Objects in SQL3

OQL extends C++ with database concepts, while SQL3 extends SQL with OO concepts.

- Personal opinion: the relation is *so* fundamental to data manipulation that retaining it as the core, as SQL3 does, is "right."
  - Systems using the SQL3 philosophy are called *object-relational.*
  - All the major relational vendors have something of this kind, allowing any class to become the type of a column.

Informix	Data Blades
Oracle	Cartridges
Sybase	Plug-Ins
IBM/DB2	Extenders

Two Levels of SQL3 Objects

1. For tuples of relations = "row types."

2. For columns of relations = "types."

   - But row types can also be used as column types.

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.

Example of Row Types

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

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

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

Creating Tables

Row-type declarations do not create tables.

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

Example

```
CREATE TABLE Bars OF TYPE BarType

CREATE TABLE Beers OF TYPE BeerType

CREATE TABLE Sells OF TYPE MenuType
```

Dereferencing

$A \rightarrow B$ = 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';
```

OID's as Values

A row type can have a reference to itself.

- Serves as the OID for tuples of that type.

Example

```
CREATE ROW TYPE BarType (
    name CHAR(20),
    addr CHAR(20),
    barID REF(BarType)
);

CREATE TABLE Bars OF TYPE BarType
    VALUES FOR barID ARE SYSTEM
        GENERATED
```

- `VALUES...` clause forces the `barID` of each tuple to refer to the tuple itself.

name	addr	barID
Joe's	Maple St.	

Example: Using References as Values

Find the menu at Joe's.

```
SELECT Sells.beer->name, Sells.price
FROM Bars, Sells
WHERE Bars.name = 'Joe''s Bar' AND
    Bars.barID = Sells.bar;
```

ADT's in SQL3

Allows a column of a relation to have a type that is a "class," including methods.

- Intended application: data that doesn't fit relational model well, e.g., locations, signals, images, etc.
- The type itself is usually a multi-attribute tuple.
- Type declaration:

```
CREATE TYPE <name> (
    attributes
    method declarations or definitions
);
```

- Methods defined in a PL/SQL-like language.

Example

```
CREATE TYPE BeerADT (
    name CHAR(20),
    manf CHAR(20),

    FUNCTION newBeer(
            :n CHAR(20),
            :m CHAR(20)
        )
        RETURNS BeerADT;
    :b BeerADT; /* local decl. */
    BEGIN
        :b := BeerADT(); /* built-in
            constructor */
        :b.name := :n;
        :b.manf := :m;
        RETURN :b;
    END;

    FUNCTION getMinPrice(:b BeerADT)
        RETURNS FLOAT;
);
```

- `getMinPrice` is declaration only; `newBeer` is definition.

- `getMinPrice` must be defined somewhere where relation `Sells` is available.

```
FUNCTION getMinPrice(:b BeerADT)
    RETURNS FLOAT;
:p FLOAT;
BEGIN
    SELECT MIN(price) INTO :p
    FROM Sells
    WHERE beer->name = :b.name;
    RETURN :p;
END;
```

Built-In Comparison Functions

We can define for each ADT two functions `EQUAL` and `LESSTHAN` that allow values of this ADT to participate in `WHERE` clauses involving `=`, `<=`, etc.

Example: A "Point" ADT

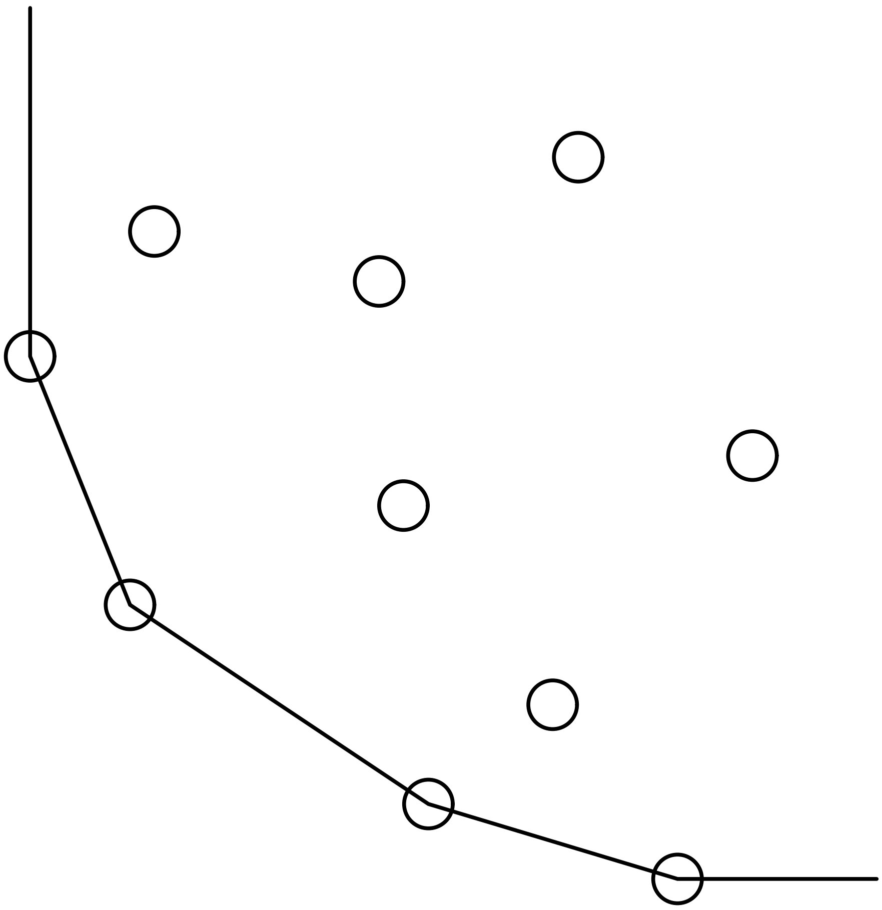
```
CREATE TYPE Point (
    x FLOAT,
    y FLOAT,

    FUNCTION EQUALS(
            :p Point,
            :q Point
        )
        RETURNS BOOLEAN;
    BEGIN
        IF :p.x = :q.x AND
                :p.y = :q.y THEN
            RETURN TRUE
        ELSE
            RETURN FALSE;
    END;
```

```
    FUNCTION LESSTHAN(
            :p Point,
            :q Point
        )
        RETURNS BOOLEAN;
    BEGIN
        IF :p.x > :q.x THEN
            RETURN FALSE
        ELSIF :p.x < :q.x THEN
            IF :p.y <= :q.y THEN
                RETURN TRUE
            ELSE RETURN FALSE
        ELSE /* :p.x = :q.x
            IF :p.y < :q.y THEN
                RETURN TRUE
            ELSE RETURN FALSE
    END;
);
```

Using the Comparison Functions

Here is a query that computes the lower convex hull of a set of points.

- Assumes `MyPoints(p)` is a relation with a single column p of type `Point`.



```
SELECT p
FROM MyPoints
WHERE NOT p > ANY MyPoints;
```