5. ORDBMS - Oracle

www.LearnDB.com

Dr. Imed Bouchrika
Dept of Mathematics & Computer Science
University of Souk-Ahras
imed@imed.ws

Object-Relational Database Systems

GOAL:

- Add **object-oriented features** on top of **relational databases**.
  - Complex Data Types
  - Inheritance
  - Encapsulation using Methods.

Object Types

- Each object has the following:
  - ***Name*** - uniquely identifies it within a schema.
  - ***Attributes*** –
    - primitive data types
    - complex objects.
  - ***Methods*** – written in PL/SQL

Declaring a type Or Object Or Class

```
CREATE TYPE Point AS OBJECT (
   x NUMBER,
   y NUMBER
);

CREATE TYPE side AS OBJECT (
    a Point,
    b Point
);
```

Object Table

- Creates a table of objects

```
CREATE TABLE mypoints OF Point;
```

- Inserting data: which one ?

```
INSERT INTO mypoints
    VALUES  ('1', '2');

INSERT INTO mypoints
    VALUES  (Point('1', '2'));

INSERT INTO mypoints
    VALUES  (new Point('1', '2'));
```

Object Table

- Creates a table of objects

```
CREATE TABLE mysides of side;
```

- Inserting data

```
INSERT INTO mysides VALUES
        ('1', '2', '1', '2');
```

will it work ?

```
INSERT INTO mysides VALUES
        ( ('1', '2') , ('1', '2'));
```

will it work ?

```
INSERT INTO mysides VALUES
     (new Point('1', '2'), new Point('1', '2'));
```

Select/Querying the data

- You can view the table as :

```
SELECT * from mypoints;

SELECT p.x, p.y from mypoints p;

SELECT * from mysides;     Will it work ?

SELECT s.a.x, s.a.y from mysides s;
```

Note the use of **AS**

Dot notation

- You can view the table as :

```
CREATE TYPE Point AS OBJECT (
  x NUMBER,
  y NUMBER
);
CREATE TYPE side AS OBJECT (
   a Point,
   b Point
);

SELECT p.a.x , p.a.y FROM tablename p;
```

Reference Pointers

- Example :

```
CREATE TYPE Point AS OBJECT (
   x NUMBER,
   y NUMBER
);

CREATE TYPE side2 AS OBJECT (
    a REF Point,
    b REF Point
);

CREATE TABLE mysides2 OF side2;
```

Reference Pointers

- To insert data : which one works ?

→
```
insert into mysides2 values
            (new Point(1,1),new Point(1,2))
```

→
```
insert into mysides2 values (
  (select ref(p) from mypoints p
        where p.x=1 and p.y=2 and ROWNUM=1),

  (select ref(p) from mypoints p
        where p.x=1 and p.y=2 and ROWNUM=2)
)
```

- ROWNUM is a magic keyword in Oracle works as Limit in MySQL

Reference Pointers

- Further Example :

```
create type lecturer as object (
    name VARCHAR2(100),
    researchArea VARCHAR2(100)
);

create table lecturers of lecturer;

create type department as object (
     name VARCHAR2(100),
     director REF lecturer
)
create table departments of department;
```

Reference Pointers

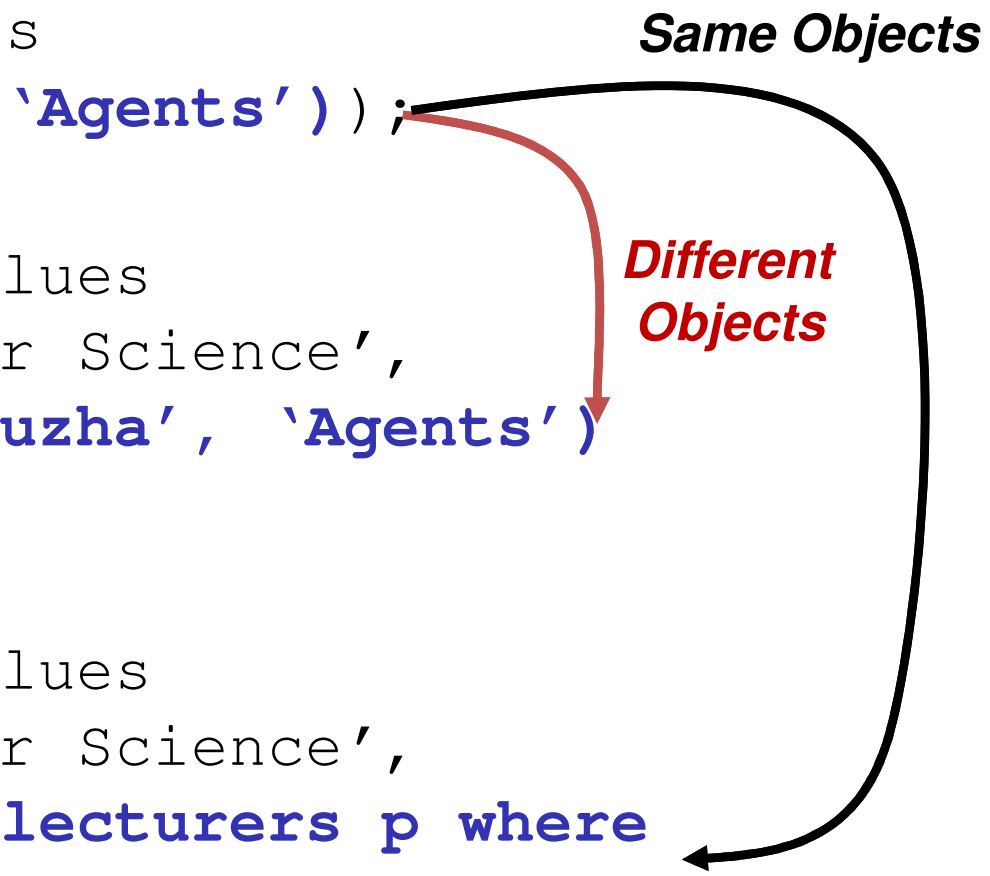
- To Insert :

```
insert into lecturers values
    (new lecturer('Nouzha', 'Agents'));

insert into deptartments values
    (new department('Computer Science',
            new lecturer('Nouzha', 'Agents')
        )
```

vs

```
insert into deptartments values
    (new department('Computer Science',
        ( select ref(p) from lecturers p where
            p.name='Nouzha' and ROWNUM=1)
        )
    )
```



Reference Pointers

- REFs are used to point from one object to another .
- REFS must be valid when they are stored
- Non-scoped

```
CREATE TABLE sides2 (a REF Point,b Point);
```

- Scoped – constrained to a table

```
CREATE TABLE sides2 (a REF Point,b Point
SCOPE FOR (a) IS points_table1);
```

Reference Pointers

- How about searching using SELECT?

```
CREATE TYPE side2 AS OBJECT (
    a REF Point,
    b REF Point
);
CREATE TABLE mysides2 OF side2;
```

```
Select * from mysides2  ?
```

```
select deref(c.a.x) from mysides2 c; ?
```

```
select deref(c.a).x from mysides2 c; ?
```

Methods

- Functions or procedures written in:
  - PL/SQL
  - **Java**

- There are different types of Methods
  - Constructor
  - Comparison
  - Instance or Members
  - Static

Constructor

- Constructors are automatically created by the system.

- For Example :

```
point('1','2');
```

- NULL values
  - INSERT INTO mysides VALUES (point(NULL,NULL))
  - INSERT INTO mysides VALUES (NULL);

Member Method

```
create or replace type side as object(
    a point,
    b point,
    member function get_length return integer
    );
/
create or replace type body side as
    member function get_length return integer is
    begin
      return sqrt( ((self.a.x-self.b.x)*(self.a.x-
      self.b.x)) + ((self.a.y-self.b.y)*(self.a.y-
      self.b.y)) );
    end;
 end;
```

Invoking the method

```
CREATE TYPE side AS OBJECT (
   a Point,
   b Point,
   member function get_length return integer
);
CREATE TABLE mysides OF side;
```

```
SELECT p.get_length()
FROM   mysides p
WHERE  p.get_length()  > 10 ;
```

Static Methods

- Static Procedures don't have the SELF passed to them as they belong to the class not the instance.

```
CREATE TYPE point AS OBJECT(
x NUMBER,
y NUMBER,
STATIC PROCEDURE info (a NUMBER,B NUMBER) RETURN STRING );
/
create or replace type body point as
     static procedure inf(a number, b number) return string is
     begin
       return 'point info (' || a || ' , ' || b || ')'
     end;
  end;
```

Comparison Method

- Methods used to do compare object instances.

- **ORDER** – takes another instance and compares it to the current instance (Don't be confused !)

negative	this < arg
zero	this = arg
positive	this > arg

- **MAP** – Returns a number that is used to rank (**order)** instances of object types

MAP and ORDER

- An object can have **a single** MAP method **or** a single ORDER method.

- Which one should you use and why?

Order Method

```
CREATE TYPE side AS OBJECT (
   a Point,
   ORDER MEMBER FUNCTION ord (hello side) RETURN NUMBER);
/
CREATE OR REPLACE TYPE BODY side AS
  ORDER MEMBER FUNCTION ord(hello side)
    RETURN NUMBER IS
   BEGIN
     IF (value.x < hello.a.x) THEN RETURN -1;
     ELSIF (value.x > hello.a.x) THEN RETURN +1;
     ELSE RETURN 0;
     END IF;
   END;
END;
/
```

Order Method

```
CREATE TYPE circle AS OBJECT (
    radius NUMBER,
    x NUMBER,
    y NUMBER,

ORDER MEMBER FUNCTION match RETURN NUMBER);
/

CREATE OR REPLACE TYPE BODY cirle AS
  MAP MEMBER FUNCTION match(c cirlce) RETURN NUMBER IS
  BEGIN
       IF radius < c.radius THEN
              RETURN -1;
       ELSIF radius > c.radius THEN
              RETURN 1;
       ELSE
              RETURN 0;
       END IF;
  END;
END;

create table circles of circle ;
```

Order Method

→insert into circles values
(new circle (4,10,4));

→ insert into circles values
(new circle (6,10,4));

→ select * from circles c
where **value(c)** > circle(4,14,1)

Map Method

- **MAP** – Returns a number that is used to rank (**order)** instances of object types

```
CREATE TYPE side AS OBJECT (
     a Point,
   MAP MEMBER FUNCTION mp RETURN NUMBER);
/
CREATE OR REPLACE TYPE BODY side AS
   MAP MEMBER FUNCTION mp RETURN NUMBER IS
   BEGIN
      RETURN value;
   END;
END;
```

Map Method

```
CREATE TYPE circle AS OBJECT (
    radius NUMBER,
    x NUMBER,
    y NUMBER,
MAP MEMBER FUNCTION get_area RETURN NUMBER);
/
CREATE OR REPLACE TYPE BODY cirle AS
  MAP MEMBER FUNCTION get_area RETURN NUMBER IS
  BEGIN
    RETURN self.radius*self.radius*3.14;
  END;
END;

create table circles of circle ;
```

Map Method

→insert into circles values
(new circle (4,10,4));

→ insert into circles values
(new circle (6,10,4));

→ select * from circles c
order by **value(c)**

Inheritance

- With Inheritance, the **subclass** inherits all properties (attributes and methods) of the parent class.

- In Oracle, we use **UNDER**

```
CREATE TYPE person AS OBJECT(
    name VARCHAR2(20)
    age NUMBER) NOT FINAL;

CREATE TYPE student UNDER person(
    school VARCHAR2(100)
);
```

- By default, all new objects are declared **FINAL**

Inheritance

- All objects of student, are by name object of person.

```
CREATE TABLE persons AS person;

INSERT INTO persons ( Person('Imed',67) );

INSERT INTO persons ( Student('Asma',7) );
```

Inheritance / Overriding Methods

- Sub-types can override the inherited methods to provide a different implementation:

```
CREATE TYPE person AS OBJECT(
  name VARCHAR2(20)
  age NUMBER,
  member function getinfo return VARCHAR2(100),
) NOT FINAL;
/
 create or replace type body person as
 member function getinfo return varchar2(100) is
   begin
     return 'Name '||self.name;
   end;
 end;
```

Inheritance / Overriding Methods

- To override a method use the keyword :

```
CREATE TYPE student under person(
 school VARCHAR2(100),
 OVERRIDING member function getinfo return VARCHAR2(100),
);
/
create or replace type body student as
OVERRIDING member function getinfo return varchar2(100)is
   begin
        return 'Student Name '||self.name;
   end;
end;
```

Collection Types

- Oracle provides two techniques for modeling one-to-many relationships.
  - VARRAY – stores a **fixed** number of repeating attributes.
  - Nested Tables – table within a table.
- Collections can be columns in tables or attributes of object types.

VARRAY

- Just like a C array :
  - It has a fixed size
  - It contains objects of the same datatype.
  - Each element has an index
- VARRAYs can be used as columns in tables or as attributes in objects.

Declaring a VARRAY

- Syntax:

```
CREATE TYPE type_name AS VARRAY (limit) OF data_type;
```

- Example

```
 CREATE TYPE Point AS OBJECT(
   x NUMBER,
   y NUMBER);
/
 CREATE TYPE side AS OBJECT(
   a Point,
   b Point);
/
 CREATE TYPE fourSides AS VARRAY(4) OF side;
/
 CREATE TABLE Square (
    name      VARCHAR2(20),
    sd     fourSides);
```

INSERT with a VARRAY

- To Insert using VARRAY:

```
insert into Square values
        (
          'Square One',
           fourSides(
                 side(Point(1,2), Point(2,3)),
                 side(Point(3,2), Point(4,3)),
                 NULL
           )
        )
```

- You don't have to set all the values.

- To retrieve data using **SELECT,** you are advised to use **PL/SQL**

```
select c.name, b.a.x from square c, table(c.sd) b
```

VARRAY – PL/SQL

- PL/SQL Program for accessing VARRAY:

```
declare
begin
  for c1 in (select * from square) loop
      dbms_output.put_line('Row fetched...');
      FOR i IN c1.sd.FIRST..c1.sd.LAST LOOP
          dbms_output.put_line(c1.name ||' a:x='|| c1.sd(i).a.x);
      END LOOP;
  end loop;
end;
/
```

Nested Table

- Table within another table.
- The tables **don’t have a fixed** maximum size,
- The nesting has a depth of one.
- The tables are unordered.
- The tables can have triggers and indexes.
- The nested table can’t be directly queried.

Creating a simple nested table

- Create the nested table datatype

```
CREATE TYPE sides AS TABLE OF side
```

- Nested table as a column

```
CREATE TABLE polygon(
    name varchar2(100),
    sd sides
)nested table sd store as sd;
```

INSERT with Nested Tables

- To Insert for Nested Table VARRAY:

```
insert into Polygon values
        (
          'Square One',
           sides(
                  side(Point(1,2), Point(2,3)),
                  side(Point(3,2), Point(4,3))
           )
        )
```

- To retrieve data using **SELECT,** you are advised to use **PL/SQL**

```
select c.name, b.a.x from square c, table(c.sd) b
```

Nested Tables

- PL/SQL Program for accessing Nested Tables:

```
declare
begin
  for c1 in (select * from polygon) loop
      dbms_output.put_line('Row fetched...');
       FOR i IN 1..c1.sd.count LOOP
        dbms_output.put_line(c1.name ||' a:x='|| c1.sd(i).a.x);
      END LOOP;
  end loop;
end;
/
```

Collection Functions

THE	Flattens the nested table.
CAST	Maps a collection of one type to another VARRAY <–> Nested
MULTISET	Maps a database to a collection
TABLE	Maps a collection to a table

PL/SQL

- The PL/SQL programming language was developed by Oracle as Procedural extension Language for SQL

- Hello World Example:

```
DECLARE
     message  varchar2(20):= 'Hello, World!';
BEGIN
   dbms_output.put_line(message);
END;
/
```

- Data Types:
  - NUMBER, INTEGER,FLOAT, REAL, INT, CHAR, VARCHAR2, BOOLEAN, BLOB

PL/SQL

- User Defined Subtypes:

```
SUBTYPE message IS varchar2(100);
greetings message:='Hello, world!';
```

- IF-ELSE Syntax:

```
IF(1==2)   THEN
        .....
ELSIF( 2==3) THEN
        .....
ELSE
        .....
END IF;
```

PL/SQL

- While Loop :

```
WHILE condition LOOP
    ....
END LOOP;
```

- FOR Loop :

```
FOR counter IN initial_value .. final_value LOOP
    .....
END LOOP;
```

- You can use the keyword : **CONTINUE;** to skip to the next iteration.

PL/SQL

- Cursor Syntax :

```
CURSOR cursor_name IS select_statement;
```

- Example

```
declare
cursor r is (select * from polygon);
begin
   for c1 in r loop
       dbms_output.put_line('Row fetched...');
       FOR i IN 1..c1.sd.count LOOP
            dbms_output.put_line(c1.name ||' a:x='||
                 c1.sd(i).a.x);
       END LOOP;
   end loop;
end;
```

For you to search !

- **Overriding vs. Overloading of methods.**
- **Auto_Increment in Oracle ?**
- **Creating View in Oracle.**