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

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

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

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

The Relational Approach

<table>
<thead>
<tr>
<th>Accounts</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>acctNum</td>
<td>balance</td>
<td>interestsRate</td>
<td>createdOn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customers</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>customerId</td>
<td>name</td>
<td>address</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phones</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>customerId</td>
<td>acctNum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customers_Accounts</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>customerId</td>
<td>acctNum</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customers_Phones</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>customerId</td>
<td>phoneNum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

- `describe phone_t`
- `select * from user_types;`

### Object Column

```sql
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**

### 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('2345 678','O')) );`
- `select * from customers;`
- `select value(c) from customers c;`

### Object Table

```sql
create table customer_t as object (  
    customer_id integer,  
    name varchar(15),  
    address varchar(15),  
    phone phone_t  
);
```

### Object Reference

```sql
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

Reference is implemented with a unique object id (OID)

```sql
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;
```

Referential Integrity Constraint – OO Style

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

```
alter table accounts
add (scope for (owner) is customers);
alter table accounts
add foreign key (owner) references customers;
```

Methods

```sql
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

```sql
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

```sql
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

<table>
<thead>
<tr>
<th>customer_id</th>
<th>name</th>
<th>address</th>
<th>phones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>123 Maple St.</td>
<td>Office 12345678</td>
</tr>
<tr>
<td>2</td>
<td>Sue</td>
<td>234 Main St.</td>
<td>Home 23456789, Office 12345678</td>
</tr>
</tbody>
</table>

◆ 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 (12345678, Office);

select phone_number
from table (select phones from customers where customer_id = 1)
  where p.phone_type = 'O';
<table>
<thead>
<tr>
<th>Varray vs. Nested Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Varray</strong></td>
</tr>
<tr>
<td>✓ Ordered elements</td>
</tr>
<tr>
<td>✓ Max size</td>
</tr>
<tr>
<td>✓ Individual element accessible in PL</td>
</tr>
<tr>
<td>✓ Small varrays (&lt;4k) are stored with parent table</td>
</tr>
<tr>
<td><strong>Nested table</strong></td>
</tr>
<tr>
<td>✓ Unordered elements</td>
</tr>
<tr>
<td>✓ No max size</td>
</tr>
<tr>
<td>✓ Individual element accessible in SQL</td>
</tr>
<tr>
<td>✓ Always stored in separate tables</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JDBC Support for Database Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The Java class has to implement SQLData interface</strong></td>
</tr>
<tr>
<td>✓ getSQLTypeName();</td>
</tr>
<tr>
<td>✓ readSQL(SQLInput stream, String typeName);</td>
</tr>
<tr>
<td>✓ writeSQL(SQLOutput stream);</td>
</tr>
<tr>
<td><strong>Update the JDBC Type Map</strong></td>
</tr>
<tr>
<td>✓ connection.getTypeMap().put(&quot;FOO&quot;, Class.forName(&quot;Foo&quot;));</td>
</tr>
<tr>
<td>✓ ResultSet.getObject()</td>
</tr>
<tr>
<td>✓ PreparedStatement.setObject()</td>
</tr>
</tbody>
</table>