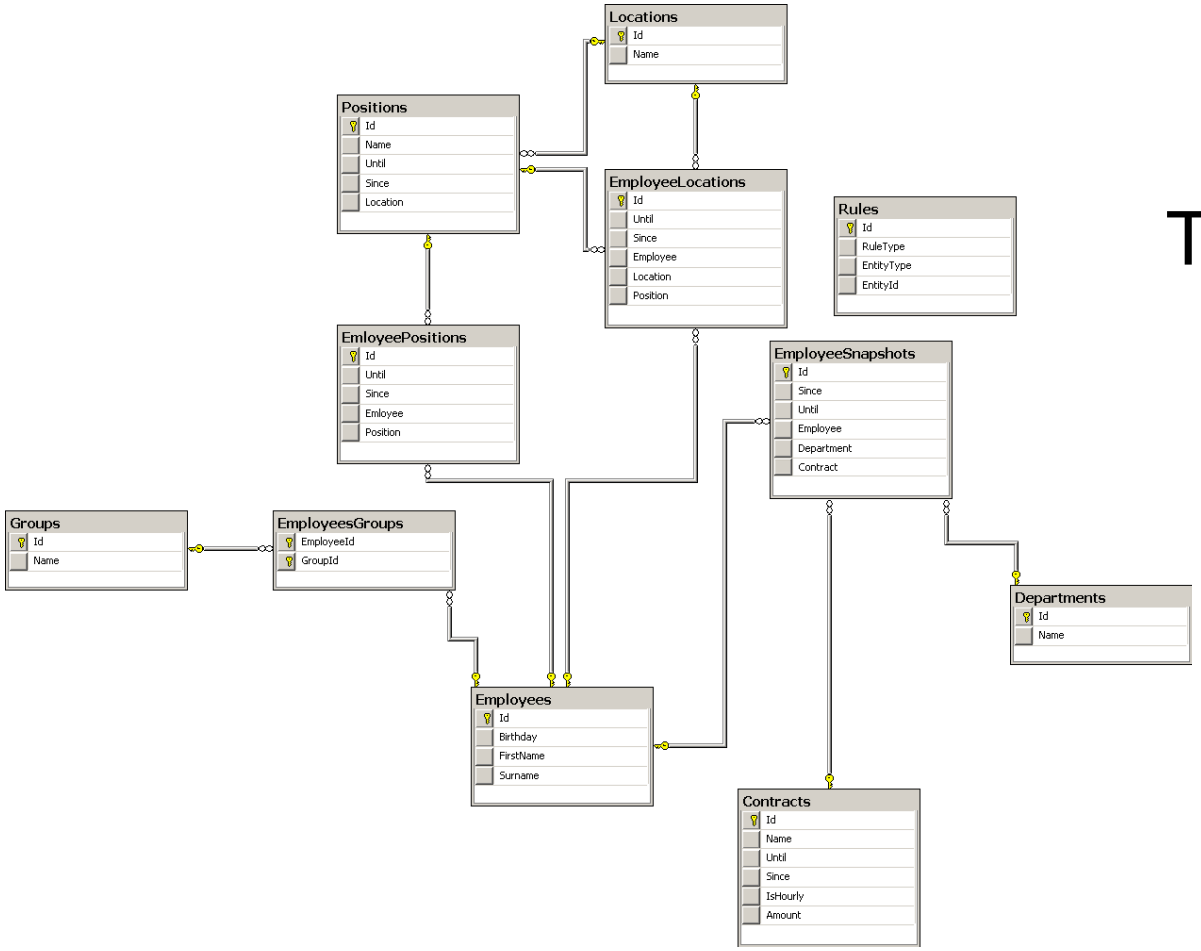
UiO : University of Oslo

Object and object-relational databases

Relational databases



Terms and concepts:

- *tuple*
- *domain*
- *attribute*
- *key*
- *integrity rules*

Relational Databases
Relational algebra

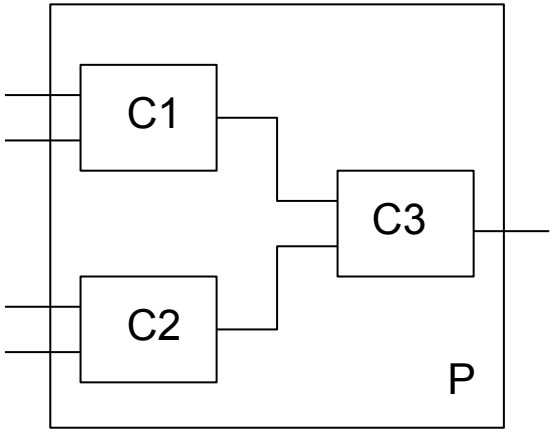
- Solid mathematical background
- Simple, powerful, expressive
- Non-procedural (i.e., declarative)
- Data independent – good for optimization

Relational Databases
What is the problem?

- Lack of complex structures (e.g., engineering design, manufacturing, science etc.)
- Semantic gap: vast increase in object-oriented programming languages (e.g., Java, C++ etc.)
- Unnatural: flattening and scattering (i.e., *normalization/join*)
- Traversal: expensive due to joins
- Impedance mismatch!

Relational Databases

What is the problem?



Gate Instance	GT	QI	Parent
	2AND	C1	4AND
	2AND	C2	4AND
	2AND	C3	4AND
	4AND		

Pin type	GT	PT	I/O
	2AND	A	I
	2AND	B	I
	2AND	C	O
	4AND	A	I
	4AND	B	I
	4AND	C	I
	4AND	D	I
	4AND	E	O

Gate Type	GT	Description
	2AND	C=A&B
	4AND	E=A&B&C&D

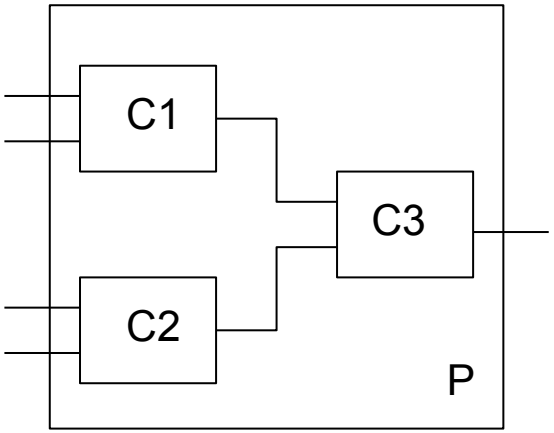
Wire Instance	WI	GT1	GI1	…
	W1	4AND	P	A
	W2	4AND	P	B
	W3	4AND	P	C
	W4	4AND	P	D
	W5	2AND	C1	C
		…		

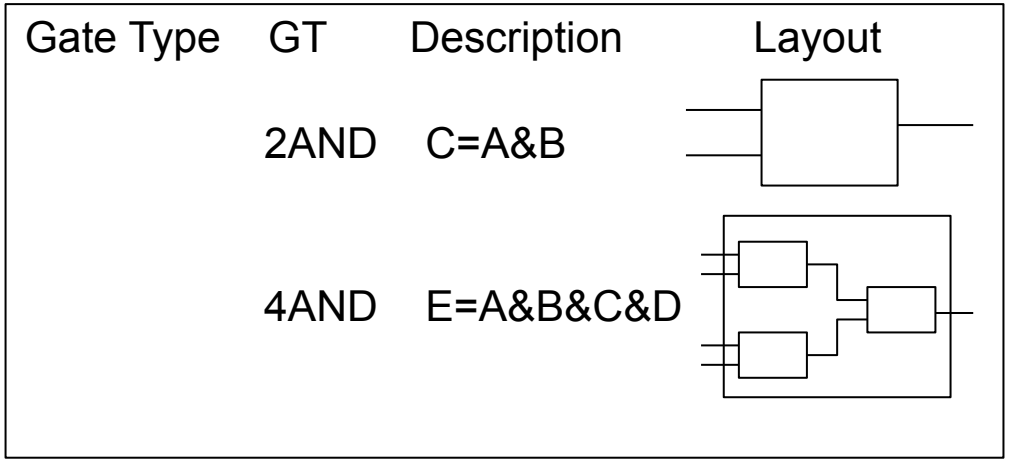
(barrowed from Frank Stajono, 1998 - ORL Technical Report TR-98-2)

Relational Databases

What is the problem?

- *Binary Large Objects* (BLOBS)
  - internal structure inaccessible
  - no type checking





Object-oriented approaches

- The origin: *object-oriented programming languages* (OOPL)
  - Pure OOPLs
    - SIMULA language (late 1960s)
    - Smalltalk (1970s) with additional *object-oriented* (OO) concepts – purely OO
  - Hybrid OOPLs
    - C++ incorporates OO concepts into C programming language

Object-oriented approaches

Object-oriented approach is designed to model a domain of interest (often called mini world) as a collection of communicating and cooperating entities (entities) that we call objects.

Object-oriented approaches

What is a class and an object?

A class

```
public class Bicycle {

    public int cadence;
    public int gear;
    public int speed;

    public void applyBrake(int decrement){
        speed -= decrement; }

    public void speedUp(int increment) {
        speed += increment; }
}
```

…is an abstract description of a concept in real world

An object

```
JohnsBicycle = new Bicycle();
```

… is an instance/member of a class

Object-oriented approaches
More about objects

- An object has two components: **state** (value) and **behavior** (operations)
- *Instance variables* – defines the internal state of an object (similar to attributes)
- Operations – *signature/interface* and *method/ body*
- *Transient objects* (exist during program execution) vs. *persistent objects* (stored)

Object oriented features

- Object identity and object identifier (OID)
- Objects versus literals
- Complex type structures and type constructors
- Operations and Encapsulation
- Type hierarchies and inheritance
- Polymorphism and operator overloading
- Multiple inheritance and selective inheritance
- Programming language compatibility

Object oriented features
Object identity and identifier

- Direct-correspondence between real world and database objects
  - integrity and identity preserved
  - can be easily identified and operated upon
- A system generated unique identity – **object identifier** (OID)
  - used internally, not visible to external user
  - identify each object and create manage inter-object references

Object oriented features
Object identity and identifier

- A main property of OID: **Immutable**
  - OID of a particular object should not change
  - each OID is used only at once
- What if OID depends on any attribute values of the object?
  - value of an attribute might change
- What if OID is based on physical address of the storage?
  - physical address might change

Object oriented features
Objects versus literals

- Some early OO models required that everything - from a simple value to complex object – was represented as an object
  - every basic value such as Boolean, integer string has an OID
  - two identical basic values have different OIDs, e.g., 50 as a weight and as the age of a person
  - not very practical, so we have **literals** (values) with no OID and can not be referenced

Object oriented features
Complex type structures

- Type structure of an arbitrary complexity for objects and literals
  - recall traditional database systems (scattered, loss of direct correspondence with the real-world object)
- Complex type may be constructed from other types by nesting type constructors
  - basic constructors: **atom**, **struct** (type generator), and **collection**

Object oriented features
Complex type structures

- **Atom** types (single-valued or atomic): basic built-in data of the object model such as integers, strings, and Booleans.
- **Struct**: composite or compound structures
  - many different structures can be generated
- **Collection**: allows a part of an object or literal value to include a collection of other objects and values: **set**, **bag, list**, **array**, **dictionary**
- Complex types can be used to define data structures

Object oriented features

Complex type structures

```
define type EMPLOYEE
        struct   ( Fname:          string;          ──────────→ atom
                   Lname:          string;
                   Birth_Date:     DATE;
                   Salary:         float;
                   Supervisor      EMPLOYEE;
                   Dept:           DEPARTMENT; );   ──────────→ object reference

define type DATE
        struct   ( Year:           integer;
                   Month:          integer;
                   Day:            integer; );

define type DEPARTMENT
        struct (   Dname:          string;
                   Locations:      set(string);     ──────────→ collection
                   Employees:      set(EMPLOYEE);   ──────────→ inverse reference
                   Mgr:            struct( Manager:  Employee
                                           StatDate: Date;);
                 );
```

Object oriented features
Operations and Encapsulation

- **Operations** (methods)**:** to manipulate object, retrieve/update object state, access data, or to apply some calculations
  - external world only knows the **interface** (signature) of operations (name and parameters)
  - implementation is hidden
- Recall the traditional databases where generic database operations are applicable to objects of all types

Object oriented features
Operations and Encapsulation

- **Encapsulation**: one of the main characteristics of OO languages and systems
- as an *information hiding* mechanism: internal representation of an object is generally hidden from view outside of the object's definition
- only the object's own methods can directly inspect or manipulate its fields.
  - complete encapsulation: too stringent
  - visible and hidden attributes

Object oriented features

Operations and Encapsulation

```
define type EMPLOYEE
        struct   ( Fname:          string;
                   Lname:          string;
                   Birth_Date:     DATE;
                   Salary:         float;
                   Supervisor      EMPLOYEE;
                   Dept:           DEPARTMENT;
        operations age:            integer;
                   createEmp:      EMPLOYEE
                   destroyEmp:     boolean );

define type DEPARTMENT
        struct (   Dname:          string;
                   Locations:      set(string);
                   Employees:      set(EMPLOYEE);
                   Mgr:            struct( Manager:  Employee
                                           StatDate: Date;);
        operations assignEmp (e: EMPLOYEE): boolean;
                   deleteEmp (e: EMPLOYEE): boolean;
                 );
```

dot notation:

d.destroyEmp;

Object oriented features
Type hierarchies and inheritance

- **Inheritance** allows the definition of new types based on other predefined types, leading to a **type (class) hierarchy**
- Consider an example where STUDENT and EMPLOYEE is **subtype** of PERSON class

```
PERSON: Name, Address, BirthDate, Age, Ssn

EMPLOYEE: Name, Address, BirthDate, Age, Ssn, Salary, HireDate, Seniority
EMPLOYEE subtype-of PERSON: Salary, HireDate, Seniority

STUDENT: Name, Address, BirthDate, Age, Ssn, Major, Gpa
STUDENT subtype-of PERSON: Major, GPA
```

Object oriented features
Polymorphism

- **Polymorphism** (operator overloading): allows the same operator name or symbol to be bound to two or more different implementations of the operator depending on the type of applied object
  - **early (static) binding** (strongly typed)
  - **late (dynamic) binding** (weak or no typing)

```
GEOMETRY_OBJECT: Shape, Area, ReferencePoint

RECTANGLE subtype-of GEOMETRY_OBJECT: Width, Height
TRIANGLE subtype-of GEOMETRY_OBJECT: Side1, Side2, Angle
CIRCLE subtype-of GEOMETRY_OBJECT: Radius
```

Object oriented features
Multiple and Selective Inheritance

- **Multiple inheritance**: when a subtype t is subtype of two or more supertypes
  - inherits the attributes and operations of both supertypes
  - e.g., ENGINEER_MANAGER subtype of both ENGINEER and MANAGER
  - What happens if both MANAGER and ENGINEER have an operation named salary?
- **Selective inheritance:** only some attributes or operations are inherited

Object oriented features
Programming language compatibility

- A database with object-oriented approach is often closely coupled with an OOPL
- Object persistence (*transient* vs. *persistent*)
  - *naming:* give a unique persistent name to object (entry points to the database)
  - *reachability:* object is reachable from some other persistent object (e.g., persistent object *N* whose state is a set of objects of some class *C*)

Objects-orientation and DBS
Different perspectives

- **Atkinson, 1989: The OO DBS Manifesto**
  - relational is old and inadequate
- **Cadf, 1990: Third Generation DBS Manifesto**
  - wrong! Add classes and inheritance, but keep declarative queries and SQL.
- **Darwen, 1995: The Third Manifesto**
  - wrong again! Extend relational but, dump SQL.

Objects-orientation and DBS
Different approaches

- Different approaches to ODBs (Object Databases) and ODBMSs (Object Database Management Systems)
  - **Object-oriented Databases** (OODBs) and Management Systems (OODBMS)
  - **Object Relational Databases** (ORDBs) and Management Systems (ORDBMS)
  - **Object-relational mappings**

Object-oriented databases
What is it?

- Follows the first manifestation
- No ties to relational systems
- Many experimental prototypes and commercial systems have been created
  - Experimental: Orion, OpenOODB, Iris, Ode, ENCORE etc.
  - Industrial: GemStone, ONTOS DB, FastObjects, Ardent etc.

Object-oriented databases
Strengths

- Tighter integration with OO languages
  - no concern on how an object is implemented, how it is stored, and retrieved
- Rich and extensible type system
- Better at modeling complex objects
  - ability to store and retrieve complex objects
- Better performance on certain data structures
- No impedance mismatch

Object-oriented databases
Weaknesses (re-learned a lot)

- Database design and application development tied too closely!
- Procedural navigation (declarative languages, e.g., SQL bring a tremendous productivity)
- Lack of a standard data model – no formal mathematical model -, leading to inconsistencies and design errors
- Querying breaks encapsulation!
- Did not find widespread use!

Object-oriented databases
Standardization

- OMG (Object Management Group): classical object-orientation mantra, no compromises for industry
- ODMG (Object Data Management Group) - a consortium of vendors and users
  - tried to address industrial concerns, otherwise close to OMG
  - proposed the ODMG (3.0 final) standard and finished activities on 2001, from 2004 managed by OMG (no activities)

Object-oriented databases
Other OMG Standards

- **CORBA** (Common Object Request Broker Architecture)
  - vendor-independent architecture and infrastructure that computer applications (object-oriented) use to work together over networks
  - an interface definition language **(IDL**) to specify the interfaces which objects present to the outer world

Object-oriented databases
Other OMG Standards

- **UML** (Unified Modeling Language)
  - is a standardized general-purpose modeling language in the field of object-oriented software engineering.
  - includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.
  - widely used in industry and education
  - http://www.omg.org/

Object-relational databases

- Follows the second manifestation
- Starts from a relational framework and extends the type system with objects
- **Object-relational model:** the relational model with object database enhancements
- SQL:99 (aka SQL3): features from object databases incorporated into SQL standard for the first time
  - SQL/Object to SQL/Foundation

Object-relational databases
SQL:2008

- SQL:2003 and SQL:2006 – features related to XML are added
- **SQL:2008** (support for XQuery regular expressions etc.)
- SQL:2011 (support for temporal databases)

Object-relational databases
SQL:2008

- Some object database features included in SQL
  - type constructors – includes row type (struct) and array type, and later set, list, bag
  - mechanism for specifying object identity (through *reference type*)
  - encapsulation through *user defined types* (UDTs)
  - inheritance mechanism using keyword UNDER

Object-relational databases
SQL:2008 – User Defined Data Types

- Allows creation of complex-structured objects

```
CREATE TYPE STREET_ADDR_TYPE AS (
        NUMBER          VARCHAR(5)
        STREET_NAME     VARCHAR(25)
        APT_NO          VARCHAR(5)
        SUITE_NO        VARCHAR(5) );

CREATE TYPE USA_ADDR_TYPE AS (
        STREET_ADDR     STREET_ADDR_TYPE,
        CITY            VARCHAR(25),
        ZIP             VARCHAR(10) );

CREATE TYPE USA_PHONE_TYPE AS (
        PHONE_TYPE      VARCHAR(5)
        AREAD_CODE      CHAR(3)
        PHONE_NUM       CHAR(7) );
```

Object-relational databases
SQL:2008 – User Defined Data Types

- Keyword **ROW** to directly create structured attribute (UDT should not have any operations)

```
CREATE TYPE USA_ADDR_TYPE AS (
        STREET_ADDR     STREET_ADDR_TYPE,
        CITY            VARCHAR(25),
        ZIP             VARCHAR(10) );

CREATE TYPE USA_ADDR_TYPE AS (
        STREET_ADDR     ROW ( NUMBER            VARCHAR(5)
                              STREET_NAME       VARCHAR(25)
                              APT_NO            VARCHAR(5)
                              SUITE_NO          VARCHAR(5) ),
        CITY            VARCHAR(25),
        ZIP             VARCHAR(10) );
```

Object-relational databases
SQL:2008 – UDTs and Collections

- SQL: *ARRAY*, *MULTISET*, *LIST*, and *SET*
- First ARRAY type was introduced, since it could be used to simulate the other types

```
CREATE TYPE PERSON_TYPE AS (
        NAME            VARCHAR(25),
        SEX             CHAR,
        BIRTH_DATE      DATE,
        PHONES          USA_PHONE_TYPE ARRAY[4],
        ADDR            USA_ADDR_TYPE );
```

<u>Access mechanism</u>

```
PHONES[1]
PHONES[CARDINALITY(PHONES)]
ADDR.CITY
```

Object-relational databases
OIDs and tables based on UDTs

- OIDs: *SYSTEM GENERATED* OR *DERIVED*
- Tables based on UDTs: *INSTANTIABLE*

```
CREATE TYPE PERSON_TYPE AS (
        NAME            VARCHAR(25),
        SEX             CHAR,
        BIRTH_DATE      DATE,
        PHONES          USA_PHONE_TYPE_ARRAY[4],
        ADDR            USA_ADDR_TYPE
INSTANTIABLE
REF IS SYSTEM GENERATED );

CREATE TABLE PERSON OF PERSON_TYPE
        REF IS PERSON_ID SYSTEM_GENERATED;
```

Object-relational databases
Encapsulation and operations

- UDT behavioral specification with methods
- Built-in functions for UDTs
  - **constructor function** (returns a new object)
  - **observer function** (for reading attributes)
  - **mutator function** (for updating attributes)
- Attributes and functions in UDT
  - **PUBLIC** (visible at the UDT interface)
  - **PRIVATE** (not visible at the UDT interface)
  - **PROTECTED** (visible only to subtypes)

Object-relational databases
Encapsulation and operations

```
CREATE TYPE PERSON_TYPE AS (
        NAME            VARCHAR(25),
        SEX             CHAR,
        BIRTH_DATE      DATE,
        PHONES          USA_PHONE_TYPE_ARRAY[4],
        ADDR            USA_ADDR_TYPE
INSTANTIABLE
REF IS SYSTEM GENERATED
INSTANCE METHOD AGE() RETURNS INTEGER;
        FOR PERSON_TYPE
        BEGIN
           RETURN /* CODE TO CALCULATE A PERSON's AGE */
        END;
 );
```

Object-relational databases
Inheritance and function overloading

- Type inheritance: specified with **UNDER** keyword
- The phase **NOT FINAL** must be included in UDT if subtypes are allowed under that UDT
- All attributes and operations are inherited
- UNDER clause determines inheritance hierarchy
- A subtype can refine any function defined in its supertype (signatures must be the same)
- Dynamic linking considers runtime types

Object-relational databases

Inheritance and function overloading

```
CREATE TYPE PERSON_TYPE AS (
        NAME            VARCHAR(35),
                ***
        PHONES          USA_PHONE_TYPE_ARRAY[4],
        ADDR            USA_ADDR_TYPE
INSTANTIABLE
NOT FINAL
REF IS SYSTEM GENERATED
INSTANCE METHOD AGE() RETURNS INTEGER;
        FOR PERSON_TYPE
        BEGIN
           RETURN /* CODE TO CALCULATE A PERSON's AGE */
        END;
 );

CREATE TYPE MANAGER_TYPE UNDER EMPLOYEE_TYPE AS (
        DEPT_MANAGED    CHAR(20)
INSTANTIABLE );
```

Object-relational databases
Inheritance and function overloading

- Table inheritance

```
CREATE TABLE PERSON OF PERSON_TYPE
        REF IS PERSON_ID SYSTEM GENERATED

CREATE TABLE EMPLOYEE OF EMPLOYEE_TYPE
        UNDER PERSON;

CREATE TABLE MANAGER OF MANAGER_TYPE
        UNDER EMPLOYEE;

CREATE TABLE STUDENT OF STUDENT_TYPE
        UNDER PERSON;
```

Object-relational databases
Specifying relationships

- Specified by using keywords **REF** and **SCOPE**

```
CREATE TYPE COMPANY_TYPE AS (
        COMP_NAME        VACHAR(20),
        LOCATION         VARCHAR(20) );

CREATE TYPE EMPLOYMENT_TYPE AS (
        Employee REF (EMPLOYEE_TYPE) SCOPE (EMPLOYEE),
        Company  REF (COMPANY_TYPE)  SCOPE (COMPANY) );

CREATE TABLE COMPANY OF COMPANY_TYPE (
        REF IS COMP_ID SYSTEM GENERATED,
        PRIMARY KEY (COMP_NAME) );

CREATE TABLE EMPLOYEMENT OF EMPLOYEMENT_TYPE;
```

Object-relational databases
Specifying relationships

- For building path expressions
  - dot notation (x.y) – to access attributes and row types
  - dereferencing symbol (x->y) – to access REF type attributes

```
SELECT   E.Employee->NAME
FROM     EMPLOYEMENT AS E
WHERE    E.Company->COMP_NAME = “ABC”;
```

Object-relational mappings

Object-oriented databases

Object-relational databases

Object-relational mappings

Object-relational mappings

The approach is based on declarative mappings between the schema of the underlying database schema and domain classes, which separates transactional and application-oriented concerns.

Object-relational mappings *Hibernate*

JAVA CLASS

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public int getId() {
      return id; }
   public void setId( int id ) {
      this.id = id; }
  public String getFirstName() {
      return firstName; }
  * * *
}
```

DATABASE TABLE

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

Object-relational mappings *Hibernate*

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="Employee" table="EMPLOYEE">
    <meta attribute="class-description">
      This class contains the employee detail.
    </meta>
    <id name="id" type="int" column="id">
      <generator class="native"/>
    </id>
    <property name="firstName" column="first_name" type="string"/>
    <property name="lastName" column="last_name" type="string"/>
    <property name="salary" column="salary" type="int"/>
  </class>
</hibernate-mapping>
```

Object-relational mappings *Hibernate*

SAVE EMPLOYEE

```
tx = session.beginTransaction();
Employee employee = new Employee(fname, lname, salary);
employeeID = (Integer) session.save(employee);
tx.commit();
```

DELETE EMPLOYEE

```
tx = session.beginTransaction();
Employee employee = (Employee) session.get(Employee.class, EmployeeID);
session.delete(employee);
tx.commit();
```

Object-relational mappings *iBatis*

JAVA CLASS

```
public class Person {
  private int id;
  private String firstName;
  private String lastName;
  private Date birthDate;
  private double weightInKilograms;
  private double heightInMeters;

  public int getId () {
     return id; }
  public void setId (int id) {
     this.id = id; }
}
```

DATABASE TABLE

```
CREATE TABLE PERSON(
    PER_ID NUMBER (5, 0) NOT NULL,
    PER_FIRST_NAME VARCHAR (40) NOT NULL,
    PER_LAST_NAME VARCHAR (40) NOT NULL,
    PER_BIRTH_DATE DATETIME ,
    PER_WEIGHT_KG NUMBER (4, 2) NOT NULL,
    PER_HEIGHT_M NUMBER (4, 2) NOT NULL,
    PRIMARY KEY (PER_ID)
)
```

Object-relational mappings *iBatis*

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap namespace="Person">
        <select id="getPerson" resultClass="examples.domain.Person">
                SELECT
                        PER_ID as id,
                        PER_FIRST_NAME as firstName,
                        PER_LAST_NAME as lastName,
                        PER_BIRTH_DATE as birthDate,
                        PER_WEIGHT_KG as weightInKilograms,
                        PER_HEIGHT_M as heightInMeters
                        FROM PERSON
                        WHERE PER_ID = #value#
        </select>
</sqlMap>
```

The next step
Current research

- Ontology-based data access (OBDA)
- Ontologies provide a higher level of abstraction
- Ontologies are more expressive and intuitive
- Ontologies provide reasoning support
- Existing approaches are similar to object-relational mappings based on Web Ontology language (OWL)
- Optique project: Scalable end-user access to big data: http://www.optique-project.eu