20-JUN-97</td>
<td>null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CNO</th>
<th>PNO</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1120</td>
<td>10506</td>
<td>1</td>
</tr>
<tr>
<td>1120</td>
<td>10007</td>
<td>1</td>
</tr>
<tr>
<td>1120</td>
<td>10906</td>
<td>2</td>
</tr>
<tr>
<td>1120</td>
<td>10209</td>
<td>3</td>
</tr>
<tr>
<td>1121</td>
<td>10601</td>
<td>4</td>
</tr>
<tr>
<td>1122</td>
<td>10601</td>
<td>1</td>
</tr>
<tr>
<td>1122</td>
<td>10701</td>
<td>1</td>
</tr>
<tr>
<td>1123</td>
<td>10800</td>
<td>1</td>
</tr>
<tr>
<td>1123</td>
<td>10900</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ZIP</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>67226</td>
<td>Wichita</td>
</tr>
<tr>
<td>66006</td>
<td>Fort Dodge</td>
</tr>
<tr>
<td>50302</td>
<td>Kansas City</td>
</tr>
<tr>
<td>54444</td>
<td>Columbia</td>
</tr>
<tr>
<td>69023</td>
<td>Liberal</td>
</tr>
<tr>
<td>61111</td>
<td>Fort Hays</td>
</tr>
</tbody>
</table>