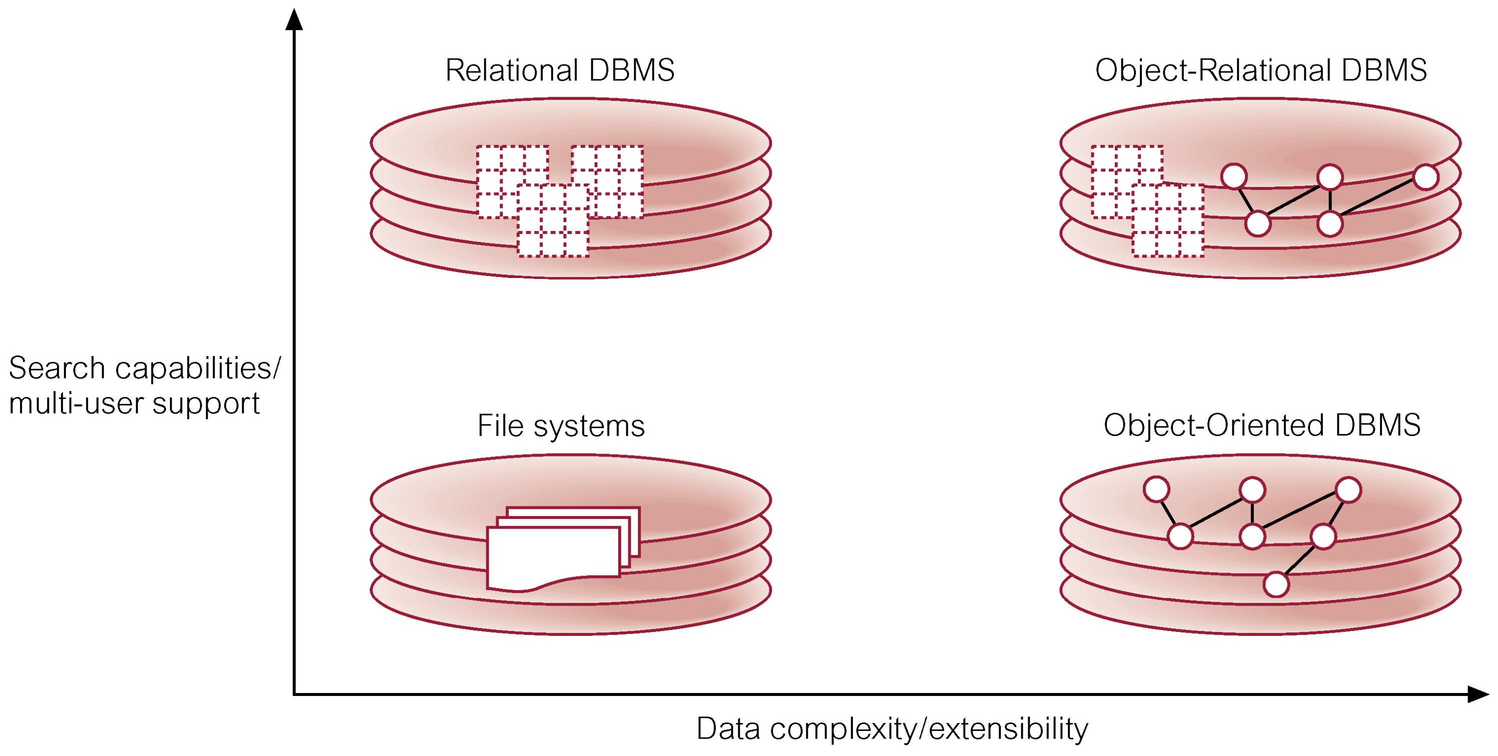
Object-Relational Features in Oracle Database

Cyrus Shahabi
Computer Science Department
University of Southern California
shahabi@usc.edu

View of the Database World



Oracle Database

- History of Oracle database
  - In 1979, Oracle Version 2 introduced
    - An early commercial relational database system.

  …

  - In 1997, Oracle version 8 released
    - Support for object-oriented development and multimedia applications.
    - **Object-Relational DBMS**
  - In 1999, Oracle 8*i* released
    - Tuned with the needs of the Internet/Web
  - In 2001, Oracle 9*i* released
    - Query-intensive data warehouses, and demanding Internet applications (XML, Text)
  - In 2003, Oracle 10g released
    - Support for Grid Computing
  - In 2007, Oracle 11g released
    - Automatic memory disk and memory management
    - Extended features (e.g., 3D capabilities)

Object-Relational Elements in Oracle 10*g*

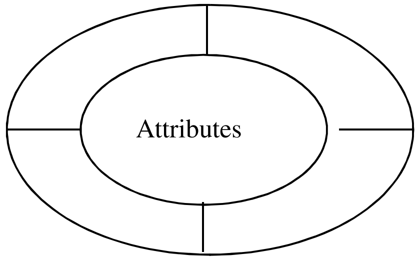
- Object-Oriented Concepts
- Objects
- Methods
- Object Tables
- Type Inheritance
- Collections
- Object Types and References

Object-Oriented Concepts

- **Abstraction and Encapsulation** (Provided by Abstract Data Types (ADT))
  - ***Abstraction*** is the process of identifying the essential aspects of an entity and ignoring the unimportant properties. Focus on what an object is and what it does, rather than how it should be implemented.
  - ***Encapsulation*** (or information hiding) provides data independence by separating the external aspects of an object from its internal details, which is hidden from the outside world.
- **Classes**
  - ***Classes:*** A class is a blueprint or prototype from which objects are created. A group of objects with the same attributes and methods. Hence, the attributes and the associated methods are defined once for the class rather than separately for each object.
  - ***Attributes*** (or instance variables) describe the current state of an object (the notation for attribute: object-name.attribute-name).

Object-Oriented Concepts

- ***Methods:*** define the behavior of the object. They can be used to change the object's state by modifying its attribute values, or to query the value of the selected attributes. A method consists of a name and a body that performs the behavior associated with the method name (notation: object-name.method-name).



- The ***instances*** of a class are those objects belonging to a class.

Advantages of ORDBMS

- Enables *reuse* and *sharing*.
  - Ability to extend the DBMS server to perform standard functionality centrally, rather than have it coded in each application. Example: Embedded Functions, it saves having to define it in each application that needs it.
- Ability and support for complex objects and rich data types, termed *abstract data types* (ADTs)
  - Complex applications such as Oracle Spatial
- Support for Inheritance
  - Inherent attributes and behavior of the pre-existing classes, hence ease of definition and programming

Oracle Object Types

- User-Defined data types (classes)
- Consist of 2 parts: attributes + methods

```
CREATE TYPE person_type AS OBJECT (
  name               VARCHAR2(30),
  phone              VARCHAR2(20),  -- attributes declared.

  MEMBER FUNCTION get_areacode RETURN VARCHAR2 ); -- method
/ -- This slash needed to get Oracle process this statement.

--Defining an object type does not allocate any storage.

--The body of method is defined in a separate CREATE
--TYPE BODY statement, written in PL/SQL or any other languages.

DROP TYPE person_type;
--First drop all tables and other types using person_type.
```

Oracle Objects

- Definition
  - Actual instance of the defined object type,
  - Storages are allocated to an object and values are assigned to the attributes of an object

```
CREATE TABLE contacts (
     contact          person_type,
     c_date           DATE );
-- object type can be used like any other built-in data types.

INSERT INTO contacts VALUES (
     person_type('Tommy Trojan', ''213-740-1114'), -- instance
     '24 Jan 2004' );
-- person_type is instantiated and values are assigned to
-- the attributes of the object instance.
```

Oracle Methods

- Definition
  - Functions/procedures declared in the object type definition to implement behavior of the object of that type.
  - Written in PL/SQL or virtually any other languages (Java, C…)

- Method types
  - Member method
    - Defined on object instance's data.
  - Static method
    - Invoked on the object type, not its instances.
    - Can be used to the operations that are global to the type (e.g. initialization)
  - Constructor method
    - Built-in constructor function, like in C++.

Member Method

- Member methods are used to access an object instance's values.

```
CREATE OR REPLACE TYPE BODY person_type AS
     MEMBER FUNCTION get_areacode RETURN VARCHAR2 IS
     BEGIN
       RETURN SUBSTR(phone, 1, 3);
     END get_areacode;
END;
/
-- Define the body of the method using CREATE OR REPLACE TYPE BODY.

SELECT c.contact.get_areacode()
FROM contacts c;
-- Invoke a member method


C.CONTACT.GET_AREACODE()
-------------------------------------------------------------
213
```

Constructor Method

- Every object type has a constructor method implicitly defined by system.
- Returns a new instance of the user-defined object type and sets up the values of its attributes.
- The name of constructor method is the same as the name of the object type.

```
p = person_type('Scott Tiger', '321-123-1234');
--Built-in constructor method, person_type(att1, att2) is invoked
--to create a new object instance of person_type, specify values
--for its attributes(name, phone), and set the object into a
--variable p.

INSERT INTO contacts
VALUES (person_type('Scott Tiger', '321-123-1234'),
       '10 Feb 2004'));
--Same thing occurs here.
```

Oracle Object Tables

- Object Table: special type of table, each row represents an object

```
    CREATE TYPE person_type AS OBJECT (
     name    VARCHAR2(30),
     phone   VARCHAR2(20) );
 /
    CREATE TABLE person_table OF person_type;
    INSERT INTO person_table
     VALUES (person_type ('Scott Tiger', '321-123-1234'));
    SELECT VALUE(p) FROM person_table p WHERE p.name = 'Scott Tiger';
    -- Single-column table: each row is a person_type object
    -- Perform object-oriented operations
```

- Comparing to a relational table

```
    CREATE TABLE person_table (
     name    VARCHAR2(30),
     phone   VARCHAR2(20) );
    INSERT INTO person_table
     VALUES ('Tommy Trojan', '213-740-1212');
    SELECT name, phone FROM person_table;
    -- Multi-column table: treat person_table as a relational table
```

Methods to Compare Objects (1)

- Define a special kind of member methods to compare objects.
- Define either a *map method* or an *order method* in an object type.
  - Map Method
    - Map object instances into one of the scalar types DATE, CHAR, NUMBER,…

```
CREATE TYPE circle_type AS OBJECT (
   x                  NUMBER,
   y                  NUMBER,
   r                  NUMBER,
   MAP MEMBER FUNCTION get_area RETURN NUMBER ); /

CREATE TYPE BODY circle_type AS
  MAP MEMBER FUNCTION get_area RETURN NUMBER IS
  BEGIN
     RETURN 3.14 * r * r;
  END get_area;
END; /

SELECT * FROM circles c
ORDER BY VALUE(c);    --Result should be ordered by circles' area
```

Methods to Compare Objects (2)

- Order Method
  - Provides direct object-to-object comparison, telling that the current object is less than, equal to, or greater than the other object.

```
CREATE or REPLACE TYPE circle_type AS OBJECT (
   x                     NUMBER,
   y                     NUMBER,
   r                     NUMBER,
   ORDER MEMBER FUNCTION match(c circle_type) RETURN INTEGER );  /

CREATE OR REPLACE TYPE BODY circle_type AS
  ORDER MEMBER FUNCTION match (c circle_type) RETURN INTEGER IS
  BEGIN
    IF r < c.r THEN  -- 3.14*r² < 3.14*c.r²
      RETURN -1;     -- any negative number
    ELSIF r > c.r THEN
      RETURN 1;      -- any positive number
    ELSE
      RETURN 0;
    END IF;
  END;
END; -- returns only one integer value among positive, 0, and negative.
```

Methods to Compare Objects (3)

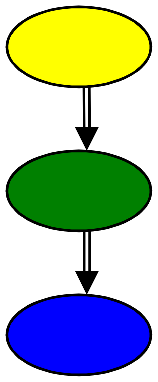
```
CREATE TABLE circles OF circle_type;

INSERT INTO circles VALUES (circle_type(10, 10, 3));

INSERT INTO circles VALUES (circle_type(40, 20, 8));

INSERT INTO circles VALUES (circle_type(10, 50, 4));

SELECT c.x, c.y
FROM circles c
WHERE VALUE(c) < (circle_type(40, 25, 5)) ;

CIRCLES.X                     CIRCLES.Y
-------------                 ----------------
10                            10
10                            50
```

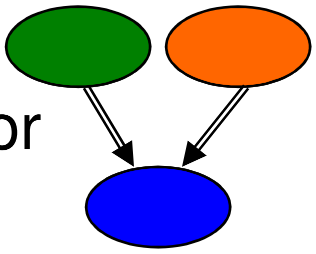
OO Concepts - Inheritance

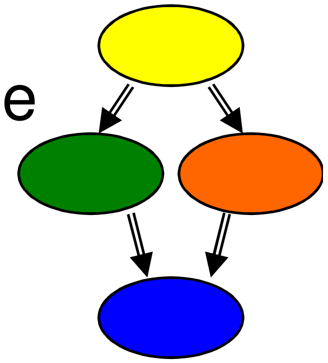
- ***Subclasses:*** A class of objects that is defined as a special case of a more general class (the process of forming subclasses is called *specialization*).
- ***Superclass:*** A class of objects that is defined as a general case of a number of special classes (the process of forming a superclass is called *generalization*). All instances of a subclass are also instances of its superclass.
- ***Inheritance:*** By default, a subclass inherits all the properties of its superclass (or it can redefine some (or all) of the inherited methods). Additionally, it may define its own unique properties.

OO Concepts - Inheritance

- ***Single inheritance:*** When a subclass inherits from no more than one superclass (note: forming *class hierarchies* is permissible here).
- ***Multiple inheritance:*** When a subclass inherits from more than one superclass (note: a mechanism is required to resolve conflicts when the Superclasses have the same attributes and/or methods). Due to its complexity, not all OO languages and database systems support this concept.
- ***Repeated inheritance:*** A special case of multiple inheritance where the multiple Superclasses inherit from a common superclass (note: must ensure that subclasses do not inherit properties multiple times).





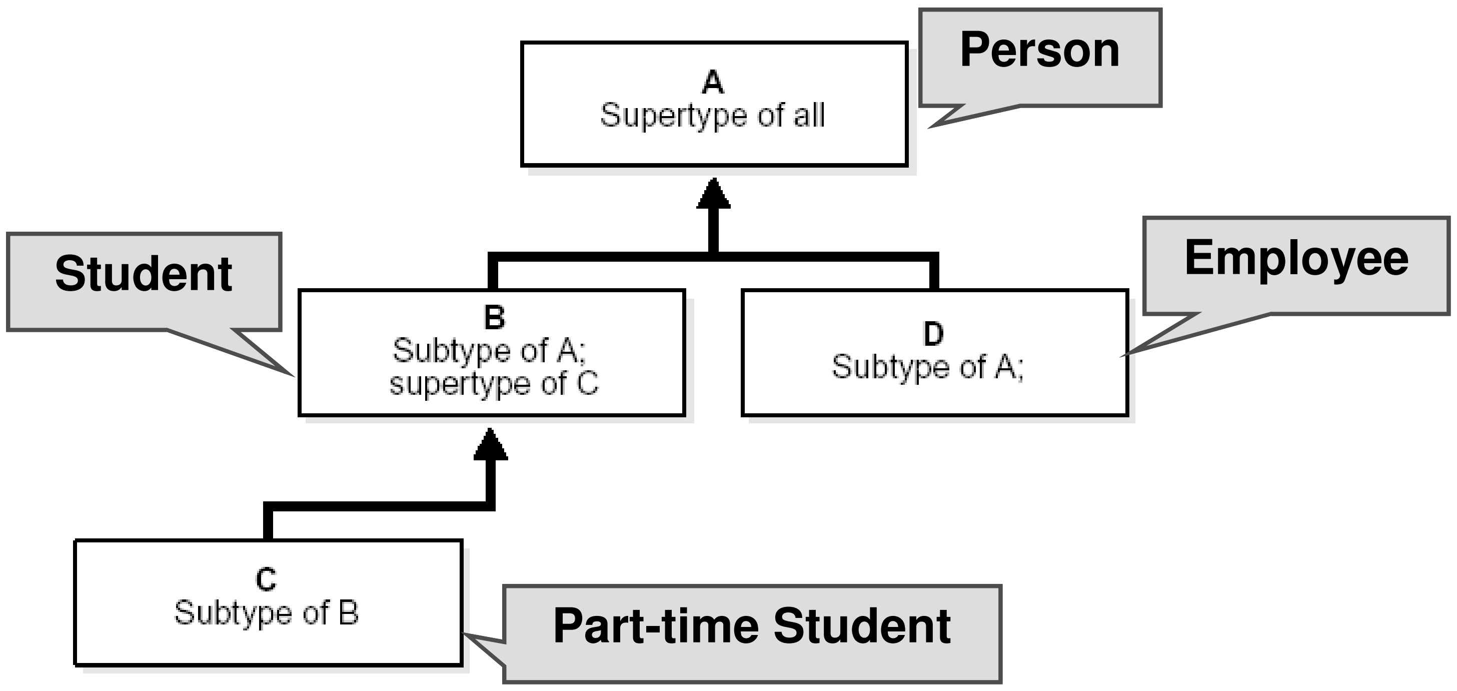


OO Concepts - Inheritance

- ***Overriding:*** To redefine an inherited property by defining the same property differently at the subclass level.
- ***Overloading:*** A general case of overriding where the same method name is reused within a class definition (overriding) or across class definitions. Hence, a single message can perform different functions depending on which object receiving it and, if appropriate what parameters are passed to the method (e.g., *print* method for different objects).
- ***Polymorphism:*** “Having many forms” in Greek, is a general case of overloading.
  - *Inclusion polymorphism:* Same as overriding.
  - *Operation (or ad hoc) polymorphism:* Same as overloading.
  - *Parametric polymorphism (or Genericity):* It uses types as parameters in generic type (or class) definition.

Oracle Type Inheritance (1)

- Supertype/Subtype
  - Subtype is derived from a parent object type, Supertype.
    - Subtype inherits all attributes and methods from its supertype.
- Example



Oracle Type Inheritance (2)

```
CREATE OR REPLACE TYPE person_type AS OBJECT (
   ssn          NUMBER,
   name         VARCHAR2(30),
   address      VARCHAR2(20)) NOT FINAL; /

--To permit subtype, object type should be defined as NOT FINAL.
--By default, an object type is FINAL

CREATE TYPE student_type UNDER person_type (
   deptid       NUMBER,
   major        VARCHAR2(30)) NOT FINAL; /

CREATE TYPE employee_type UNDER person_type (
   empid        NUMBER,
   mgr          VARCHAR2(30)); /

CREATE TYPE part_time_student_type UNDER student_type (
   numhours     NUMBER ); /
```

Oracle Type Inheritance (2.5!)

```
CREATE TABLE persons OF person_type;

INSERT INTO persons VALUES (student_type(123, 'Tommy',
   'PHE-306', 1, 'cs'));
INSERT INTO persons VALUES (employee_type(789,'Trojan',
   'PHE-314', 888, 'Cyrus'));
INSERT INTO persons VALUES (456, ‘Mike‘,'PHE-314’);

SELECT * FROM persons;
SELECT VALUE(p) FROM persons p WHERE VALUE(p) IS OF
   (employee_type);
SELECT TREAT(VALUE(p) AS student_type).major FROM
  persons p WHERE VALUE(p) IS OF (student_type);
```

Oracle Type Inheritance (3)

- Overloading/Overriding methods

```
CREATE TYPE Shape_typ AS OBJECT (...,
  MEMBER PROCEDURE Enlarge(x NUMBER),
  ...) NOT FINAL; /
CREATE TYPE Circle_typ UNDER Shape_typ (...,
  MEMBER PROCEDURE Enlarge(x CHAR(1))); /
--Define the inherited method Enlarge() to deal with different types of
--input parameters.

CREATE TYPE Shape_typ AS OBJECT (...,
  MEMBER PROCEDURE Area(),
  FINAL MEMBER FUNCTION id(x NUMBER)...
) NOT FINAL; /
CREATE TYPE Circle_typ UNDER Shape_typ (...,
  OVERRIDING MEMBER PROCEDURE Area(),
  ...); /
--Redefine an inherited method Area() to make it do something different
--in the subtype.
```

Oracle Collections

- Set of data elements
  - VArray - ordered set of data elements.

```
CREATE TYPE phones AS VARRAY(3) of VARCHAR2(20); /
--Each element has an index, corresponding to its position in
--the array
```

  - Nested Table - unordered set of data elements

```
CREATE TYPE people_type AS TABLE OF person_type; /
--Declare the table type used for a nested table.

CREATE TABLE contacts (
   contact                  people_type,
   c_date          DATE )
  NESTED TABLE contact STORE AS people_table;
--Declare a nested table
```

Inserting/Querying Collections

```
INSERT INTO contacts
VALUES (people_type(person_type('Tommy Trojan', '213-740-1234'),
                    person_type('Scott Tiger', '321-123-1234')),
        '12 Feb 2004');

SELECT * FROM contacts;
CONTACT(NAME, PHONE)                                                C_DATE
-------------------------------------------------------------------------------
PEOPLE_TYPE(PERSON_TYPE('Tommy Trojan', '213-740-1234'), PERSON_TYPE('Scott
   Tiger', '321-123-1234'))                                         12-FEB-04


SELECT p.phone, c.c_date FROM contacts c, TABLE(c.contact) p;
PHONE                C_DATE
-------------------- ---------
213-740-1234         12-FEB-04
321-123-1234         12-FEB-04

SELECT p.phone FROM TABLE(SELECT c.contact FROM contacts c) p;
-- result(?)
```

Oracle Object Types and References

- REF Datatype
  - REF is a logical "pointer" to a row object.
    - For an object type *t*, REF *t* is the reference to the values of type *t*.

```
CREATE TABLE contacts (
     contact          REF person_type,
     c_date           DATE );
--contact attribute is a reference to the values of person_type.

CREATE TABLE person_table OF person_type;
--create object table consisting of person_type objects

INSERT INTO person_table VALUES (‘Tommy Trojan’, ‘213-740-1212’);

INSERT INTO contacts
  SELECT REF(p), ’12 Jan 2004’
  FROM person_table p
  where p.name like ‘%Tommy%’;
```

Oracle Object Types and Reference

- Querying to a REF

```
SELECT *
FROM contacts;

 CONTACT                                         C_DATE
 ------------------------------------------------------------------------
 0000220208D28EEDE1C5736BD7E034080020B68B64      12-JAN-04

SELECT c.contact.name, c.c_date
FROM contacts c;
--using dot notation to follow the reference.

 CONTACT.NAME         C_DATE
 -------------------- ---------
 Tommy Trojan         12-JAN-04
```

References

- For more information,
  - Online Oracle 10*g* Documentations

    http://www.oracle.com/technology/documentation/database10g.html

  - A.R. 4: Application Developer's Guide – Object-Relational Features