Vrije Universiteit Brussel

Introduction to Databases

Object and Object-Relational Databases

Prof. Beat Signer

Department of Computer Science
Vrije Universiteit Brussel

http://vub.academia.edu/BeatSigner

Impedance Mismatch Revisited

- Combination of SQL with a host language
  - mix of *declarative* and *procedural* programming paradigms
  - two completely different data models
  - different set of data types
- Interfacing with SQL is not straightforward
  - data has to be converted between host language and SQL due to the *impedance mismatch*
  - *~30%* of the *code and effort* is used *for this conversion!*
- The problem gets even worse if we would like to use an object-oriented host language
  - two approaches to deal with the problem
    - *object databases* (object-oriented databases)
    - *object-relational databases*

Impedance Mismatch Revisited ...

```
public float getAverageCDLength() {
  float result = 0.0;
  try {
    Connection conn = this.openConnection();
    Statement s = conn.createStatement();
    ResultSet set = s.executeQuery("SELECT length FROM CD");
    int i = 0;
    while (set.next()) {
      result += set.getInt(1);
      i++;
    }
    return result/i;
  } catch (SQLException e) {
    System.out.println("Calculation of average length failed.");
    return 0;
  }
}
```

- Note that it would be easier to use the SQL AVG operator

Object Databases

- ODBMSs use the same data model as object-oriented programming languages
  - no object-relational impedance mismatch due to a uniform model
- An object database combines the features of an object-oriented language and a DBMS (*language binding*)
  - treat *data as objects*
    - object identity
    - attributes and methods
    - relationships between objects
  - *extensible type hierarchy*
    - inheritance, overloading and overriding as well as customised types
  - *declarative query language*

Persistent Programming Languages

- Several approaches have been proposed to make transient programming language objects persistent
  - *persistence by class*
    - declare that a class is persistent
    - all objects of a persistent class are persistent whereas objects of non-persistent classes are transient
    - not very flexible if we would like to have persistent and transient objects of a single class
    - many ODBMS provide a mechanism to make classes persistence capable
  - *persistence by creation*
    - introduce new syntax to create persistent objects
    - object is either persistent or transient depending on how it was created
  - *persistence by marking*
    - mark objects as persistent after creation but before the program terminates

Persistent Programming Languages ...

- *persistence by reachability*
    - one or more objects are explicitly declared as persistent objects (root objects)
    - all the other objects are persistent if they are reachable from a root object via a sequence of one or more references
    - easy to make entire data structures persistent

ObjectStore Example

- Persistence by reachability via specific database roots

```
Person ariane = new Person("Ariane Peeters")
db.createRoot("Persons", ariane);
```

- Persistence capable classes
  - post-processor makes specific classes persistent capable
- Persistent aware classes
  - can access and manipulate persistent objects (not persistent)
- Three states after a persistent object has been loaded
  - *hollow*: proxy with load on demand (lazy loading)
  - *active*: loaded in memory and flag set to clean
  - *stale*: no longer valid (e.g. after a commit)

ObjectStore Example ...

- Post processing
  1. compile all source files

```
javac *.java
```

  2. post-process the class files to generate annotated versions of the class files

```
osjcfp –dest . –inplace *.class
```

  3. run the post-processed main class

```
java mainClass
```

ODBMS History

- First generation ODBMS
  - 1986
    - G-Base (Graphael, F)
  - 1987
    - GemStone (Servio Corporation, USA)
  - 1988
    - Vbase (Ontologic)
    - Statice (Symbolics)
- Second generation ODBMS
  - 1989
    - Ontos (Ontos)
    - *ObjectStore* (Object Design)
    - *Objectivity* (Objectivity)
    - *Versant ODBMS* (Versant Object Technology)

ODBMS History ...

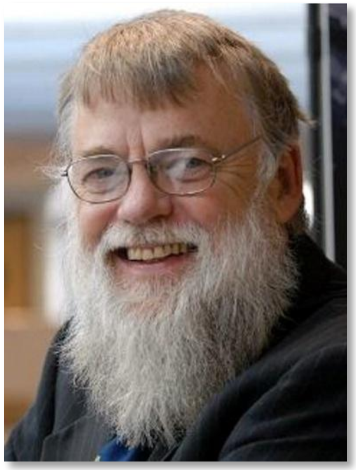
- 1989
    - *The Object-Oriented Database System Manifesto*
- Third generation ODBMS
    - 1990
        - Orion/Itasca (Microelectronis and Computer Technology Cooperation, USA)
        - O_2 (Altaïr, F)
        - Zeitgeist (Texas Instruments)
- Further developments
    - 1991
        - foundation of the Object Database Management Group (ODMG)
    - 1993
        - ODMG 1.0 standard

ODBMS History ...

- 1996
  - PJama (Persistent Java)
- 1997
  - ODMG 2.0 standard
- 1999
  - *ODMG 3.0 standard*
- 2001
  - *db4o* (database for objects)
- ...

The Object-Oriented Database Manifesto

- There have been different attempts to define object-oriented databases
- One of the efforts was the *Object-Oriented Database System Manifesto* by Atkinson et. al
  - defines 13 mandatory features that an object-oriented database system must have
    - 8 object-oriented system features
    - 5 DBMS features
  - optional features
    - multiple inheritance, type checking, versions, ...
  - open features
    - points where the designer can make a number of choices


Malcolm Atkinson

The Object-Oriented Database Manifesto ...

- Object-oriented system features
  - *complex objects*
    - complex objects built from simple ones by constructors (e.g. set, tuple and list)
    - constructors must be orthogonal
  - *object identity*
    - two objects can be *identical* (same object) or *equal* (same value)
  - *encapsulation*
    - distinction between interface and implementation
  - *types and classes*
    - type defines common features of a set of objects
    - class as a container for objects of the same type
  - *type and class hierarchies*
  - *overriding, overloading and late binding*

The Object-Oriented Database Manifesto ...

- - *computational completeness*
    - should be possible to express any computable function using the DML
  - *extensibility*
    - set of predefined types
    - no difference in usage of system and user-defined types
- DBMS features
  - *persistence*
    - orthogonal persistence (persistence capability does not depend on type)
  - *secondary storage management*
    - index management, data clustering, data buffering, access path selection and query optimisation
  - *concurrency*
    - atomicity, consistency, isolation and durability (ACID)
    - serialisability of operations

The Object-Oriented Database Manifesto ...

- - *recovery*
    - in case of hardware or software failures, the system should recover
  - *ad hoc query facility*
    - high-level declarative query language
- The OODBS Manifesto lead to discussion and reactions form the RDBMS community
  - *Third-Generation Database System Manifesto*, Stonebraker et al.
  - *The Third Manifesto*, Darwen and Date
- Issues not addressed in the manifesto
  - database evolution
  - constraints
  - object roles
  - ...

ODMG

- Object Database Management Group (ODMG) founded in 1991 by Rick Cattell
  - standardisation body including all major ODBMS vendors
- Define a standard to increase the portability accross different ODBMS products
  - *Object Model*
  - Object Definition Language (*ODL*)
  - Object Query Language (*OQL*)
  - *language bindings*
    - C++, Smalltalk and Java bindings

Rick Cattell

ODMG Object Model

- ODMG object model is based on the OMG object model
- Basic modelling primitives
  - *object*: unique identifier
  - *literal*: no identifier
- An object's *state* is defined by the values it carries for a set of properties (*attributes* or *relationships*)
- An object's *behaviour* is defined by the set of operations that can be executed
- Objects and literals are categorised by their *type* (common properties and common behaviour)

Types

- Specification
  - properties (attributes and relationships)
  - operations
  - exceptions
- Implementation
  - language binding
  - a specification can have more than one implementation

Type Specifications

- *Interface* defines only *abstract behaviour*
    - attribute declarations in an interface define only abstract behaviour (can be implemented as a method!)
- *Class* defines *abstract behaviour and abstract state*
- *Literal* defines *abstract state*

Objects

- Atomic objects
  - user defined
  - no built-in atomic object types
- Collection objects
  - `Set<t>`
  - `Bag<t>`
  - `List<t>`
  - `Array<t>`
  - `Dictionary<t,v>`
- Structured objects
  - `Date`, `Interval`, `Time`, `Timestamp`

Literal Types

- Atomic literals
  - `long`, `long long`, `short`, `unsigned long`, `unsigned short`, `float`, `double`, `boolean`, `octet`, `char`, `string`, `enum`
- Collection literals
  - `set<t>`
  - `bag<t>`
  - `list<t>`
  - `array<t>`
  - `dictionary<t,v>`
- Structured literals
  - `date`, `interval`, `time`, `timestamp`
  - user defined structures (`struct`)

Relationships

- *One-to-one*, *one-to-many* or *many-to-many* relationships with referential integrity maintained by the system

```
class Assistant {
  ...
  relationship set<ExerciseGroup> leads
    inverse ExerciseGroup::isLeadBy;
  ...
}

class ExerciseGroup {
  ...
  relationship Assistant isLeadBy
    inverse Assistant::leads;
  ...
}
```

Behaviour

- Behaviour is specified as a set of *operation signatures*
- An operation signature defines
  - name of the operation
  - names and types of arguments
  - type of return value
  - names of exceptions

Inheritance of Behaviour

- A subtype may
  - define new behaviour in addition to the one defined in its supertypes
  - refine a supertype's behaviour

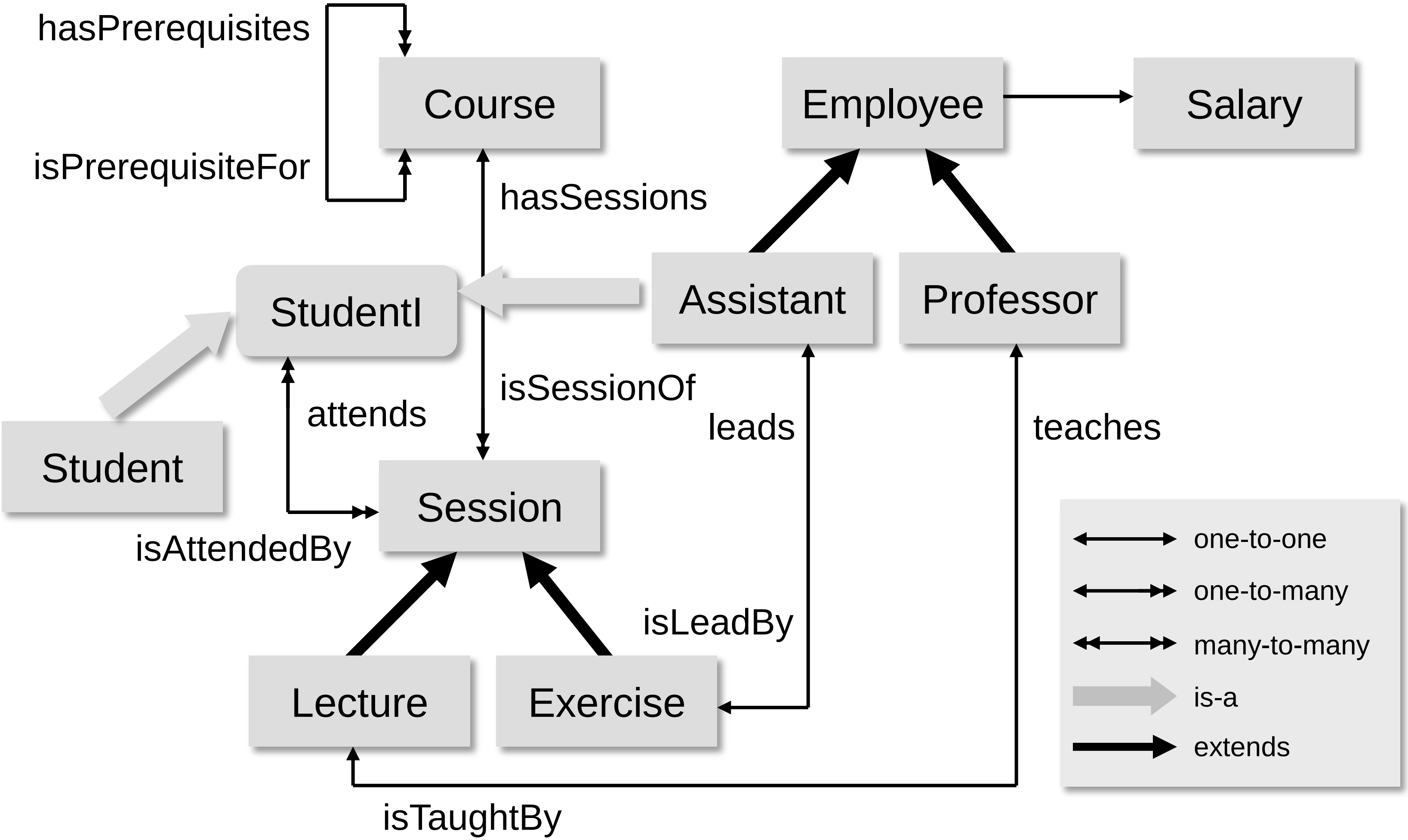
```
interface Contact {...}
interface Person : Contact {...}
interface ETHPerson : Person {...}
```

Inheritance of State and Behaviour

- Keyword EXTENDS
- A subclass inherits all the properties and behaviour of its superclass

```
interface Contact {...}
interface Student {...}
class Person : Contact {...}
class ETHPerson
  extends Person : Student {...}
```

Object Definition Language (ODL) Example



ODL Example ...

```
module Education {
  exception SessionFull{};
  ...

  class Course (extent courses) {
    attribute name;
    relationship Department offeredBy
      inverse Department::offers;
    relationship list<Session> hasSessions
      inverse Session::isSessionOf;
    relationship set<Course> hasPrerequisites
      inverse Course::isPrerequisiteFor;
    relationship set<Course> isPrerequisiteFor
      inverese Course::hasPrerequisites;
  };

  class Salary (extent salaries) {
    attribute float base;
    attribute float bonus;
  };
```

ODL Example ...

```
class Session (extent sessions) {
  attribute string number;
  relationship Course isSessionOf
    inverse Course::hasSessions;
  relationship set<Student> isAttendedBy
    inverse Student::attends;
};

class Lecture extends Session (extent lectures) {
  relationship Professor isTaughtBy
    inverse Professor::teaches;
};

class Exercise extends Session (extent exercises) {
  attribute unsigned short maxMembers;
  relationship Assistant isLeadBy
    inverse Assistant::leads;
};
```

ODL Example ...

```
interface StudentI {
  attribute string name;
  attribute Address address;
  relationship set<Session> attends
    inverse Session::isAttendeBy;
};

class Student : StudentI (extent students) {
  attribute Address address;
  relationship set<Session> attends
    inverse Session::isAttendedBy;
};

class Employee (extent employees) {
  attribute string name
  attribute Salary salary;
  void hire();
  void fire() raises (NoSuchEmployee);
};
```

ODL Example ...

```
class Professor extends Employee (extent professors) {
   attribute enum Type{assistant, full, ordinary} rank;
   relationship worksFor
     inverse Department:hasProfessors;
   relationship set<Lectures> teaches
     inverse Session::isTaughtBy;
 };

class Assistant extends Employee : StudentI (extent assistants) {
   attribute Address address;
   relationship Exercise leads
     inverse Exercise::isLeadBy
   relationship set<Session> attends
     inverse Session::isAttendedBy;
 };
```

"ODMG 4.0" Standard

- After the ODMG 3.0 standard the group disbanded
    - ODMG Java language binding formed basis for the *Java Data Objects (JDO)* specification
- The OMG *Object Database Technology Working Group* (ODBT WG) was founded in 2005 due to the new interest in object databases
- ODBT WG is now working on a fourth version of an object database standard

Object Databases

- Many ODBMS also implement a versioning mechanism
- Many operations are performed by using a navigational rather than a declarative interface
  - following pointers
- In addition, an object query language (OQL) can be used to retrieve objects in a declarative way
  - some systems (e.g. db4o) also support native queries
- Faster access than RDBMS for many tasks
  - no join operations required
- However, object databases lack a formal mathematical foundation!

Object-Relational Mapping

- "Automatic" mapping of object-oriented model to relational database
  - developer has to deal less with persistence-related programming
- Hibernate
  - mapping of Java types to SQL types
  - generates the required SQL statements behind the scene
  - standalone framework
- Java Persistence API (JPA)
  - Enterprise Java Beans Standard 3.0
  - use annotations to define mapping
  - `javax.persistence` package

Object-Relational Databases

- The object-relational data model extends the relational data model
  - introduces complex data types
  - object-oriented features
  - extended version of SQL to deal with the richer type system
- Complex data types
  - new collection types including multisets and arrays
  - attributes can no longer just contain atomic values (1NF) but also collections
  - `nest` and `unnest` operations for collection type attributes
  - ER concepts such as composite attributes or multivalued attributes can be directly represented in the object-relational data model

Object-Relational Databases ...

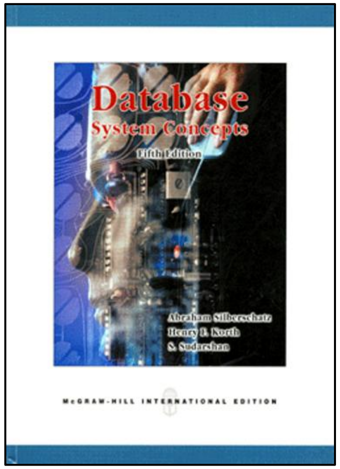
- Since SQL:1999 we can define user-defined types
- Type inheritance can be used for inheriting attributes of user-defined types

Object vs. Object-Relational Databases

- Object databases
  - complex datatypes
  - tight integration with an object-oriented programming language (persistent programming language)
  - high performance
- Object-relational databases
  - complex datatypes
  - powerful query languages
  - good protection of data from programming errors

Homework

- Study the following chapter of the *Database System Concepts* book
  - chapter 9
    - Object-Based Databases



Exercise 12

- General Q&A about previous exercises

References

- A. Silberschatz, H. Korth and S. Sudarshan, *Database System Concepts* (Fifth Edition), McGraw-Hill, 2005
- R. Cattell et al., *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann, 2000
- M. Atkinson et al., *The Object-Oriented Database System Manifesto*, Proceedings of 1st International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan, December 1989

Vrije Universiteit Brussel

Next Lecture

Current Trends and Review