CS422 Principles of Database Systems
Object-Oriented Features in DBMS

Chengyu Sun
California State University, Los Angeles

Bank Accounts Example

- Bank account
  - account number
  - balance
  - interests rate
  - creation date
  - owned by *one or more* customers
- Customer
  - id
  - name
  - address
  - *one or more* phones
- Phone
  - number
  - type
    - office, home, mobile

The Object-Oriented Approach

```
public class Account {
    int      acctNum;
    double   balance;
    double   interestsRate;
    Date     createdOn;
    List<Customer> owners;
};
```

```
public class Customer {
    int      customerId;
    String   name;
    String   address;
    List<Phones> phones;
};

public class Phone {
    String   number;
    String   type;
}
```

The Relational Approach

Accounts

acctNum	balance	interestsRate	createdOn

Customers

customerId	name	address

Phones

number	type

Customers_Accounts

customerId	acctNum

Customers_Phones

customerId	phoneNum

OO vs. Relational

- Composite types
- Collection types
- References
- And more …
  - *Methods* – operations that are associated with certain types
  - Inheritance and polymorphism

OO Features in DBMS

- Oracle OO features
  - Objects
  - Collections
- JDBC support for database objects

Object Type

```
create type phone_t as object (
        phone_number      char(7),
        phone_type        char(1)
);
/
```

◈ describe phone_t
◈ select * from user_types;

Object Column

```
create table customers (
        customer_id integer,
        name        varchar(15),
        address     varchar(15),
        phone       phone_t
);
```

Access Object Columns

```
insert into customers values
    (1,'Joe','123 Maple St.', phone_t ('1234567','H') );

select * from customers c
    where c.phone.phone_number = '1234567';
```

◈ Constructor of phone_t
◈ Table alias is required

Object Table

```
create type customer_t as object (
        customer_id integer,
        name        varchar(15),
        address     varchar(15),
        phone       phone_t
);
/
create table customers of customer_t;
```

◈ describe customers
◈ set describe depth {1|n|all}

Access Object Tables

◈ Object tables can be accessed as regular tables, or tables with a single column of an object type

```
insert into customers values
    (1, 'Joe', '123 Maple St.', phone_t('1234567','H'));

insert into customers values
    ( customer_t (2,'Sue','234 Main St.', phone_t('2345678','O')) );

select * from customers;
select value(c) from customers c;
```

Object Reference

```
create type account_t as object (
        account_id          integer,
        balance             number(10,2),
        interests_rate      number(4,2),
        created_on          date,
        owner               ref customer_t
);
/

create table accounts of account_t;
```

REF and DEREF

insert into accounts values
 (1,100.0,1.0,sysdate,
   (select ref(*c*) from customers c where customer_id = 1));

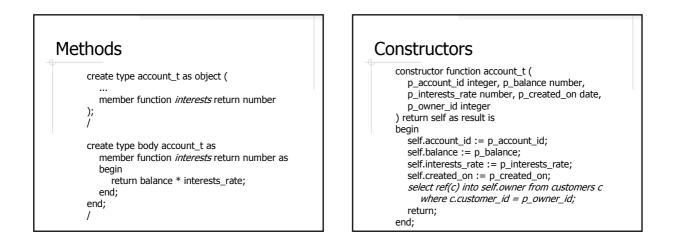select owner from accounts where account_id = 1;

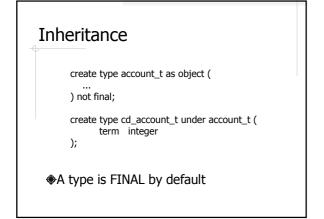select deref(*owner*) from accounts where account_id = 1;

◆Reference is implemented with an
  unique object id (OID)

Referential Integrity Constraint – OO Style

alter table accounts
    add (scope for (owner) is customers);

alter table accounts
    add foreign key (owner) references customers;

◈A reference can be scoped or unscoped
◈Scoped references are more efficient to use
  than unscoped ones
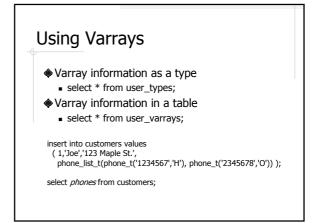◈Scoped references can still be *dangling*

Methods

create type account_t as object (
   ...
   member function *interests* return number
);
/

create type body account_t as
   member function *interests* return number as
   begin
      return balance * interests_rate;
   end;
end;
/

Constructors

constructor function account_t (
    p_account_id integer, p_balance number,
    p_interests_rate number, p_created_on date,
    p_owner_id integer
) return self as result is
begin
    self.account_id := p_account_id;
    self.balance := p_balance;
    self.interests_rate := p_interests_rate;
    self.created_on := p_created_on;
    select ref(c) into self.owner from customers c
        where c.customer_id = p_owner_id;
    return;
end;

Inheritance

create type account_t as object (
  ...
) not final;

create type cd_account_t under account_t (
      term   integer
);

◆A type is FINAL by default

Collection Types

◆Varrays
◆Nested tables

Varray

◈ Variable arrays, or varray
  - Array is bounded by a maximum size
  - All elements must be of the same type
  - *Elements can be accessed individually by index in a procedural language, but the array is treated as a whole in SQL.*

  create type phone_list_t as varray(10) of *phone_t*;

Using Varrays

◈ Varray information as a type
  - select * from user_types;
◈ Varray information in a table
  - select * from user_varrays;

  insert into customers values
    (1,'Joe','123 Maple St.',
      phone_list_t(phone_t('1234567','H'), phone_t('2345678','O')));

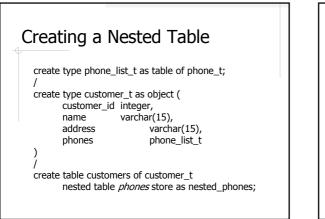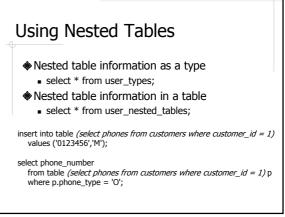 select *phones* from customers;

Nested Table

◈ A collection type in the form of *a table with a single column*
  - Each element is a row in the table
  - Any number of elements
  - Elements are of the same type
  - Each element can be accessed individually in SQL

A Nested Table Example

customer_id	name	address	phones	
1	Joe	123 Maple St.	**number**	**type**
			1234567	Home
			2345678	Office
2	Sue	234 Main St.	**number**	**type**
			7654321	Home
			8765432	Office
			0123456	Mobile

◈ Note that the nested table has a single column of a object type *phone_t*

Creating a Nested Table

```
create type phone_list_t as table of phone_t;
/
create type customer_t as object (
        customer_id  integer,
        name         varchar(15),
        address            varchar(15),
        phones             phone_list_t
)
/
create table customers of customer_t
        nested table phones store as nested_phones;
```

Using Nested Tables

◈ Nested table information as a type
  - select * from user_types;
◈ Nested table information in a table
  - select * from user_nested_tables;

  insert into table *(select phones from customers where customer_id = 1)*
    values ('0123456','M');

  select phone_number
    from table *(select phones from customers where customer_id = 1)* p
    where p.phone_type = 'O';

Varray vs. Nested Table

◈ Varray
- Ordered elements
- Max size
- Individual element accessible in PL
- Small varrays (<4k) are stored with parent table

◈ Nested table
- Unordered elements
- No max size
- Individual element accessible in SQL
- Always stored in separate tables

JDBC Support for Database Objects

◈ The Java class has to implement SQLData interface
- getSQLTypeName();
- readSQL(SQLInput stream, String typeName);
- writeSQL(SQLOutput stream);

◈ Update the JDBC Type Map
- connection.getTypeMap().put("FOO", Class.forName("Foo"));

◈ Resultset.getObject()

◈ PreparedStatement.setObject()