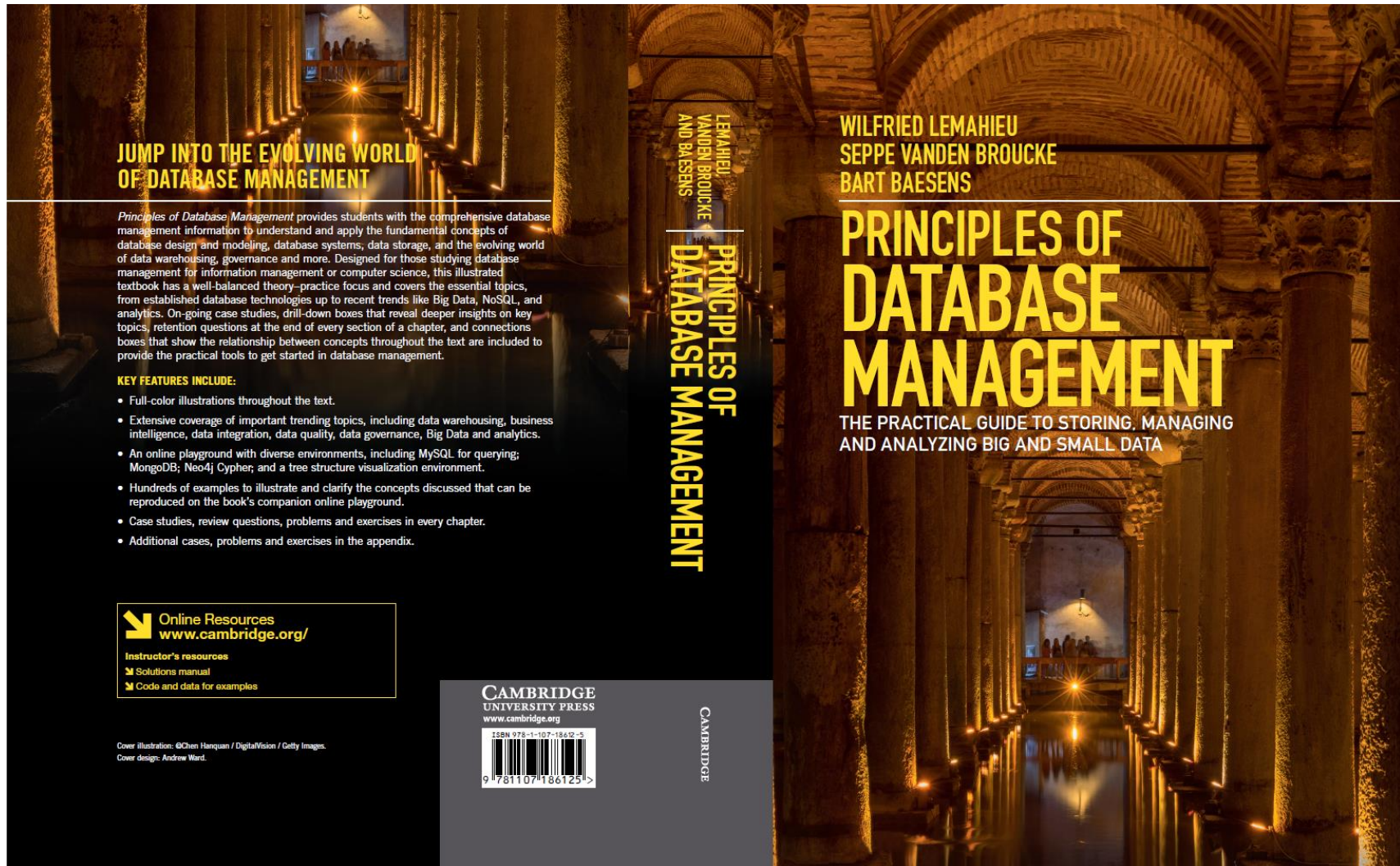


Object Oriented Databases and Object Persistence



www.pdbmbook.com

Introduction

- Recap: Basic Concepts of OO
- Advanced Concepts of OO
- Basic Principles of Object Persistence
- OODBMS
- Evaluating OODBMSs

Recap: Basic Concepts of OO

- Object is an instance of a class
- Class contains a blueprint description of all the object's characteristics
- Object bundles both variables (which determine its state) and methods (which determine its behavior) in a coherent way

Recap: Basic Concepts of OO

```
public class Employee {  
  
    private int EmployeeID;  
    private String Name;  
    private String Gender;  
    private Department Dep;  
  
    public int getEmployeeID() {  
        return EmployeeID;  
    }  
    public void setEmployeeID(  
        int id ) {  
        this.EmployeeID = id;  
    }  
    public String getName() {  
        return Name;}  
}
```

```
    public void setName( String  
        name ) {  
        this.Name = name;  
    }  
    public String getGender() {  
        return Gender;  
    }  
    public void setGender( String  
        gender ) {  
        this.Gender = gender;  
    }  
    public Department getDep() {  
        return Dep;  
    }  
    public void setDep(Department  
        dep) {this.Dep = dep;}}
```

Recap: Basic Concepts of OO

- Getter and setter methods implement the concept of information hiding (aka encapsulation)
- Encapsulation enforces a strict separation between interface and implementation.
 - interface consists of the signatures of the methods.
 - implementation is based upon the object's variables and method definitions

Recap: Basic Concepts of OO

```
public class EmployeeProgram {  
    public static void main(String[] args) {  
        Employee Bart = new Employee();  
        Employee Seppe = new Employee();  
        Employee Wilfried = new Employee();  
        Bart.setName("Bart Baesens");  
        Seppe.setName("Seppe vanden Broucke");  
        Wilfried.setName("Wilfried Lemahieu");  
    }  
}
```

Advanced Concepts of OO

- Method overloading
- Inheritance
- Method overriding
- Polymorphism
- Dynamic binding

Advanced Concepts of OO

- Method overloading refers to using the same name for more than one method in the same class.
- OO language environment can then determine which method you are calling, provided the number or type of parameters is different in each method

Advanced Concepts of OO

```
public class Book {  
    String title;  
    String author;  
    boolean isRead;  
    int numberOfReadings;  
  
    public void read(){  
        isRead = true;  
        numberOfReadings++;  
    }  
}
```

```
    public void read(int i){  
        isRead = true;  
        numberOfReadings +=  
        i;  
    }  
}
```

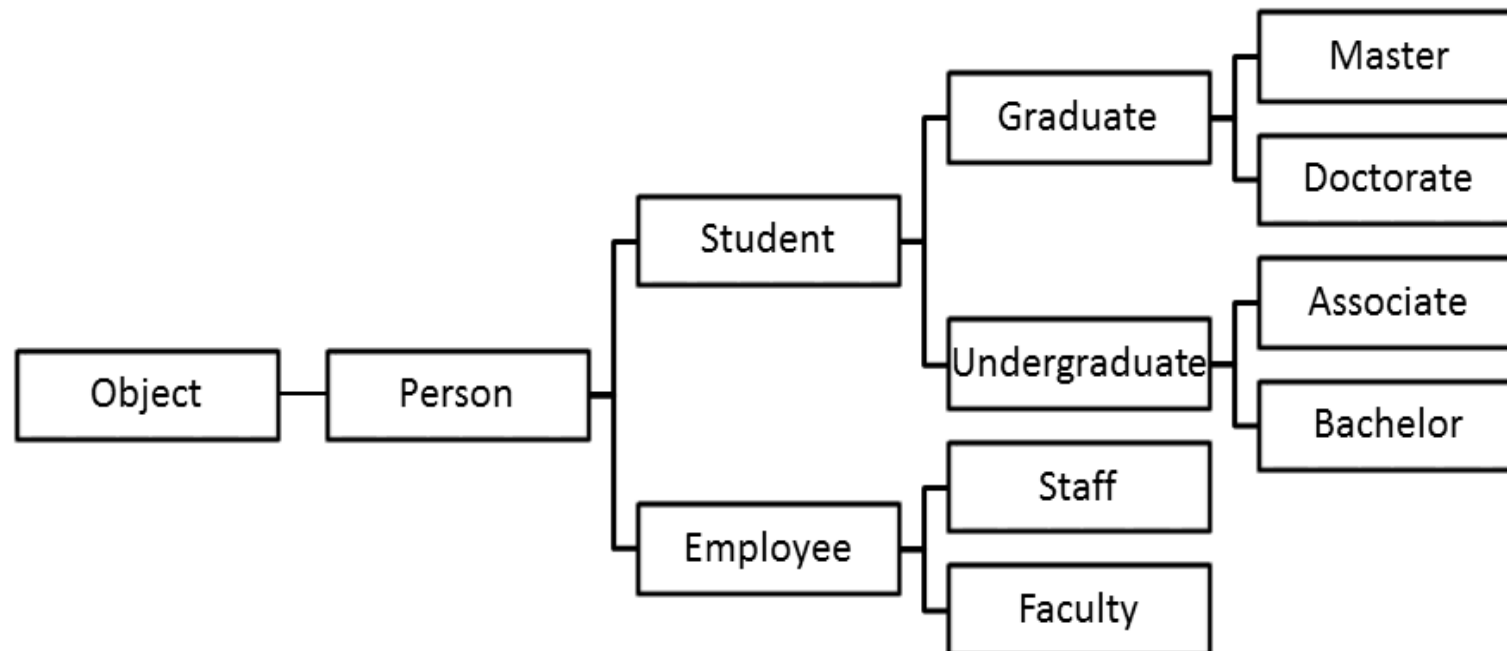
read(1) same effect as read()

Advanced Concepts of OO

- Method overloading is a handy feature when defining constructors for a class
- A **constructor** is a method which returns an object of a class
- Examples:
 - `Student(String name, int year, int month, int day)`
 - `Student(String name)`

Advanced Concepts of OO

- Inheritance represents an “is a” relationship
 - E.g. Student and Employee inherit from Person
 - Superclass versus Subclass



Advanced Concepts of OO

```
public class Person {  
    private String name;  
  
    public Person(String name){  
        this.setName(name);  
    }  
    public String getName(){  
        return this.name;  
    }  
    public void setName(String name){  
        this.name = name;  
    }  
}
```

```
public class Employee extends Person {  
    private Employee manager;  
    private int id;  
  
    public Employee(String name, Employee manager,  
        int empID) {  
        super(name);  
        this.setManager(manager);  
        this.setEmployeeID(empID);  
    }  
    public Employee getManager() {  
        return manager;  
    }  
    public void setManager(Employee manager) {  
        this.manager = manager;  
    }  
    public int getEmployeeID() {  
        return id;  
    }  
    private void setEmployeeID(int employeeID) {  
        this.id = employeeID;}}}
```

Advanced Concepts of OO

- Method overriding: subclasses can override an inherited method with a new, specialized implementation

Advanced Concepts of OO

Student Class

```
public double calculateGPA() {  
    double sum = 0;  
    int count = 0;  
    for (double grade : this.getGrades()){  
        sum += grade;  
        count++;  
    }  
    return sum/count;  
}
```

Graduate Class

```
public double calculateGPA(){  
    double sum = 0;  
    int count = 0;  
    for (double grade : this.getGrades()){  
        if (grade > 80){  
            sum += grade;  
            count++;  
        }  
    }  
    return sum/count;  
}
```

Advanced Concepts of OO

- Polymorphism refers to the ability of objects to respond differently to the same method
 - closely related to inheritance
 - depending on the functionality desired, the OO environment might consider a particular Master object as a Master, a Graduate, a Student, or a Person
- Static binding binds a method to its implementation at compile time
- Dynamic binding binds a method to its appropriate implementation at runtime, based on the object and its class.

Advanced Concepts of OO

```
public class PersonProgram {  
    public static void main(String[] args){  
        Student john = new Master("John Adams");  
        john.setGrades(0.75,0.82,0.91,0.69,0.79);  
        Student anne = new Associate("Anne Philips");  
        anne.setGrades(0.75,0.82,0.91,0.69,0.79);  
        System.out.println(john.getName() + ": " +  
            john.calculateGPA());  
        System.out.println(anne.getName() + ": " +  
            anne.calculateGPA());  
    }  
}
```

OUTPUT:

John Adams: 0.865

Anne Philips: 0.792

Basic Principles of Object Persistence

- Transient object is only needed during program execution and can be discarded when the program terminates
- Persistent object is an object that should survive program execution
- Persistence strategies:
 - Persistence by class
 - Persistence by creation
 - Persistence by marking
 - Persistence by inheritance
 - Persistence by reachability

Basic Principles of Object Persistence

- **Persistence by class** implies that all objects of a particular class will be made persistent
- **Persistence by creation** is achieved by extending the syntax for creating objects to indicate at compile-time that an object should be made persistent
- **Persistence by marking** implies that all objects will be created as transient. An object can then be marked as persistent during program execution

Basic Principles of Object Persistence

- **Persistence by inheritance** indicates that the persistence capabilities are inherited from a pre-defined persistent class
- **Persistence by reachability** starts by declaring the root persistent object(s). All objects that are referred to (either directly or indirectly) by the root object(s) will then be made persistent as well.

Basic Principles of Object Persistence

- Persistence orthogonality
 - persistence independence: persistence of an object is independent of how a program manipulates it
 - type orthogonality: all objects can be made persistent, irrespective of their type or size
 - transitive persistence: refers to persistence by reachability

Basic Principles of Object Persistence

- Persistent programming languages extend an OO language with a set of class libraries for object persistence
- Serialization translates an object's state into a format that can be stored (for example, in a file) and reconstructed later

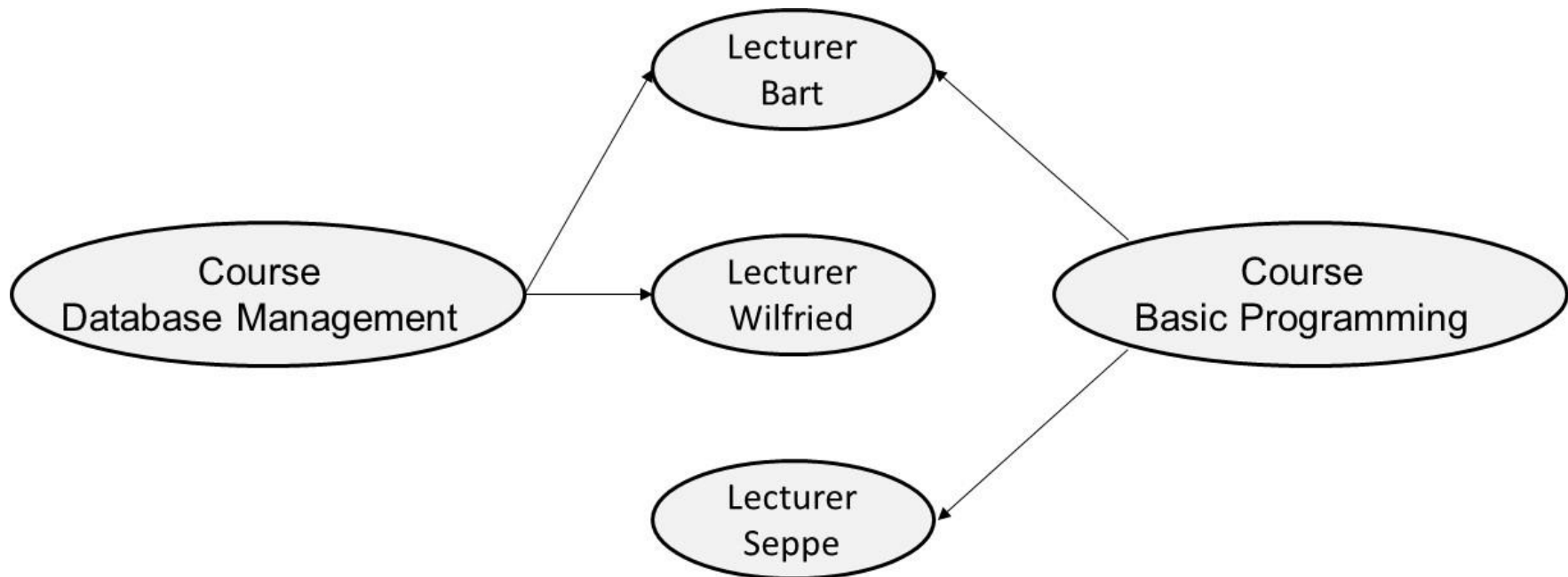
Basic Principles of Object Persistence

```
public class EmployeeProgram {  
    public static void main(String[] args) {  
        Employee Bart = new Employee();  
        Employee Seppe = new Employee();  
        Employee Wilfried = new Employee();  
        Bart.setName("Bart Baesens");  
        Seppe.setName("Seppe vanden Broucke");  
        Wilfried.setName("Wilfried Lemahieu");  
        try{  
            FileOutputStream fos = new FileOutputStream("myfile.ser");  
            ObjectOutputStream out = new ObjectOutputStream(fos);  
            out.writeObject(Bart);  
            out.writeObject(Seppe);  
            out.writeObject(Wilfried);  
            out.close;  
        }  
        catch (IOException e){e.printStackTrace();}  
    }  
}
```

persistence by reachability!

Basic Principles of Object Persistence

- Serialization suffers from the same disadvantages of the file based approach
- Lost object identity



OODBMS

- Object-oriented DBMSs (OODBMSs) store persistent objects in a transparent way
- OODBMSs originated as extensions of OO programming languages
- OODBMSs support persistence orthogonality
- OODBMSs guarantee the ACID properties

OODBMS

- Every object has a unique and immutable object identifier (OID)
 - Not dependent upon state of object (\leftrightarrow primary key)
 - Unique within entire OO environment (\leftrightarrow primary key)
 - Invisible to the user (\leftrightarrow primary key)
- OIDs are used to identify objects and to create and manage references between objects
- OO model is often referred to as an identity-based model
 - Relational model: value based model

OODBMS

- Two objects are said to be **equal** when the values of their variables are the same (object equality)
 - Shallow versus deep equality
- Two objects are said to be **identical** or equivalent when their OIDs are the same (object identity)

OODBMS

- The Object Database Management Group (ODMG) was formed in 1991 by a group of OO database vendors
 - Changed to Object Management Group (OMG) in 1998
- Promote portability and interoperability for object persistence by introducing a DDL and DML similar to SQL
- only one language for dealing with both transient and persistent objects

OODBMS

- OMG introduced 5 standards(most recent ODMG 3.0 in 2000) with following components:
 - **Object Model**: provides a standard object model for OODBMS
 - **Object Definition Language (ODL)**: specifies object definitions (classes and interfaces)
 - **Object Query Language (OQL)**: allows to define SELECT queries
 - **Language Bindings** (e.g., for C++, Smalltalk and Java): retrieve and manipulate object data.

OODBMS

- **Object Model** provides a common model to define classes, variables or attributes, behavior and object persistence.
- Two basic building blocks are objects and literals
- A **literal** does not have an OID and cannot exist on its own (\leftrightarrow an object)
- Types of literals: atomic, collection, structured

OODBMS

- **Atomic literals:** short (short integer), long (long integer), double (real number), float (real number), boolean (true or false), char, and string
- **Collection literals:**
 - Set: unordered collection of elements without duplicates
 - Bag: unordered collection of elements which may contain duplicates
 - List: ordered collection of elements
 - Array: ordered collection of elements which is indexed
 - Dictionary: unordered sequence of key-value pairs without duplicates

OODBMS

- A **structured literal** consists of a fixed number of named elements
- E.g., Date, Interval, Time and TimeStamp

```
struct Address{  
    string street;  
    integer number;  
    integer zipcode;  
    string city;  
    string state;  
    string country;  
};
```

OODBMS

- **Object Definition Language (ODL)** is a DDL to define the object types that conform to the ODMG Object Model

OODBMS

```
class EMPLOYEE
(extent employees
 key SSN)
{
attribute string SSN;
attribute string ENAME;
attribute struct ADDRESS;
attribute enum GENDER {male, female};
attribute date DATE_OF_BIRTH;
relationship set<EMPLOYEE> supervises
inverse EMPLOYEE:: supervised_by;
relationship EMPLOYEE supervised_by
inverse EMPLOYEE:: supervises;
relationship DEPARTMENT works_in
inverse DEPARTMENT:: workers;
relationship set<PROJECT> has_projects
inverse PROJECT:: has_employees;
string GET_SSN();
void SET_SSN(in string new_ssn);}
...
```

OODBMS

```
class MANAGER extends EMPLOYEE
(extent managers)
{
attribute date mgrdate;
relationship DEPARTMENT manages
inverse DEPARTMENT:: managed_by
}
```

```
class DEPARTMENT
(extent departments
 key DNR)
{
attribute string DNR;
attribute string DNAME;
attribute set<string> DLOCATION;
relationship set<EMPLOYEE> workers
inverse EMPLOYEE:: works_in;
relationship set<PROJECT>
assigned_to_projects
inverse PROJECT:: assigned_to_department
relationship MANAGER managed_by
inverse MANAGER:: manages;
string GET_DNR();
void SET_DNR(in string new_dnr);
...}
```

OODBMS

```
class PROJECT
(extent projects
key PNR)
{
attribute string PNR;
attribute string PNAME;
attribute string PDURATION;
relationship DEPARTMENT assigned_to_department
inverse DEPARTMENT:: assigned_to_projects;
relationship SET<EMPLOYEE> has_employees
inverse EMPLOYEE:: has_projects;
string GET_PNR();
void SET_PNR(in string new_pnr);
```

OODBMS

- A class is defined using the keyword **class**
- The **extent** of a class is the set of all current objects of the class
- A variable is declared using the keyword **attribute**
- Operations or methods can be defined by their name followed by parentheses
 - keywords **in**, **out**, and **inout** are used to define the input, output and input/output parameters
- **extends** keyword indicates the inheritance relationship

OODBMS

- Relationships can be defined using the keyword **relationship**.
- Only unary and binary relationships with cardinalities of 1:1, 1:N, or N:M are supported in ODMG.
- Ternary (or higher) relationships and relationship attributes need to be decomposed by introducing extra classes and relationships.

OODBMS

- Every relationship is defined in a bidirectional way, using the keyword **inverse**

```
relationship DEPARTMENT works_in  
inverse DEPARTMENT:: workers;
```

```
relationship set<EMPLOYEE> workers  
inverse EMPLOYEE:: works_in;
```

OODBMS

- N:M relationship can be implemented by defining collection types (e.g. set, bag)

```
relationship set<PROJECT> has_projects  
inverse PROJECT:: has_employees;
```

```
relationship SET<EMPLOYEE> has_employees  
inverse EMPLOYEE:: has_projects;
```

OODBMS

- **Object Query Language (OQL)** is a declarative, non-procedural query language
- OQL can be used for both navigational (procedural) as well as associative (declarative) access

OODBMS

- A **navigational query** explicitly navigates from one object to another

Bart.DATE_OF_BIRTH

Bart.ADDRESS

Bart.ADDRESS.CITY

OODBMS

- An **associative query** returns a collection (e.g., a set or bag) of objects which are located by the OODBMS.

Employees

OODBMS

- **SELECT... FROM ... WHERE** OQL queries
- OQL query returns a bag

```
SELECT e.SSN, e.ENAME, e.ADDRESS, e.GENDER  
FROM employees e  
WHERE e.name="Bart Baesens"
```

OODBMS

```
SELECT e.SSN, e.ENAME, e.ADDRESS,  
e.GENDER, e.age  
FROM employees e  
WHERE e.name="Bart Baesens"
```

```
SELECT e  
FROM employees e  
WHERE e.age > 40
```

OODBMS

- OQL join queries

```
SELECT e.SSN, e.ENAME, e.ADDRESS, e.GENDER, e.age
FROM employees e, e.works_in d
WHERE d.DNAME="ICT"
```

```
SELECT e1.ENAME, e1.age, d.DNAME, e2.ENAME,
e2.age
FROM employees e1, e1.works_in d, d.managed_by e2
WHERE e1.age > e2.age
```

OODBMS

count(employees)

```
SELECT e.SSN, e.ENAME  
FROM employees e  
WHERE EXISTS e IN (SELECT x FROM  
projects p WHERE p.has_employees x)
```

```
SELECT e.SSN, e.ENAME, e.salary  
FROM employees e
```

OODBMS

- ODMG language bindings provide implementations for the ODL and OQL specifications in popular OO programming languages (e.g. C++, Smalltalk or Java)
- Object Manipulation Language (OML) is kept language-specific
- E.g., for the Java language binding, this entails that Java's type system will also be used by the OODBMS, that the Java language syntax is respected and that the OODBMS should handle management aspects based on Java's object semantics

Evaluating OODBMSs

- Complex objects and relationships are stored in a transparent way (no impedance mismatch!)
- Success of OODBMSs has been limited to niche applications
 - E.g., processing of scientific data sets by CERN
- Disadvantages
 - the (ad-hoc) query formulation and optimization procedures
 - robustness, security, scalability and fault-tolerance
 - no transparent implementation of the 3 layer database architecture (e.g. views)

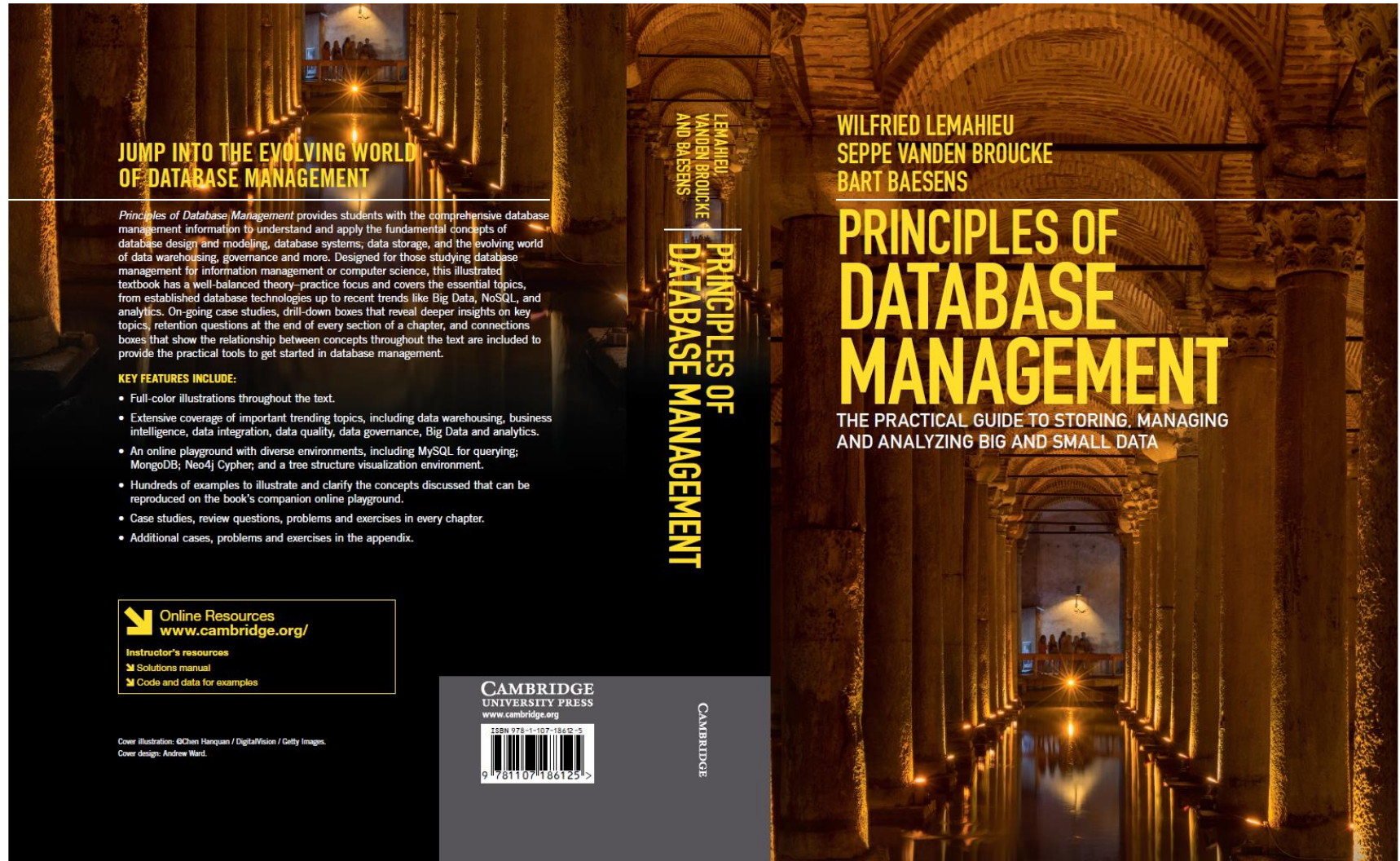
Evaluating OODBMSs

- Most mainstream database applications will, however, typically be built using an OO programming language in combination with an RDBMS
- **Object Relational Mapping (ORM)** framework is used as middleware to facilitate the communication between both environments: OO host language and RDBMS

Conclusions

- Recap: Basic Concepts of OO
- Advanced Concepts of OO
- Basic Principles of Object Persistence
- OODBMS
- Evaluating OODBMSs

More information?



www.pdbmbook.com