The object model of Oracle is an under implemented and commercially unexplored flavor of the database. In the late 1980s, the relational model of Oracle faced several deficiencies, which forced the evolution of the object model. Undoubtedly, the relational model did well for flat data storage, maintenance of integral data, and information security but could not offer a concrete approach to accommodate complex data structures of nested, hierarchical, and extensible types. These subtle restrictions led to the adoption of OOP's concepts in database systems.

The object-oriented model of the database carries the essence of object-oriented concepts like inheritance, data instantiation and abstraction, and establishment of relationships among types. It ensures persistent data management, effective compatibility and integration with OOPs supported languages like C++ and Java, which are the best market flavors of application development.

The article shall be a fair effort to explore the object model of Oracle. The scope of the study shall include the evolutionary background, comparative drive with RDBMS, commercial study and a lot more details. I shall present a sample case study to highlight the structure building implementation and demonstrate the working of common data tasks.

1. Introduction

Even since the days of the evolution of Oracle as the master of commercial database solutions in the industry, it has distinguished itself in terms of modeling implementation, customer satisfaction, scalability and flexibility. The RDBMS concepts, or the relational model, matured with every fit, dominated and suited a maximum sort of data, which can be thought of in an application. Commercially and conceptually, it proved to be a great success unless the lacked support for relationships, extensible and derivative data structures were identified. In a highly performing scientific or manufacturing database model, the data are proclaimed to be less relational and highly extensible. The inability to model out such applications led to the induction of object oriented concepts in Oracle, in parallel to the relational engine. Though some of the OOP's concepts like encapsulation and polymorphism were actively exercised earlier in the Oracle relational model, one of the core OOP's concept of inheritance was still untouched.

Object orientation of data was a new, but different flavor for Oracle users. Here, the objects and their instances, being the model protagonists, are more talked about than tables and columns. Complex data structures which share ‘extends from’ relationship or have multi-level hierarchical structure were best candidates to fall under this category. Digging over the object model, it could efficiently do semantic data modeling, where reusability and extensibility are its essence. Professionals, in the current phase, are more comfortable with UML and OOP’s designs, where relational models are just another data storage layer.

Here, I shall outline the philosophy of OODBS (Object Oriented Database Management Systems) and show the evolution of ORDBMS (Object Relational Database Management Systems), the hybrid model of the object and relational model of the Oracle database. The scope of the paper covers the object-oriented concepts, the kick off implementation and prepares the reader to explore the next level.

2. Object Oriented Model: The Philosophy

On a broader sense, the object oriented model follows the principle ‘Every entity as an object has the tendency to extend from and extend to another entity’. Unlike the relational model which considers entity as an abstract unit to build a table, the object model perceives ‘entity’ as a physical object. In my personal observation, the definition of an entity in the object model looks closer to real terms, as it clearly defines its structure and behavior. Multiple instances of the same object type can reside in different database tables. In a single request, one can fetch the complete instance, which has a full set of object attributes. It is analogous to the ‘Classes’ in C++ or Java.

3. Object: High Level Design

An object is the basic unit in the Object Oriented Model (OOM) of Oracle. It is the base of the model, which can realize an entity. Technically, an object type has a set of attributes, which builds up its skeleton. It can be instantiated in multiple contexts to give a meaningful dimension of its behavior. Instantiation of an object can be understood as the role of a person at different places. A ‘person’ is a ‘student’ at ‘college’, ‘employee’ at ‘workplace’, ‘customer’ at a ‘shop’ and many others. As an entity, his behavior in a context is a determinant of his behavioral values, structure remaining the same.

Figure 1. Skeleton of an Object
Getting into the high-level design of an object, its structural and behavioral aspects are essential and considerable over its design.

### 3.1 Structural Aspects

Structural aspects include the identification of the relationship amongst the objects and their properties. In an object, its attributes demonstrate the structural aspect of the object.

The relationship can be classified into inheritance, association and aggregation.

Two objects share an ‘inheritance’ or ‘extends from’ relationship, when one (subtype) of them extends from the other (super type). In such case, the derived object inherits the structural and behavioral features of its parent. Later, we shall discuss the ‘inheritance’ relationship as the sole and soul feature of object oriented model.

Association relationship is when an object is associated to another as ‘one to one’, ‘one to many’ or ‘many to one’. This relationship can be best observed between objects and the application. Mere structure and property would not be sufficient to demonstrate such a relationship.

Aggregation relationships demonstrate the ‘part of’ relationship among the objects. An object can accommodate several other objects to become master objects, hence becoming a composite object. Note that it is different from a Parent-Child relationship as the member object is just a component of the master part. A single query to the composite object would be sufficient to fetch a bill of material of all the component objects. It can fall among the following categories:

- Existence dependent – subcomponent objects are dependent on the master object
- Nonexistence dependent – subcomponent objects are an independent entity, but accommodated by the master object
- Existence composition – subcomponent is an exclusive constituent object of the master object
- Nonexistence composition – subcomponent object is a component of the master object, which can be shared by other master objects too
- Homogeneous composition – subcomponents objects are of same type as that of the master object
- Multilevel (complex) composition – subcomponents objects involve many branching and level hierarchies

### 3.2 Behavioral Aspect

The behavior of an object can be knobbed in a situation by its member components. The skeleton of an object encapsulates attributes and methods. The type ‘specification’ carries only the method prototypes, while the type ‘body’ contains the definition of the methods. This ensures data hiding and prevents method algorithms to be public. The attributes build up the property structure of the object, while methods are the operational routines on object attributes. Implications are clear that attributes are part of the data layer while methods are part of the data access/interface layer. This is one of the major advantages of the object over database tables in that an object not only holds the data, but optionally can also have operations upon the data.

The methods can be generic or user defined. Generic methods are meant for accessing the attributes and basic operations like insert, update, delete, and query. User defined methods embed a defined business logic, which gets activated on the object instance.

### 4. Object Relational Database Management System: The Evolution

Over the years of trials and results, the relational model (RDBMS) has been an established and trusted data model, while the object-oriented database still needs the drive to catch up with the competition. Both have their own advantages and disadvantages. On one side, the relational model stands mum before inheritance, while on the other side, the object model lacks a universal data model and carries ‘tight coupling’. The object relational model is an extended version of the relational model which includes object orientation behavior. It offers an adhesive platform for the advanced database systems, which not only widens the scope of data fit by multiplying the features of both the models, but also adds to the trust of the customer to handle the real world application scenarios.

### 5. ORDBMS vs. RDBMS: A Comparative Study

By now, I believe we are familiar enough with objects to shoot a comparative drive between object relational and relational models of the Oracle database.

#### 5.1 Modeling Becomes Easier

The biggest plus with OODBMS is reusability, sharing, and the support of inheritance. It offers semantic modeling of real world problems, while RDBMS fails to model structured and extensible data. Heterogeneous data of a recursive nature demands inheritance implementation, which can be easily done with ORDBMS.

By real world problems, I mean the realization of an entity, hence the data, in an accessible form. Since ORDBMS considers an entity as an object, it requires less referential integrity, mapping and complex normalization paradigms.

For example, consider a ‘company’, which employs people at different levels like ‘director’, ‘manager’, ‘developer’, ‘admin’, and ‘HR’. All have a different set of structural and behavioral properties at each level. There are two related entities identified as ‘company’ and ‘employee’.

![Figure 2. EMPLOYEE entity realization in Object Model](image)

An object instance of ‘employee’ type is capable of holding all its descendents like director, manager, developer, admin and HR. But similar case implementation in the relational model would end up in the creation of separate tables for each level and a complex referential network. So clearly the implication from the case is that modeling of the application scenario becomes easy when seen through the object orientation frame.

#### 5.2 As a Performance Booster

Object models have been observed to give better performance than relational ones. A properly optimized object model would yield
1 to 5 times better performance while querying and inserting object instances.

Since the object model involves a high level of coupling factors, there are several other factors which determine the performance. They are transaction overhead, network configuration, object level locking, instance caching, and query optimization techniques.

5.3 Miscellaneous
Apart from the above listed benefits, the object oriented database system persistently stores the objects in the database. It is easy to integrate with OOP’s supportive languages like C++ and Java. Client developers can hold and access the object instances without breaking them into a relational format. This secures the data access methodology from the client.

6. Object Oriented Model: The Implementation
Until now, we have discussed the high-level design of an object. Now, we shall move on to the implementation part and learn how the objects are stored in the database, tables and their references.

We shall move ahead with the object-oriented glossary with the help of the case study, which has been earlier discussed. The ‘employee’ model as shown below,

![Image](image.png)

**Figure 3: Data model for the Employee Hierarchy in the Company**

The model represents the type evolution and hierarchy model of an organization. An employee can be a director, a manager or a developer. There are different sets of responsibilities defined at each level, which we shall try to model in the object-oriented methodology.

6.1. Object Type and Instance: The object type is a user defined object type and physically resides in the database as a schema object. Similar to classes, it can encapsulate attributes (mandatory) and methods (optional) to demonstrate its structural and behavioral aspect respectively. Object attributes can be of a primitive data type (number, varchar2, date) or any valid user defined type. Like the Oracle supplied types, an object can be used to state the data type of a column in a table, or an attribute in another object, or as a parameter. The object instance is a physical realization of the object. A variable of the object type behaves as its instance, which can store attribute values and invoke its methods.

For an object, extension and instantiation are two properties which can be defined at the type declaration level. ‘Extension allowed’ object types can extend themselves to a new object. ‘Instantiation allowed’ allows the creation of the object’s instance in the database.

From our case study ‘employee’ model, EMPLOYEE object can be created as below.

```
-----Creating EMPLOYEE object type specification
CREATE OR REPLACE TYPE EMPLOYEE_OT AS OBJECT
    (EmpID NUMBER,
     EmpName VARCHAR2(1000),
     EmpBirthDate DATE,
     EmpJoinDate Date,
     EmpSalary NUMBER,
     EmpLoc VARCHAR2(100),
     MEMBER FUNCTION GetEmpExp RETURN NUMBER
    )
NOT FINAL; ---‘employee’ object tends to extend
NOT INSTANTIABLE; ---‘employee’ object instance
    cannot be created
```

The attributes build up the structure of the object, while the only method ‘GetEmpExp’ demonstrates one of the traits of the object. The NOT FINAL clause creates an extendible object type. The NOT INSTANTIABLE clause restricts the creation of any instance of the object in the database.

6.2. Object Methods: Object methods can be member, static or constructor. Member methods are majorly operative on the object attributes. Static methods embed a generic logic, which does not involve the object attributes. Constructors are implicit methods created by the database for initialization and default instantiation purpose. It can be overridden with a user-defined constructor.

The type definition can be completed by defining its TYPE BODY as below.

```
-----Creating EMPLOYEE object type body
CREATE OR REPLACE TYPE BODY EMPLOYEE_OT IS
    MEMBER FUNCTION GetEmpExp RETURN NUMBER IS
        BEGIN
            RETURN SYSDATE-EmpJoinDate;
        END GetEmpExp;
    END;
/
```

```
-----Invoking the member method
SELECT T.GetEmpExp () FROM EMPLOYEES T /
```

6.3. Object Type Inheritance: This is one of the most distinguished features of the object model. It enables the creation of object type hierarchies. Type hierarchies can initiate from a generic type and extend incrementally to inherit the properties of its originator and accommodate specific properties at each derivative node. A node in this type hierarchy can behave either as a parent node or a child node or both. In technical terms, parent and child nodes are known as super-type and sub-type respectively.

The type’s structural changes can be achieved using the ALTER TYPE statement and can be cascaded across all the instances contained by all the tables in the database. If any dependency
check fails in the type hierarchy, the statement raises the appropriate exception and rolls back the operation.

In the ‘employee’ model, the ‘employee_ot’ object type can be extended to create objects for director, manager and developer as shown below.

Note that a type, which has the ability to extend, must be created with the NOT FINAL option.

-------Creating DIRECTOR object sub type under EMPLOYEE
CREATE OR REPLACE TYPE DIRECTOR_OT UNDER EMPLOYEE_OT

 DirBusinessUnit VARCHAR2(100),
 DirTarget NUMBER
 NOT FINAL; ---director tends to extend as delivery director

Explicitly also, a type can be declared NOT FINAL to extend from there onwards using the ALTER TYPE statement.

ALTER TYPE DIRECTOR_OT NOT FINAL;

-------Creating DELIVERY DIRECTOR Object sub-type under DIRECTOR
CREATE OR REPLACE TYPE DEL_DIRECTOR_OT UNDER DIRECTOR_OT

 DelDirFuncArea VARCHAR2(100),
 DelDirClient VARCHAR2(1000),
 DelDirProjNature VARCHAR2(10)
);

-------Creating MANAGER Object sub-type under EMPLOYEE
CREATE OR REPLACE TYPE MANAGER_OT UNDER EMPLOYEE_OT

 ManProj VARCHAR2(1000),
 ManDelv VARCHAR2(100)

 NOT FINAL; ---manager tends to extend as project manager and tech manager

-------Creating PROJECT MANAGER Object sub-type under MANAGER
CREATE OR REPLACE TYPE PROJMAN_OT UNDER MANAGER_OT

 PManTeamSize NUMBER,
 PManAccounts NUMBER);

-------Creating TECHNICAL MANAGER Object sub-type under MANAGER
CREATE OR REPLACE TYPE TECHMAN_OT UNDER MANAGER_OT

 TManTech VARCHAR2(100));

-------Creating DEVELOPER Object sub-type under EMPLOYEE
CREATE OR REPLACE TYPE DEVELOPER_OT UNDER EMPLOYEE_OT

 DevSkill VARCHAR2(100),
 DevExp NUMBER
};

6.4. Object Tables: Database tables can accommodate objects in two ways: either as object tables or relational tables. An object table is created on top of a valid schema object. Every row in the object tables holds the instance of that object, hence known as row object. It can be accessed either as a single object instance table or as a relational table with the dependent object attributes listing as columns of the table. As a single instance table, it can demonstrate many distinct object oriented features.

In a conventional relational heap table, a column of an object type can exist to hold the object’s instances, hence known as a column object.

For the object types created above, I would create a table EMPLOYEES to hold instances of all employees, falling within the organization hierarchy model.

-------Creating object table EMPLOYEES with EMPLOYEE_OT object as row object
CREATE TABLE EMPLOYEES OF EMPLOYEE_OT

(EmpID PRIMARY KEY,
 Check (EmpSalary>0));

The object type upon which the table is created is known as declared type of the table.

Object level scalar attributes can be indexed or constrained in the object tables. All the attributes (inherited and self) which are available until the declared object level node, can be constrained or indexed. Database triggers can be created as usual on the object tables or columns.

The EMPLOYEES table declares a primary key ‘EmpID’ and imposes a check constraint on the attribute ‘EmpSalary’. It must be more than zero in all the instances of the ‘employee’ object and its descendents.

Now, I shall show the creation of instances within the EMPLOYEES table.

The insert statement in the screenshot creates an instance of the EMPLOYEE_OT object type, but it fails.

SQL> INSERT INTO EMPLOYEES VALUES

(EMPLOYEE_OT(100,'Emp 1',TO_DATE('01-JUN-1975'),
 TO_DATE('26-JUL-2001'), 4500, 'IND'));

ERROR at line 2:
ORA-22826: cannot construct an instance of a non instantiable type

The reason is that EMPLOYEE_OT is a NON INSTANTIABLE type, due to which its instances cannot be created in the table.

-------Populating the test data into EMPLOYEES table
BEGIN

-------Creating instance of DIRECTOR
INSERT INTO EMPLOYEES VALUES

SQL> UPDATE  •  Winter 2011 13
(DIRECTOR_OT(100,'Emp 1',TO_DATE('01-JUN-1968'), TO_DATE('26-JUL-2001'), 4500, 'IND', 'BANKING', 10000));

-----Creating instance of DELIVERY DIRECTOR
INSERT INTO EMPLOYEES
VALUES (DEL_DIRECTOR_OT(200,'Emp 2',TO_DATE('11-JAN-1971'), TO_DATE('18-DEC-2010'), 5720, 'AUS', 'BANKING', 10000, 'APAC', 'XYZ Bank', 'Dev'));

-----Creating instance of MANAGER
INSERT INTO EMPLOYEES
VALUES (MANAGER_OT(300,'Emp 3',TO_DATE('13-FEB-1981'), TO_DATE('27-SEP-2003'), 8302, 'NZ', 'ABC Pvt Ltd', 'Banking App'));

-----Creating instance of PROJECT MANAGER
INSERT INTO EMPLOYEES
VALUES (PROJMAN_OT (400,'Emp 4',TO_DATE('21-MAR-1982'), TO_DATE('16-APR-2006'), 9320, 'UK', 'ABC Pvt Ltd', 'App Dev', 25, 10));

-----Creating instance of TECHNICAL MANAGER
INSERT INTO EMPLOYEES
VALUES (TECHMAN_OT (500,'Emp 5',TO_DATE('03-JAN-1999'), TO_DATE('18-DEC-2010'), 3728, 'SL', 'ABC Pvt Ltd', 'App Support', 'Database'));

-----Creating instance of DEVELOPER
INSERT INTO EMPLOYEES
VALUES (DEVELOPER_OT(600,'Emp 6',TO_DATE('17-DEC-1973'), TO_DATE('19-APR-1998'), 4830, 'RUS', 'JAVA', 6));

END;

The above block execution inserts six instances of the declared object’s descendent types into the EMPLOYEES table.

6.5. Object References: Referential relationships can be established between a column and an object type. The reference column serves as a pointer to any instance of the object type, which exists as a row or column object in the database. Their scope can be restricted too, such that the column would point to the object instance created within the specified table only.

For example, the organization keeps track of the salary modifications of its employees, along with their employment details. The employee details could have a referenced EMPLOYEES table for the details. But REF object is a better solution to get rid of the referential network.

The column ‘EmpDetails’ stores a reference pointer, which always points to the instances of EMPLOYEE_OT across the database. The CREATE TABLE statement below creates the table.

-----Create the table with REF column
CREATE TABLE SAL_UPDATE
(EmpID NUMBER,
EmpDetails REF EMPLOYEE_OT,
EmpModSal NUMBER,
EmpModSalDate DATE)

-----Inserting test data from EMPLOYEE’s table
INSERT INTO SAL_UPDATE
SELECT EmpId, REF (T) FROM EMPLOYEES T;

-----Query the table to check the REF Column value

SQL> SELECT EMPID, EMPDETRILS FROM SAL_UPDATE;

EMPID EMPDETRILS
----------
100 0000220208
200 0000220208

6.6. Object Type Metadata: The object type metadata are stored in the data dictionary views [ALL | DBA | USER]_TYPES and [ALL | DBA | USER]_TYPE_ATTRS.

[ALL | DBA | USER]_TYPES – Stores the object type information. I shall briefly describe some of the key columns of the view.

TYPE_OID is the unique 32-bit hex code generated for each object type created in the database.

TYPECODE differentiates a type as an object or collection.

ATTRIBUTES and METHODS are the count of attributes for the type.

FINAL and INSTANTIABLE determine if the type can be further extended or instantiated.

SUPERTYPE_OWNER and SUPERTYPE_NAME are the type’s parent type details. They are NULL for non-hierarchical object types.

TYPEID column uniquely identifies an object type node in a type hierarchy structure. It is NULL for non-hierarchical object types.

[ALL | DBA | USER]_TYPE_ATTRS – Stores the information of net attributes available to an object type. The INHERITED column distinguishes an attribute as inherited or owned. Other than INHERITED, all other columns of the view are attribute general info columns.

SQL> desc dba_types

Name Null? Type
----------------------- ----------  ----------
OWNER VARCHAR2(30)
TYPE_NAME VARCHAR2(30)
TYPE_OID RAW(16)
TYPECODE VARCHAR2(30)
ATTRIBUTES NUMBER
METHODS NUMBER
PREDEFINED VARCHAR2(3)
INCOMPLETE VARCHAR2(3)
FINAL VARCHAR2(3)
INSTANTIABLE VARCHAR2(3)
SUPERTYPE_OWNER VARCHAR2(30)
SUPERTYPE_NAME VARCHAR2(30)
LOCAL_ATTRIBUTES NUMBER
LOCAL_METHODS NUMBER
TYPEID RAW(16)
7. Useful Functions In The Object Relational Model

VALUE, TREAT, REF, DEREF, IF OF, and SYS_TYPEID are a few amongst the many functions which are frequently used with object type instances.

7.1. The VALUE function is used to query object instances in a table. These instances can be of the declared type or even its sub-types.

For example, the SELECT query on the EMPLOYEES table is shown in the below screen dump.

```sql
SQL> SELECT VALUE(T) FROM EMPLOYEES T;
VALUE(T)(EMPID, EMPNAME, EMPBIRTHDATE, EMPJOINDATE, EMPSALARY, EMPLOC)
---------------------------------------------------
DIRECTOR_OT(100, 'Emp 1', '01-JUN-68', '26-JUL-01', 4500, 'IND', 'BANKING', 10000)
DEL_DIRECTOR_OT(200, 'Emp 2', '11-JAN-71', '18-DEC-10', 5720, 'AUS', 'BANKING', 10000, 'APAC', 'XYZ Bank', 'Bev')
MANAGER_OT(300, 'Emp 3', '13-FEB-81', '27-SEP-03', 8302, 'NZ', 'ABC Put Ltd', 'Banking App')
PROJMAN_OT(400, 'Emp 4', '21-MAR-82', '16-APR-06', 9320, 'UK', 'ABC Put Ltd', 'App Devis', 25, 10)
DEVELOPER_OT(600, 'Emp 6', '17-DEC-73', '19-APR-98', 4830, 'RUS', 'JAUA', 'Database')
6 rows selected.
```

7.2. The TREAT function is used to access a sub-type attribute of the object table. Normally, the attributes of the declared type are directly accessible from a SELECT statement, but not the sub-type attributes. It typecasts all the instances as one sub-type and tries to get the value of the requested attribute for all the instances. For indifferent instances, the value appears NULL.

Check the example below.

```sql
SQL> SELECT treat(value(t) as DEVELOPER_OT).DevSkill from employees t;
TREAT(VALUE(T)ASDEVELOPER_OT).DEVSKILL
---------------------------------------------------
JAVA
6 rows selected.
```

7.3. The REF function is used to get the reference pointer value of an object instance. Refer to the example below.

```sql
select ref(t) from employees t;
REF(T)
---------------------------------------
000028020910EE4104966945
00002802098308FA3FBED543
0000280209ED15636142B24F
0000280209FC59EB9982644
0000280209F4159E8B61514
0000280209BD6B730691C
6 rows selected.
```

It is used to populate a REF object column in a table. Similarly, the DEREF function works just at the opposite track. From a REF column value, it can get the actual instance values. Check this DEREF example in the below screenshot.

```sql
SQL> SELECT deref(t.empdetails) FROM SAL_UPDATE T;
DEREF(T.EMPDETAILS)(EMPID, EMPNAME, EMPBIRTHDATE, EMPJOINDATE, EMPSALARY, EMPLOC)
000028020910EE4104966945
00002802098308FA3FBED543
0000280209ED15636142B24F
0000280209FC59EB9982644
0000280209F4159E8B61514
0000280209BD6B730691C
6 rows selected.
```
7.4. The IS OF function is used to filter the instances of a specific branch only. This means that it will get the instances of the specified type and its descendents. Check the example below.

```
SQL> SELECT VALUE(T) FROM EMPLOYEES T WHERE VALUE(T) IS OF (MANAGER_OT);
VALUE(T)(EMPID, EMPNAME, EMPBIRTHDATE, EMPJOINDATE, EMPSALARY, EMPLOC)
-----------------------------------------------
MANAGER_OT(300, 'Emp 3', '13-FEB-81', '27-SEP-03', 8302, 'NZ', 'ABC Pvt Ltd', 'Banking App')
```

In the above screen shot, the SELECT query requests to get the instances of MANAGER. But the query output lists the instances of manager, project manager and technical manager. Another enhancement to the IS OF function is 'IS OF ONLY'. It would perform hard filtering by fetching only the instances of the specified type.

```
---------The query fetches the instance of MANAGER_OT only
SELECT VALUE(T) FROM EMPLOYEES T WHERE VALUE(T) IS OF (ONLY MANAGER_OT);
VALUE(T)(EMPID, EMPNAME, EMPBIRTHDATE, EMPJOINDATE, EMPSALARY, EMPLOC)
------------------------------
MANAGER_OT(300, 'Emp 3', '13-FEB-81', '27-SEP-03', 8302, 'NZ', 'ABC Pvt Ltd', 'Banking App')
```

7.5. The SYS>TypeID function returns the TYPEID value of the object instance. It is useful in identifying the object level node of an instance. Its value is the same as contained in TYPEID column of the USER_TYPES dictionary view.

```
SELECT EmpID, SYS_TYPEID(VALUE(T)) FROM EMPLOYEES T;
```

```
EMPID SYS_TYPEID(VALUE(T))
---------- -------------------
100 02
200 03
300 04
400 05
500 06
600 07
```

For cross verification, we shall query the TYPEID column in the USER_TYPES dictionary view for the above types.

```
SELECT TYPE_NAME, SUPERTYPE_NAME, TYPEID
FROM USER_TYPES
WHERE TYPE_NAME IN ('EMPLOYEE_OT', 'DIRECTOR_OT', 'DEL_DIRECTOR_OT', 'MANAGER_OT', 'PROJMAN_OT', 'TECHMAN_OT', 'DEVELOPER_OT')
ORDER BY TYPEID;
```

```
TYPE_NAME SUPERTYPE_NAME TYPEID
----------------------------- ----------------------
EMPLOYEE_OT 01
DIRECTOR_OT EMPLOYEE_OT 02
DEL_DIRECTOR_OT DIRECTOR_OT 03
MANAGER_OT EMPLOYEE_OT 04
PROJMAN_OT MANAGER_OT 05
TECHMAN_OT MANAGER_OT 06
DEVELOPER_OT EMPLOYEE_OT 07
```

**Conclusion: A Step Ahead**

This paper saw the exploration, observation and implementation of a business problem with the object model. With its unquestionable model, the object relational platform is capturing the latest flavor of database designing. Relational structures have already proven their capabilities since 1979, but over the years, the database technologies have matured enough to induct a non-database paradigm, object oriented, to meet the competent business needs. The upcoming years would be the testing time for the OODBMS.
and ORDBMS. Reliability, robustness and ability to pace with day-to-day problems are the few factors which would be considerable over its commercialization. Additionally, it would be a new learning track of PL/SQL programming for database professionals.

I hope my sincere effort to familiarize you with the ‘world of objects’ would be noted. I shall welcome your comments and feedback at sbh.orcl@gmail.com.

I am Saurabh Gupta, PL/SQL developer for more than four years. I am an Oracle 11g Advanced PL/SQL Professional and specialized on PL/SQL programming, Database modeling and designing, and PL/SQL code optimization. I share my experiences and learning at OTN, through online publications and at my blog sbhoracle.wordpress.com. I have authored more than 60 online publications on Oracle PL/SQL and 11g features at PSOUG (www.psoug.org), DBANOTES (www.dbanotes.com), EXFORSYS (www.exforsys.com), and CLUB-ORACLE (www.club-oracle.com).

Besides digging into Oracle, I love to play snooker but at times, I find drawing out my thoughts on paper catalyzes refreshment. One can reach me @ sbh.orcl@gmail.com for any query or clarification.