Chapter 10 Advanced topics in relational databases

- Security and user authorization in SQL
- Recursion in SQL
- Object-relational model

1. User-defined types in SQL
2. Operations on object-relational data

- Online analytic processing & data cubes

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.

Object-Relational Data Models

- Include object orientation and constructs to deal with added data types.
- Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
- **Preserve relational foundations**, in particular the declarative access to data, while extending modeling power.
- **Upward compatibility** with existing relational languages.

SQL-99

- SQL-99 includes many of the object-relational features to be described.
- However, different DBMS’s use different approaches.

User Defined Types

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

UDT Definition

```
CREATE TYPE <typename> AS (
    <list of attribute-type pairs>
);
```

Example: UDT Definition

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

Method Declarations in UDTs

```
CREATE TYPE BarType AS (
  name    CHAR(20),
  addr    CHAR(20))
   METHOD Telnumber() returns CHAR(10);

CREATE METHOD Telnumber() returns
  CHAR(10)
   FOR BarType
   Begin … End; // method body
```

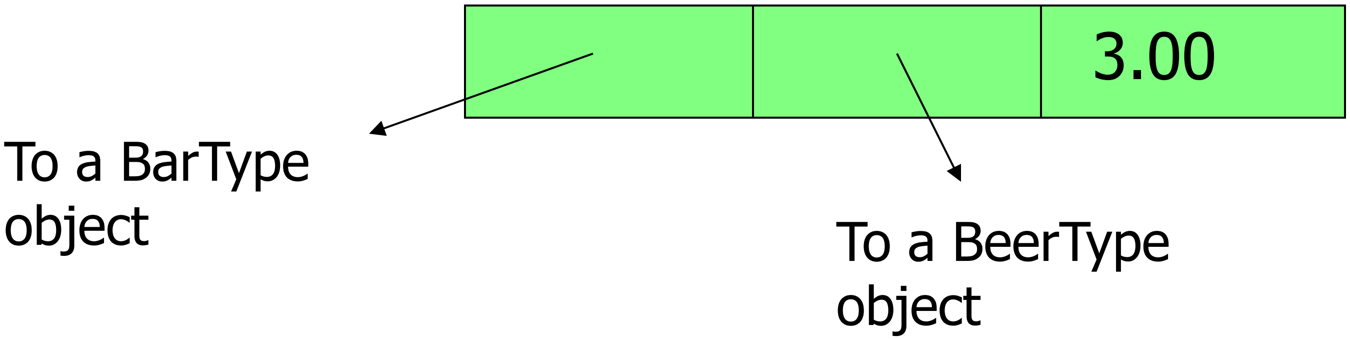
References

- If *T* is a type, then 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.

Example: REF

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

- MenuType objects look like:



3.00

To a BarType object

To a BeerType object

Example: REF (cont.)

- A REF(T) SCOPE R: A reference to tuples in relation R, where R is a table whose type is UDT T

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

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>
   (<list of elements>);
```

Example: Creating a Relation

```
CREATE TABLE Bars OF BarType (
  PRIMARY KEY (name));
CREATE TABLE Beers OF BeerType (
  PRIMARY KEY (name));
CREATE TABLE Sells OF MenuType (
  PRIMARY KEY (bar, beer),
  FOREIGN KEY (. . . ));
```

Constraints are elements of tables, not types.

Values of Relations with a Rowtype

- 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, which wraps objects of that type.

Example: Type Constructor

- The query

```
SELECT * FROM Bars;
```

- Produces “tuples” such as:

  BarType(’Joe’’s Bar’, ’Maple St.’)

Creating Objects ID's for Tables

REF IS <attribute name><how generated>

- SYSTEM GENERATED: DBMS is responsible for maintaining a unique value in the column.
- DERIVED: use primary key of the relation to produce unique values for the column.

For example:

```
CREATE TABLE Bars OF BarType (
  REF IS nameID SYSTEM GENERATED,
  primary key (name));
```

Accessing Values From a Rowtype

- In Oracle, the dot works as expected.
  - 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;

CREATE TABLE Bars OF BarType {
 PRIMARY KEY (name)};
```

Inserting Rowtype Values

- Oracle: use a standard INSERT statement.
  - But remember 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 declaration, 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
);
```

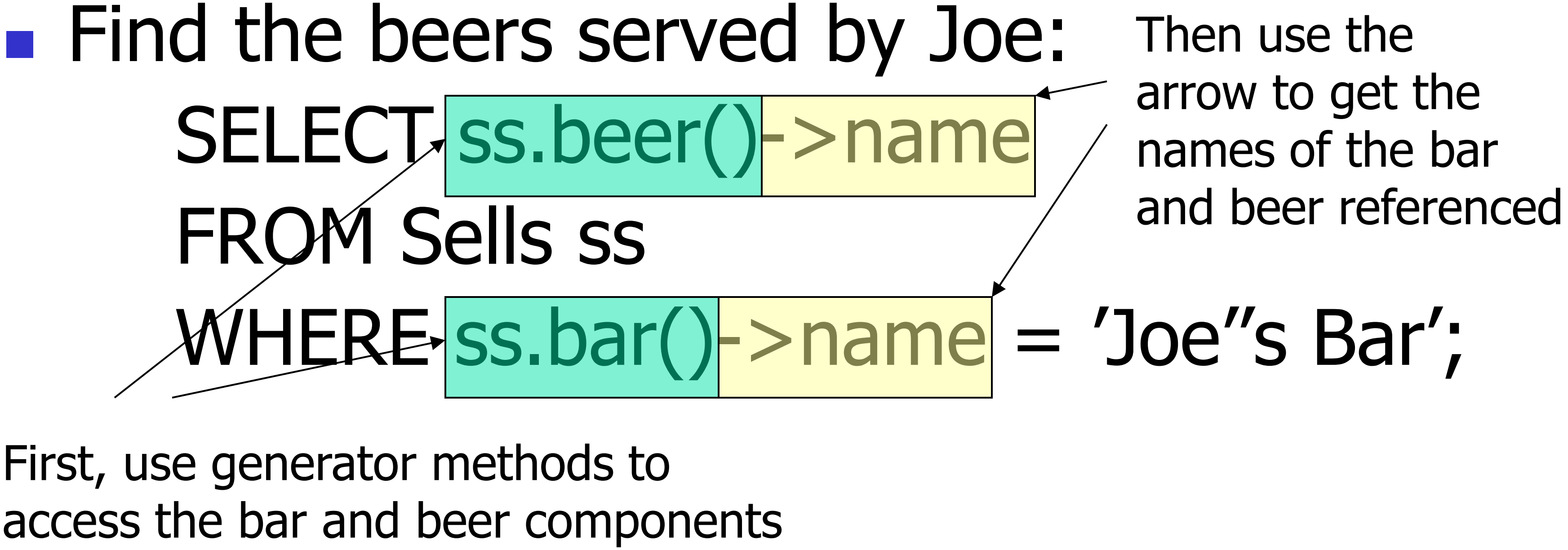
Values of addr and favBeer components are objects with 3 and 2 fields, respectively.

Following REF's: SQL-99 Style

- *A* -> *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

Using DEREF

- DEREF Applies to a reference and produces the tuple referenced.

- CREATE TABLE Sells (

```
 bar      REF    BarType,
beer      REF    BeerType,
price    FLOAT );
```

Instead of CREATE TABLE Sells OF MenuType

- To see the BeerType objects, use:

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

- Produces values like:

  BeerType('Bud', 'Anheuser-Busch')

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).

Ordering Relationships on UDT's

To specify an ordering or comparison:

- CREATE ORDERING FOR T EQUALS ONLY BY STATE;

  Two members of UDT T are considered equal if all of their corresponding components are equal.

- CREATE ORDERING FOR T

  ORDERING FULL BY RELATIVE WITH F

  apply the function F to these objects to do 6 comparisons (< <= > >= = <>), so that F(x1,x2) <0, means x1<x2, F(x1,x2)=0 means x1=x2, so on

```
CREATE ORDERING FOR AddressType
ORDERING FULL BY RELATIVE WITH AddrLEG (example)
   CREATE FUNCTION AddrLEG(
         x1 AddressType,
         x2 AddressType
   ) RETURNS INTEGER
   IF x1.city() < x2.city() THEN RETURN(-1)
   ELSEIF x1.city() > x2.city() THEN RETURN(1)
   ELSEIF x1.street() < x2.street() THEN RETURN(-1)
   ELSEIF x1.street() = x2.street() THEN RETURN(0)
   ELSE RETURN(1)
   END IF;
```

Summary

- UDT : User Defined Type

→ as the type of a table

→ as the type of an attribute

- Reference types: a type of an attribute can be a reference to a UDT.

→ A pointer to objects of that UDT.