Object-Relational Systems

- Object-oriented ideas enter the relational world.
  - ❖ Keep relation as the fundamental abstraction.
- Compare with "object-oriented DBMS," which uses the class as the fundamental abstraction and tacks on relations as one of many types.
  - ❖ ODL design language was covered; its query language OQL will be.

Motivations

- Allow DBMS's to deal with specialized types — maps, signals, images, etc. — with their own specialized methods.
- Supports specialized methods even on conventional relational data.
- Supports structure more complex than "flat files."

Slides to be used with Ullman/Widom *A First Course in Database Systems*, Prentice-Hall, 1997

Plan

1. Basic ideas from SQL3 standards documents.
2. Use Oracle 8 notation when similar.
3. Introduce some new concepts from Oracle 8.
   - ❖ On-line document: `or-objects.html`.

Two Levels of SQL3 Objects

1. For tuples of relations = "row types."
2. For columns of relations = "types" or "ADT's."

References

Row types can have *references*.

- If T is a row type, then `REF(`T`)` is the type of a reference to a T object.
- Unlike OO systems, refs are values that can be seen by queries.

Oracle 8 Approach

While SQL3 talks about types (for columns) and row-types (for relations), Oracle 8 uses object-types for both.

Example

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

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

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

- Note: in Oracle, type definitions must be followed by a slash (`/`) in order to get them to compile.

Creating Tables

Type declarations do not create tables.

- They are used in place of element lists in `CREATE TABLE` statements.

Example

```
CREATE TABLE Bars OF BarType;

CREATE TABLE Beers OF BeerType;

CREATE TABLE Sells OF MenuType;
```

Values of Object Types

Each object type (type defined with `AS OBJECT`) has a type constructor of the same name.

- Values of that type are the values of its fields wrapped in the constructor.

Example

```
SELECT * FROM Bars;
```

produces values such as

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

Accessing Fields of an Object

The dot operator works as expected.

- Thus, if we want the bar name and address without the constructor:

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

- The alias `bb` is not technically necessary, but there are other places where we must use an alias in order to access objects, and it is a good habit to use an alias always.

Inserting Values

We can use the standard `INSERT` in Oracle, but we must wrap the inserted object in its type-constructor.

Example

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

Types for Columns

In Oracle 8, the same object-type declaration can be the type of a column.

Example

Let's create an address type for use with bars and drinkers.

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

We can then create a table of drinkers that includes their name, address, and favorite beer.

- The beer is included as a beer object, rather than a reference, which "unnormalizes" the relation but is legal.

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

Need to Use Aliases

If you access an attribute whose type is an object type, you *must* use an alias for the relation. E.g.,

```
SELECT favBeer
FROM Drinker;
```

will not work. Neither will:

```
SELECT Drinker.favBeer
FROM Drinker;
```

You have to say:

```
SELECT dd.favBeer
FROM Drinker dd;
```

Dereferencing in SQL3

$A \rightarrow B$ = the B attribute of the object referred to by reference A.

Example

Find the beers served by Joe.

```
SELECT beer -> name
FROM Sells
WHERE bar -> name = 'Joe''s Bar';
```

Dereferencing in Oracle 8

- Dereferencing automatic, using dot operator.

Example

Same query in Oracle 8:

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

Oracle's DEREF Operator

If we wanted the entire `BeerType` object, we might try to write

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

That is legal, but `ss.beer` is a reference, and we'd get a gibberish value.

- To see the whole beer object, use:

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

Methods

Real reason object-relational isn't just nested structures in relations.

- We'll follow Oracle 8 syntax.
- Declared in a `CREATE TYPE` statement, defined in a `CREATE TYPE BODY` statement.
- Methods are functions or procedures; in Oracle they are defined like any PL/SQL function.
  - ❖ But, there is a special tuple variable `SELF` that refers to that object to which the method is applied.

Example

Let's add a method `priceInYen` to the `MenuType` and thus to the `Sells` relation.

```
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)
);
/

CREATE TYPE BODY MenuType AS
    MEMBER FUNCTION
            priceInYen(rate FLOAT) IS
        BEGIN
            RETURN rate * SELF.price;
        END;
END;
/

CREATE TABLE Sells OF MenuType;
```

Some Points to Remember

- The pragma is needed to allow `priceInYen` to be used in queries.
  - ❖ WNDS = "write no database state."
- In the declaration, function/procedure arguments need a mode, IN, OUT, or IN OUT, just like ODL method declarations.
  - ❖ But the mode does not appear in the definition.
- Many methods will take no arguments (relying on the built-in "self"). You need `()` after the method name anyway.
- The body can have any number of function declarations, separated by semicolons.

Example of Method Use

Follow a designator for the object to which you want to apply the method by a dot, the name of the method, and argument(s).

```
SELECT ss.beer.name,
       ss.priceInYen(117.0)
FROM Sells ss
WHERE ss.bar.name = 'Joe''s Bar';
```

Built-In Comparison Functions (SQL3)

We can define for each ADT two functions `EQUAL` and `LESSTHAN`.

- Allow values of this ADT to participate in `WHERE` clauses involving `=`, `<=`, etc. and in `ORDER-BY` sorting.

Order Methods in Oracle 8

We can declare one method for a type to be an `ORDER` method.

- Definition of this method must return <0, 0, >0, if "self" is less than, equal to, or greater than the argument object.
- Also used in comparisons for `WHERE` and `ORDER BY`.

Example

Order `BarType` objects by name.

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

```
CREATE TYPE BODY BarType AS
    ORDER MEMBER FUNCTION
            before(bar2 BarType) 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;
/
```

- The extra codes in the pragma guarantee no reading or writing of the database state or the "package state."