jACOB

Modeling Inheritance

Andreas Herz © 2005 sales@tarragon-software.de

Modeling Inheritance

Why?

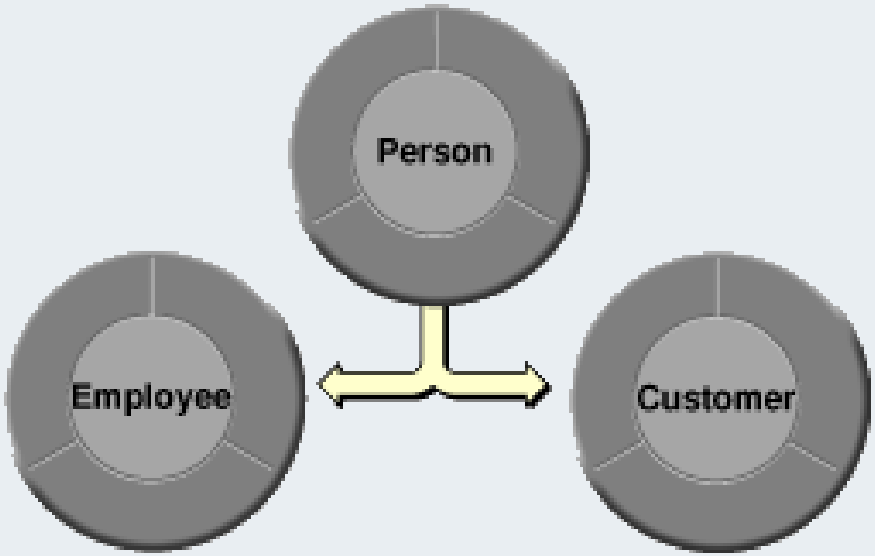
One of the issues that may arise in designing your enterprise objects—whether you're creating a schema from scratch or working with an existing database schema—is the modeling of ***inheritance relationships***.

A *Customer* object naturally inherits certain characteristics from a *Person* object, such as name, address, and phone number. In inheritance hierarchies, the parent object is usually rather generic so that less generic subclasses of a related type can easily be added. So, in addition to the *Customer* object, a *Client* object also naturally derives from a *Person* object.

Modeling Inheritance

How?

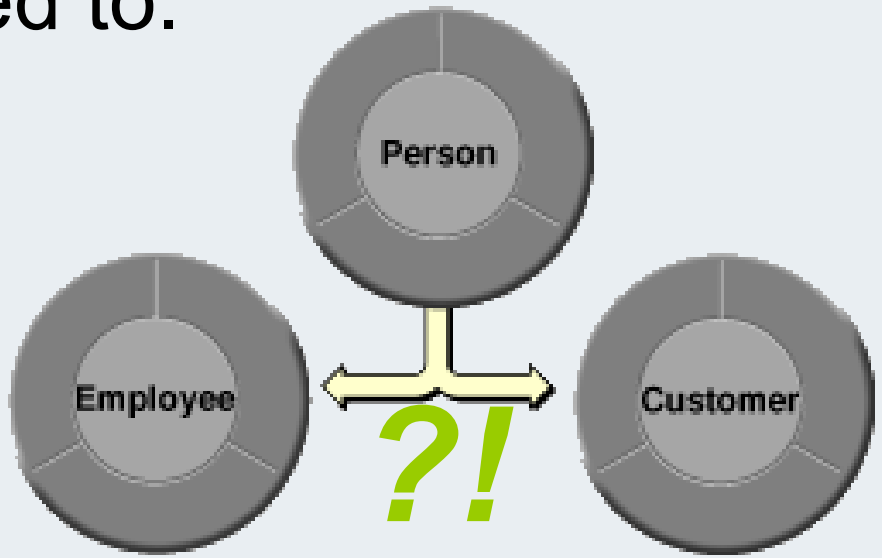
Using **jACOB**, you can build data models that reflect object hierarchies. That is, you can design database tables to support inheritance.



Modeling Inheritance

Deciding to Use

Using inheritance adds another level of complexity to your data model, data source, and thus to your application. While it has its advantages, you should use it only if you really need to.



Modeling Inheritance

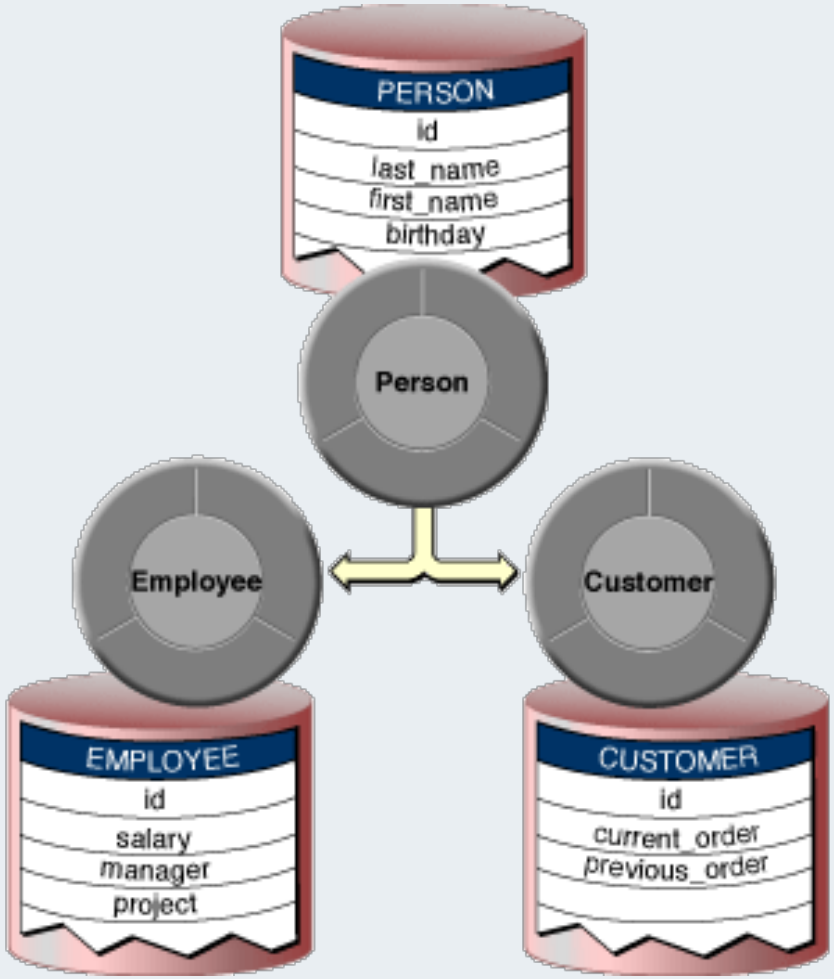
Three types

The three approaches of inheritance that can are supported by jACOB.

- Vertical Mapping (to be done)
- Horizontal Mapping (to de done)
- Single-Table Mapping

Modeling Inheritance

Vertical Mapping

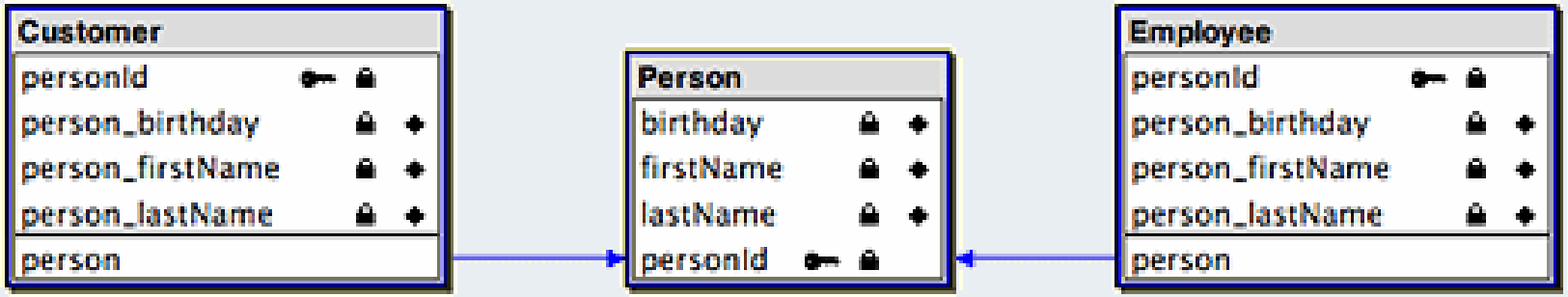


In this approach, each class is associated with a separate table. There is a *Person* table, an *Employee* table, and a *Customer* table. Each table contains only the attributes defined by that class.

Implementation:

- Create a *to-one* relationship from each child (*Employee* and *Customer*) to the parent (*Person*) entity.

- Add in the Inheritance tab of the child entity (*Employee*, *Customer*) the parent entity *Person*.



Advantages:

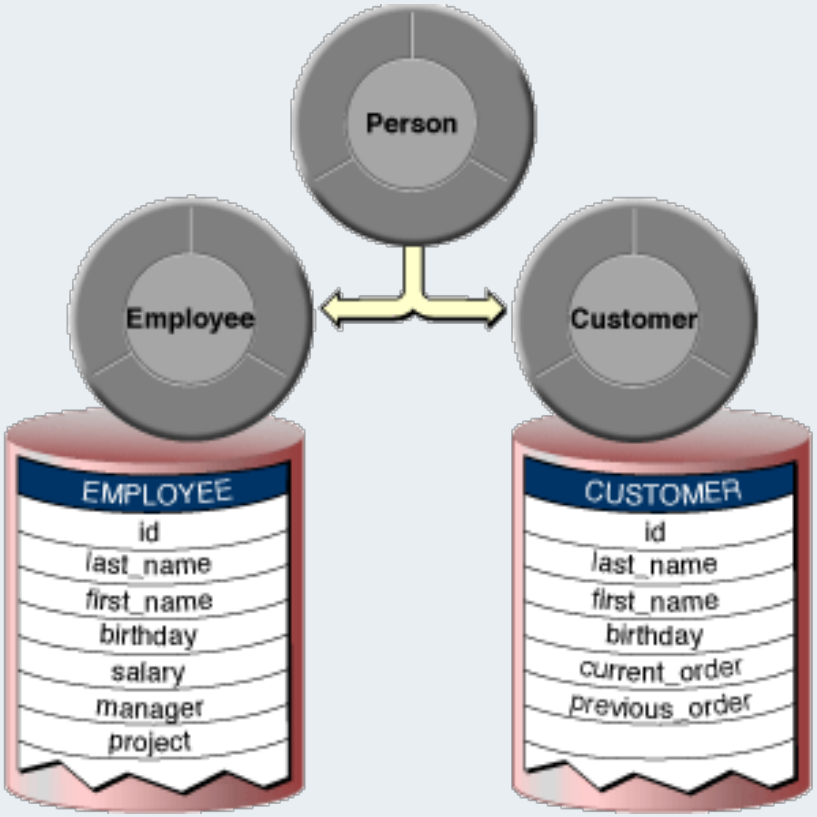
- A subclass can be added at any time without modifying the parent.
- Existing subclasses can be modifying without affecting others.
- Database constraints can be used at any level of inheritance.
- The primary vitue of this approach is its clean „normalized design“.

Disadvantages:

- Every layer of class hirarchy requires a join table to resolve the relationship.
- Each subclass record needs at least *n* database records. Where *n* is the level of inheritance.
- Complex database reconfiguration if inheritance changed!
- Slow search.

Modeling Inheritance

Horizontal Mapping



In this approach, you have separate tables for *Employee* and *Customer* that each contain columns for Person. The *Employee* and *Customer* tables contain not only their own attributes, but all of the *Person* attributes as well.

Implementation:

- Create a ghost entity *Person*.
- Add in the Inheritance tab of the child entity (*Employee*, *Customer*) the ghost parent (*Person*) entity.

All attributes of the ghost table will copied to all inheritance entities.

The ghost table will not be part of the physical database.

Modeling Inheritance

Horizontal *Mapping*

Advantages:

- a subclass can be added at any time without modifying the parent
- Existing subclasses can be modifying without affecting others
- It is the most efficient mapping approach if you fetch instances of only one leaf subclass at a time.

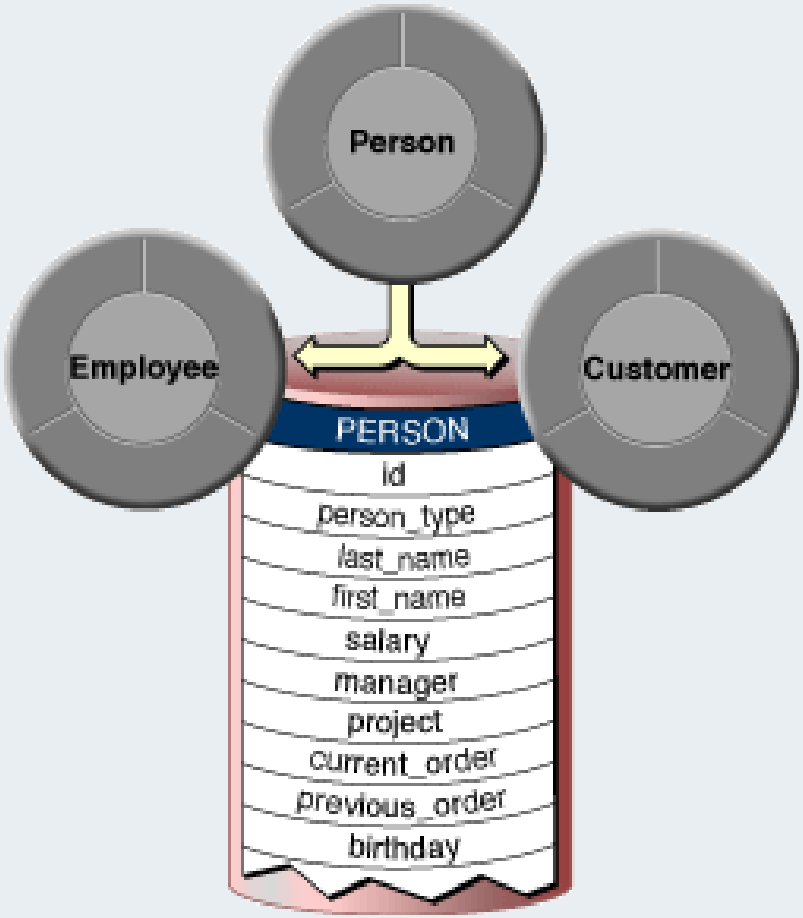
Disadvantages:

- complex search of parent entity. One single search per subclass.
- No *to-Any* relation possible from the parent entity (*Person*)
- The number of tables that need to be altered is equal to the number of subclasses.

jACOB

Modeling Inheritance

Single-Table Mapping



Put all of the data in one table that contains all superclass and subclass attributes. The attributes that don't apply for each object have *null* values. You fetch an Employee or Customer by using a query that returns just objects of the specified type (the table includes a *type* column to distinguish records of one type from the other)

Implementation:

- Create a entity Person with an int attribute `type`.
- Create a new table alias *Employee* from the entity *Person*
- In the Table Alias Editor, assign a restricting condition to the *Employee* entity that distinguishes its rows from the rows of other entities (`person.type=2`).
- Ensure via table hook that each new *Employee* record has the attribute type=2.

Modeling Inheritance

Single-Table Mapping

Advantages:

- This approach is faster than the other two methods for deep fetches.
- Unlike vertical or horizontal mapping, you can retrieve superclass (*Person*) objects with a single fetch, without performing joins.
- Adding a subclass or modifying the superclass requires changes to just one table.

Disadvantages:

- Single-table mapping results in tables that have columns for all of the attributes of each entity in the inheritance hierarchy.
- Unable to use the `not null` constraint on subclass attributes.
- It also results in many *null* row values.
- They may conflict with some database design philosophies.