Object-Oriented Systems

Within true object-oriented database systems, there are several directions.

1. Individual Systems (some no longer available):
   - O_2 / Ardent
   - GemStone
   - ObjectStore

2. In-house systems
   - Aircraft manufacturers

3. Standards:
   - ODMG (Object Database Management Group) proposals.

Other relevant standards:

- OMG (Object Management Group) standards.
  - OMA (Object Management Architecture)
    - CORBA (Common Object Request Broker Architecture)

A Closer Look at the ODMG Standard

Fundamental Ideas:

1. Do not base the design upon an existing model, such as the relational model. Rather, define database concepts "from scratch" within the object-oriented paradigm.

2. A stand-alone query language is not supported. Rather, the design depends upon a host object-oriented programming language.

3. Many such host languages (*e.g.,* C++, Smalltalk) should be supported.

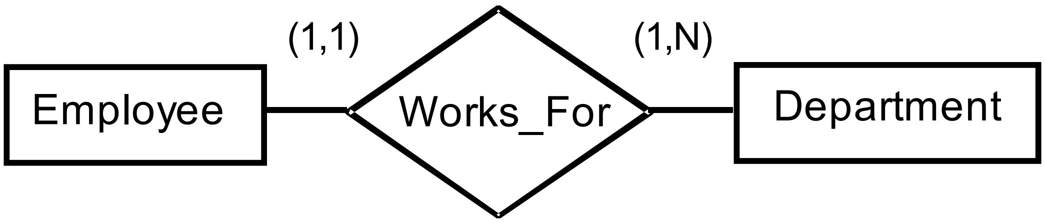
The details of this standard are substantial.

These slides will address only a few key points which identify ideas which are substantially different than those found in object-oriented programming languages.

Relationships:

In the ordinary relational model, relationships are hidden.

Example from the employee database:



In the relational model, this association must be known implicitly to the SQL query designer, who then extracts the appropriate information from the database.

In the ODBC standard, it may be made explicit.

Example, which shows some other features as well:

```
Class Employee
    (extent Employees
      key     SSN)
{
    attribute SSNType SSN;
    attribute NameType Name;
    attribute Date BDate;
    attribute AddressType Address;
    attribute enum Gender{M, F} Sex;
    attribute Int Salary;
    attribute Employee Supervisor;
    relationship <Department> Works_For
                    inverse Department::Workers;
    void give_raise(in int raise);
};

Class Department
     (extent Departments
       key     DName)
{
    attribute string DName;
    attribute integer DNumber;
    attribute Employee Manager;
    relationship set<Employee> Workers
                    inverse Employee::Works_For;
};
```

Other notes:

- *Extent* defines a collection of objects of that class.

Object Creation:

The simplest way to create a new named object is with the creator functions which are defined automatically when a new class is declared.
To create a new employee, one might proceed as follows in the host programming language

```
N1 = NameType(LName: "English";
              FName:  "Joyce;
              MInit: "A"
              );

A1 = AddressType(Street = "5131 Rice";
                 City = "Houston;
                 State = "TX";
                 Zip = "55555"
                 );

Joyce = Employee(Name: N1,
                 BDate: '1972-07-31',
                 Address: A1;
                 Sex: F;
                 Salary: 25000;
                 Supervisor: Wong;
                 Works_for: Research
                 );
```

This assumes:

- Appropriate NameType and AddressType classes have been declared.
- N1, A1, and Joyce are variables of the appropriate type.
- Research and Wong are objects of the appropriate type.

Objects may also be created using *factory objects* (see the text).

ODL and OQL:

- That which has just been sketched is part of the *Object Definition Language* (ODL).

- The ODMG framework also includes an SQL-like query language, the *Object Query Language* (OQL).
  - Queries must nonetheless be embedded within the host programming language; stand-alone queries are not supported.

Example:

```
SELECT D.Salary
FROM    D in Employees
WHERE D.Sex = F;
```

This query returns a *set* of values of type int, and may be assigned to a variable of that type.

Persistence:

In programming languages, one often speaks of the scope and extent of an entity.

Scope: The spatial region over which the entity is visible.

- Lexical scope
- Dynamic scope
- Indefinite scope

Extent: The temporal region over which the entity is visible.

- Dynamic extent
- Indefinite extent

In a programming language, *indefinite extent* (an object is visible throughout the lifetime of the execution of the program) is the widest possible.

In database systems, entities must generally exist far beyond the lifetime of execution of a single program.

The term *persistence* is used to describe this state of affairs.

An entity B is said to be *persistent*, relative to the execution E of a program, if either B exists before the program begins, or continues to exist after the program has completed execution.

- In programming languages, entities are generally *not* persistent.

- In database systems, the database itself is by its very nature persistent.

In a relational database system, persistence is a natural consequence of the design.

In an object-oriented database system, the principal database entities are objects. Although these database objects may appear to be very similar in structure to programming-language objects, persistence must be incorporated into the design.

There are two basic approaches:

Persistence by naming: The object is given a name which is understood beyond the program environment. (This is similar to naming a table and its attributes in the relational model.)

Persistence by reachability: In a large database system, it is impractical to name every object in the database. One may then have a system of references, in which an object is reachable if it is accessible by following a chain of references.

The Bottom Line:

Question: Is an ODBG system better than a relational system:

Answer: It depends upon what you are trying to do.

+: It is more flexible than a relational system, and modelling of complex ideas may be performed more naturally.

−: The user must now be a programmer. The level of sophistication required to use it is greater than for a relational system.