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.

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."

Plan

1. Basic ideas from SQL standards documents.
2. Use Oracle 8i/9i notation when similar.
3. Introduce some new concepts from Oracle.
   - On-line document: `or-objects.html`.

User-Defined Types

SQL allows *UDT*'s that play a dual role:

1. They can be the types of relations; i.e., the type of their tuple.
   - Sometimes called a *row type.*
2. They can be the type of an attribute in a relation.

Defining UDT's — Example and Oracle Syntax

```
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
);
/
```

Notes

- In Oracle, type definitions must be followed by a slash (`/`) in order to get them to compile.
- The SQL standard is similar, but "`OBJECT`" is not used after "`AS`."

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 User-Defined Types — Oracle Approach

- Each UDT 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 — Oracle Approach

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.

- SQL standard: Same idea, but the attribute is treated as a *generator* method, with parentheses, e.g., `bb.name()` .

Inserting Values — Oracle Approach

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.')
);
```

- SQL standard involves generator and mutator methods; see text.

Types for Columns

A UDT can also be the type of a column.

Example — Oracle Syntax

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, 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.name
FROM Drinker;
```

will not work in Oracle; neither will:

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

You have to say:

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

References

UDT's can have *references*.

- If T is a UDT, 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.

Dereferencing in SQL

$A \rightarrow B$ = 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

- Dereferencing automatic, using dot operator.

Example

Same query in Oracle syntax:

```
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 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 procedure or 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)
                RETURN 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 PL/SQL procedures.
  - But the mode does not appear in the definition.
- Many methods will take no arguments (relying on the built-in "self").
  - In that case, do not use parentheses after the function name.
- The body can have any number of function definitions, 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(120.0)
FROM Sells ss
WHERE ss.bar.name = 'Joe''s Bar';
```

Built-In Comparison Functions (SQL)

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

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

Order Methods in Oracle

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)
                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;
/
```

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