

Object-Relational Databases

User-Defined Types

Object ID's

Nested Tables

Merging Relational and Object Models

- ◆ Object-oriented models support interesting data types --- not just flat files.
 - ▶ Maps, multimedia, etc.
- ◆ The relational model supports very-high-level queries.
- ◆ Object-relational databases are an attempt to get the best of both.

Evolution of DBMS's

- ◆ Object-oriented DBMS's failed because they did not offer the efficiencies of well-entrenched relational DBMS's.
- ◆ Object-relational extensions to relational DBMS's capture much of the advantages of OO, yet retain the relation as the fundamental abstraction.

SQL-99 and Oracle Features

- ◆ SQL-99 includes many of the object-relational features to be described.
- ◆ However, being so new, different DBMS's use different approaches.
 - ▶ We'll sometimes use features and syntax from Oracle.

User Defined Types

- ◆ A *user-defined type*, or UDT, is essentially a class definition, with a structure and methods.
- ◆ Two uses:
 1. As a *rowtype*, that is, the type of a relation.
 2. As the type of an attribute of a relation.

UDT Definition

```
CREATE TYPE <typename> AS (  
    <list of elements, as in CREATE TABLE>  
);
```

◆ Oracle syntax:

1. Add "OBJECT" as in CREATE ... AS OBJECT.
2. Follow with / to have the type stored.

Example: UDT Definition

```
CREATE TYPE BarType AS (  
    name    CHAR(20),  
    addr    CHAR(20)  
);
```

```
CREATE TYPE BeerType AS (  
    name    CHAR(20),  
    manf    CHAR(20)  
);
```

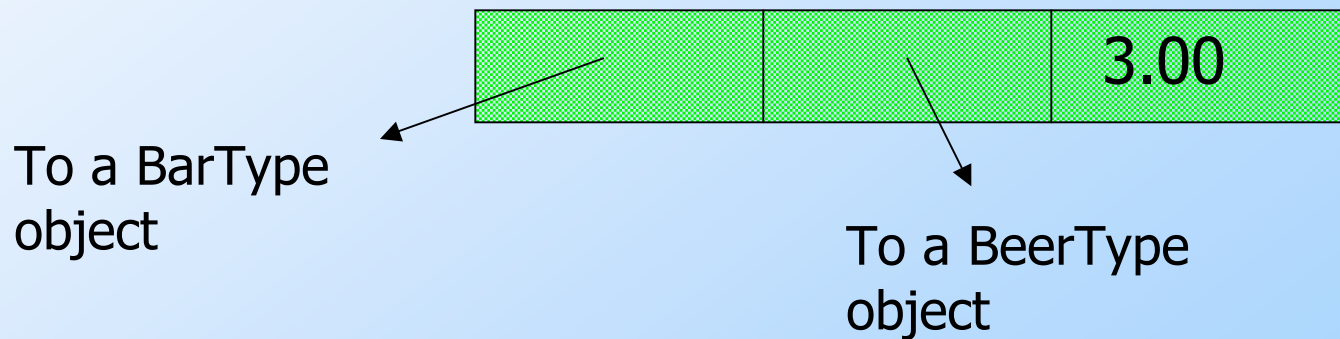
References

- ◆ If T is a type, then $\text{REF } T$ is the type of a reference to T , that is, a pointer to an object of type T .
- ◆ Often called an “object ID” in OO systems.
- ◆ Unlike object ID’s, a REF is visible, although it is usually gibberish.

Example: REF

```
CREATE TYPE MenuType AS (  
  bar      REF BarType,  
  beer     REF BeerType,  
  price    FLOAT  
);
```

◆ MenuType objects look like:



UDT's as Rowtypes

- ◆ A table may be defined to have a schema that is a rowtype, rather than by listing its elements.
- ◆ Syntax:

```
CREATE TABLE <table name> OF  
  <type name>;
```

Example: Creating a Relation

```
CREATE TABLE Bars OF BarType;
```

```
CREATE TABLE Beers OF BeerType;
```

```
CREATE TABLE Sells OF MenuType;
```

Values of Relations with a Rowtype

- ◆ Technically, a relation like Bars, declared to have a rowtype BarType, is not a set of pairs --- it is a unary relation, whose tuples are objects with two components: name and addr.
- ◆ Each UDT has a *type constructor* of the same name that wraps objects of that type.

Example: Type Constructor

- ◆ The query

```
SELECT * FROM Bars;
```

- ◆ Produces “tuples” such as:

```
BarType('Joe"s Bar', 'Maple St.')
```

Accessing Values From a Rowtype

- ◆ In Oracle, the dot works as expected.
 - ▶ But it is a good idea, in Oracle, to use an alias for every relation, when O-R features are used.

- ◆ Example:

```
SELECT bb.name, bb.addr  
FROM Bars bb;
```

Accessing Values: SQL-99 Approach

- ◆ In SQL-99, each attribute of a UDT has *generator* (get the value) and *mutator* (change the value) methods of the same name as the attribute.
 - ▶ The generator for *A* takes no argument, as *A()*.
 - ▶ The mutator for *A* takes a new value as argument, as *A(v)*.

Example: SQL-99 Value Access

- ◆ The same query in SQL-99 is

```
SELECT bb.name(), bb.addr()  
FROM Bars bb;
```

Inserting Rowtype Values

- ◆ In Oracle, we can use a standard INSERT statement, remembering that a relation with a rowtype is really unary and needs that type constructor.

- ◆ Example:

```
INSERT INTO Bars VALUES(  
    BarType('Joe''s Bar', 'Maple St.')  
);
```

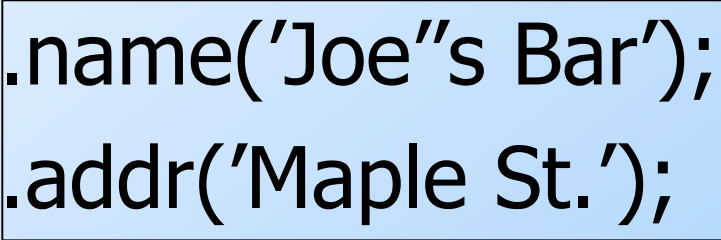
Inserting Values: SQL-99 Style

1. Create a variable X of the suitable type, using the constructor method for that type.
2. Use the mutator methods for the attributes to set the values of the fields of X .
3. Insert X into the relation.

Example: SQL-99 Insert

- ◆ The following must be part of a procedure, e.g., PSM, so we have a variable `newBar`.

```
SET newBar = BarType();  
newBar.name('Joe"s Bar');  
newBar.addr('Maple St.');
```



```
INSERT INTO Bars VALUES(newBar);
```

Mutator methods change `newBar`'s name and addr components.

UDT's as Column Types

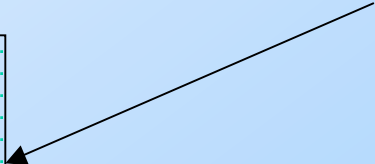
- ◆ A UDT can be the type of an attribute.
- ◆ In either another UDT definition, or in a CREATE TABLE statement, use the name of the UDT as the type of the attribute.

Example: Column Type

```
CREATE TYPE AddrType AS (  
    street    CHAR(30),  
    city      CHAR(20),  
    zip       INT  
);
```

```
CREATE TABLE Drinkers (  
    name      CHAR(30),  
    addr      AddrType,  
    favBeer   BeerType  
);
```

Values of addr and favBeer components are objects with 3 and 2 fields, respectively.



Oracle Problem With Field Access

- ◆ You can access a field F of an object that is the value of an attribute A by $A.F$.
- ◆ However, you must use an alias, say rr , for the relation R with attribute A , as $rr.A.F$.

Example: Field Access in Oracle

◆ Wrong:

```
SELECT favBeer.name  
FROM Drinkers;
```

◆ Wrong:

```
SELECT Drinkers.favBeer.name  
FROM Drinkers;
```

◆ Right:

```
SELECT dd.favBeer.name  
FROM Drinkers dd;
```

Following REF's

- ◆ $A \rightarrow B$ makes sense if:
 1. A is of type REF T .
 2. B is an attribute (component) of objects of type T .
- ◆ Denotes the value of the B component of the object pointed to by A .

Example: Following REF's

- ◆ Remember: Sells is a relation with rowtype MenuType(bar, beer, price), where bar and beer are REF's to objects of types BarType and BeerType.

- ◆ Find the beers served by Joe:

```
SELECT ss.beer()->name
```

```
FROM Sells ss
```

```
WHERE ss.bar()->name = 'Joe"s Bar';
```

Then use the arrow to get the names of the bar and beer referenced

First, use generator methods to access the bar and beer components

Following REF's: Oracle Style

- ◆ REF-following is implicit in the dot.
- ◆ Just follow a REF by a dot and a field of the object referred to.
- ◆ Example:

```
SELECT ss.beer.name  
FROM Sells ss  
WHERE ss.bar.name = 'Joe''s Bar';
```

Oracle's DEREFE Operator -- Motivation

- ◆ If we want the set of beer objects for the beers sold by Joe, we might try:

```
SELECT ss.beer
```

```
FROM Sells ss
```

```
WHERE ss.bar.name = 'Joe's Bar';
```

- ◆ Legal, but `ss.beer` is a REF, hence gibberish.

Using Deref

- ◆ To see the BeerType objects, use:
SELECT Deref(ss.beer)
FROM Sells ss
WHERE ss.bar.name = 'Joe''s Bar';
- ◆ Produces values like:
BeerType('Bud', 'Anheuser-Busch')

Methods --- Oracle Syntax

- ◆ Classes are more than structures; they may have methods.
- ◆ We'll study the Oracle syntax. Declare in `CREATE TYPE`, and define methods in a `CREATE TYPE BODY` statement.
 - ▶ Use PL/SQL syntax for methods.
 - ▶ Variable `SELF` refers to the object to which the method is applied.

Example: Method Declaration

- ◆ Let's add method priceInYen to MenuType.

```
CREATE TYPE MenuType AS OBJECT (  
  bar      REF BarType,  
  beer     REF BeerType,  
  price    FLOAT,  
  MEMBER FUNCTION priceInYen(rate IN FLOAT)  
  RETURN FLOAT,  
  PRAGMA RESTRICT_REFERENCES(priceInYen, WNDS)  
);  
/
```

What Oracle calls
methods.

"Write no database state."
That is, whatever priceInYen does
it won't modify the database.

Method Definition – Oracle Style

- ◆ Form of create-body statement:

```
CREATE TYPE BODY <type name> AS
```

```
  <method definitions = PL/SQL  
  procedure definitions, using
```

```
    "MEMBER FUNCTION" in place of
```

```
    "PROCEDURE">
```

```
END;
```

```
/
```

Example: Method Definition

```
CREATE TYPE BODY MenuType AS  
MEMBER FUNCTION
```

No mode (IN)
in body, just
in declaration

```
priceInYen(rate FLOAT) RETURN FLOAT IS
```

```
BEGIN
```

```
RETURN rate * SELF.price;
```

```
END;
```

```
END;
```

```
/
```

Use parentheses *only*
when there is at
least one argument

Method Use

- ◆ Follow a name for an object by a dot and the name of the method, with arguments if any.

- ◆ Example:

```
SELECT ss.beer.name,  
       ss.priceInYen(120.0)  
FROM Sells ss  
WHERE ss.bar.name = 'Joe''s Bar';
```

Order Methods: SQL-99

- ◆ Each UDT T may define two methods called EQUAL and LESSTHAN.
 - ▶ Each takes an argument of type T and is applied to another object of type T .
 - ▶ Returns TRUE if and only if the target object is = (resp. <) the argument object.
- ◆ Allows objects of type T to be compared by =, <, etc. in WHERE clauses and for sorting (ORDER BY).

Order Methods: Oracle

- ◆ We may declare any one method for any UDT to be an *order method*.
- ◆ The order method returns a value <0 , $=0$, or >0 , as the value of object SELF is $<$, $=$, or $>$ the argument object.

Example: Order Method Declaration

- ◆ Order BarType objects by name:

```
CREATE TYPE BarType AS OBJECT (  
    name      CHAR(20),  
    addr      CHAR(20),  
    ORDER MEMBER FUNCTION before(  
        bar2 IN BarType) RETURN INT,  
    PRAGMA RESTRICT_REFERENCES(before,  
        WNDS, RNDS, WNPS, RNPS)  
);  
/
```

Read/write no database state/package state. A "package" is a collection of procedures and variables that can communicate values among them.

Example: Order Method Definition

```
CREATE TYPE BODY BarType AS
  ORDER MEMBER FUNCTION
    before(bar2 BarType) RETURN INT IS
  BEGIN
    IF SELF.name < bar2.name THEN RETURN -1;
    ELSIF SELF.name = bar2.name THEN RETURN 0;
    ELSE RETURN 1;
    END IF;
  END;
END;
/
```

Oracle Nested Tables

- ◆ Allows values of tuple components to be whole relations.
- ◆ If T is a UDT, we can create a type S whose values are relations with rowtype T , by:

```
CREATE TYPE  $S$  AS TABLE OF  $T$ ;
```

Example: Nested Table Type

```
CREATE TYPE BeerType AS OBJECT (  
    name    CHAR(20),  
    kind    CHAR(10),  
    color   CHAR(10)  
);  
/  
CREATE TYPE BeerTableType AS  
    TABLE OF BeerType;  
/
```

Example --- Continued

- ◆ Use BeerTableType in a Manfs relation that stores the set of beers by each manufacturer in one tuple for that manufacturer.

```
CREATE TABLE Manfs (  
    name      CHAR(30),  
    addr      CHAR(50),  
    beers     beerTableType  
);
```

Storing Nested Relations

- ◆ Oracle doesn't really store each nested table as a separate relation --- it just makes it look that way.
- ◆ Rather, there is one relation R in which all the tuples of all the nested tables for one attribute A are stored.
- ◆ Declare in CREATE TABLE by:
NESTED TABLE A STORE AS R

Example: Storing Nested Tables

```
CREATE TABLE Manfs (  
    name    CHAR(30),  
    addr    CHAR(50),  
    beers   beerTableType
```

)

```
NESTED TABLE beers STORE AS BeerTable;
```

Note where the semicolon
goes and doesn't go.

Querying a Nested Table

- ◆ We can print the value of a nested table like any other value.
- ◆ But these values have two type constructors:
 1. For the table.
 2. For the type of tuples in the table.

Example: Query a Nested Table

- ◆ Find the beers by Anheuser-Busch:

```
SELECT beers FROM Manfs  
WHERE name = 'Anheuser-Busch';
```

- ◆ Produces one value like:

```
BeerTableType(  
  BeerType('Bud', 'lager', 'yellow'),  
  BeerType('Lite', 'malt', 'pale'),...  
)
```

Querying Within a Nested Table

- ◆ A nested table can be converted to an ordinary relation by applying THE(...).
- ◆ This relation can be used in FROM clauses like any other relation.

Example: Use of THE

- ◆ Find the ales made by Anheuser-Busch:

```
SELECT bb.name
```

```
FROM THE(
```

```
    SELECT beers
```

```
    FROM Manfs
```

```
    WHERE name = 'Anheuser-Busch'
```

```
) bb
```

```
WHERE bb.kind = 'ale';
```

The one nested table for the Anheuser-Busch beers

An alias for the nested table, which has no name

Turning Relations Into Nested Tables

- ◆ Any relation with the proper number and types of attributes can become the value of a nested table.
- ◆ Use `CAST(MULTISET(...) AS <type>)` on the relation to turn it into the value with the proper type for a nested table.

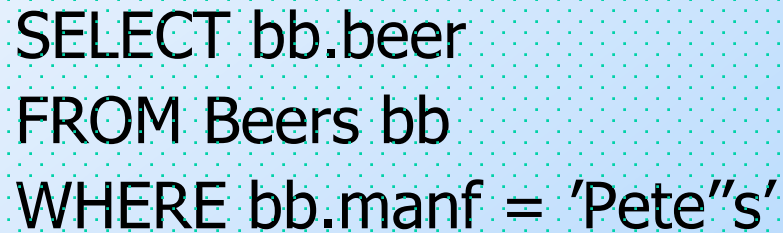
Example: CAST --- 1

- ◆ Suppose we have a relation $\text{Beers}(\text{beer}, \text{manf})$, where beer is a BeerType object and manf a string --- the manufacturer of the beer.
- ◆ We want to insert into Manfs a new tuple, with Pete's Brewing Co. as the name and a set of beers that are whatever Beers has for Pete's.

Example: CAST --- 2

```
INSERT INTO Manfs VALUES (  
  'Pete"s', 'Palo Alto',  
  CAST(  
    MULTISET(  
      SELECT bb.beer  
      FROM Beers bb  
      WHERE bb.manf = 'Pete"s'  
    ) AS BeerTableType  
  )  
);
```

The set of BeerType
objects for Pete's



```
SELECT bb.beer  
FROM Beers bb  
WHERE bb.manf = 'Pete"s'
```



```
) AS BeerTableType
```

Turn the set of objects
into a nested relation