COMP207
Database Development

Lecture 26

Beyond Relational Data:
Object-Relational Databases

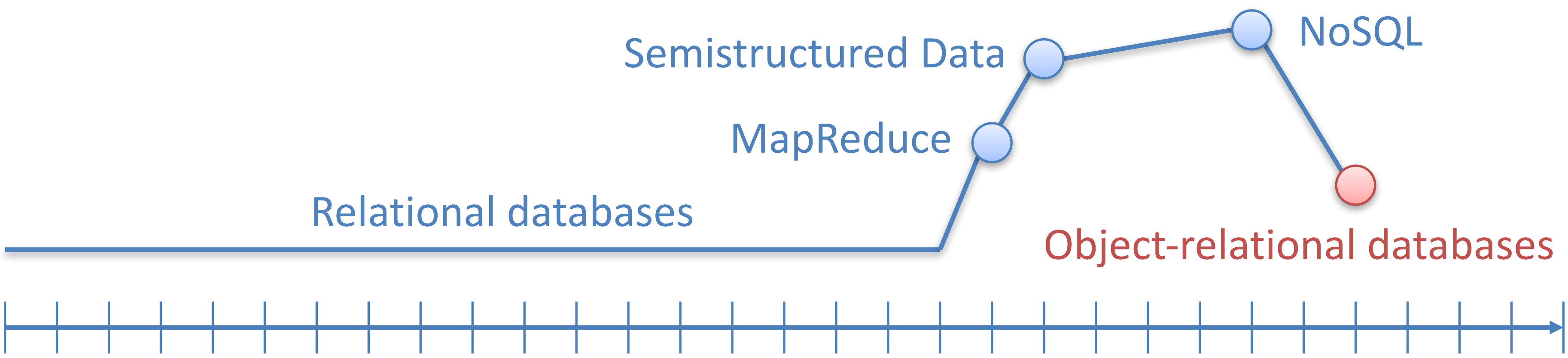
- End of Module Questionnaires
  - Should have received the link by email
  - If you prefer, bring your laptop/mobile device and complete the questionnaire in the end of the first lecture next Tuesday

No lecture tomorrow!

- There is no lecture tomorrow, since I have to go to a workshop
  - Hence, the announcement saying so
  - Also, last lecture (the revision lecture) is Monday the 9^{th} of December

What you should know at the end
(Learning Outcomes)

- Transaction management:
  - Identification & application of the principles underpinning transaction management within DBMS
- Advanced SQL:
  - SQL from COMP102 extended with indexes, transactions, query optimisation
  - Application in problem solving
- Object-relational models:
  - Identification of principles
- Web technologies:
  - Illustrate issues related to web technologies as a semi-structured data representation formalism
- Data warehouses and data mining:
  - Interpret the main concepts and security aspects in data warehousing
  - Interpret the main concepts of data mining


Semistructured Data
NoSQL
MapReduce
Relational databases
Object-relational databases

Object-Oriented Programming

```
class Person {
    private String name;
    public Person(String name) { … }
    public String getName() { … }
    …
}
```

```
class Student extends Person {
    private Programme prg;
    public Student(String name, Programme prg) { … }
    public Programme getProgramme() { … }
}
```

- Objects:
  - Identity, state & behaviour
  - Have a type (class)
- Key concepts:
  - Encapsulation
  - Inheritance
  - Polymorphism
- Same identity ≠ same state

Relational Databases

Person

id	name
…	…

Student

pid	programme
…	…

- Relations
  - Collections of tuples with the same attributes
- Tuples:
  - Just the state (values)
  - No behaviour
- Values are extracted or updated by high-level queries rather than by calling methods on tuples
- Same identity = same state!

Impedance Mismatch

- Mismatch between the two worlds leads to problems
  - Objects vs tuples in relations
  - Procedural vs declarative
  - Data types
- Referred to as **impedance mismatch** (http://blogs.tedneward.com/post/the-vietnam-of-computer-science/)
- Seamless integration of object-oriented applications with relational databases requires significant resources

Object-Oriented Databases

Object-Oriented DBMS

- Extends object-oriented programming technology

```
class Person {
   private String name;
   public Person(String name) { … }
   public String getName() { … }
   …
}
```

```
class Student extends Person {
   private Programme prg;
   public Student(String name, Programme prg) { … }
   public Programme getProgramme() { … }
}
```

- Idea: add **persistence** of objects

  Think of a table per class

  – Database takes care of storing objects in memory/on disk, concurrency control, transactions, etc.
  – Sharing of objects between applications possible

Early Data Management

(from lecture 1)

```
class Student {
    String name;
    int number;
    String programme;
    …
}

Vector<Student> students;
…
```

```
Anna, 20181989, G402
John, 20184378, G702
…
```

External data file

- Early DBMS of the 1960’s were based on this idea
- Disadvantages:
  - difficult to program
  - not very robust, especially when dealing with updates to data by many users in parallel
  - Hard to add fields or new efficient queries

Object-Oriented DBMS

- Surge of academic & commercial interest in early 90s
  - Thought to gain ~ 50% market share
- Today:
  - Relatively small market, say 5%
  - Used in areas such Computer Aided Design
- Main problems:
  - Application design tied too closely to database design
  - No declarative query language
  - Early on: No standards → design errors & inconsistencies
- Many of the ideas found in object-relational DBMS

Application design tied to database design

- Common to *refractor* code
  - Make small improvements without changing the behaviour
- Hard to do with object oriented databases
  - Must change all objects
  - Must check that all changes are made correctly!

No declarative query language

- (SQL queries state what you want, not how to do it)
  - Lots of optimizations
- Exists some declarative-like transformations in general oo programming language compilers
  - Much harder to do automatically because of more options and side effects

No early standards

- Exists standard from ‘93 called ODMG
  - Object Data Management Group
- Mostly looks like Java or C++
- Differences:
  - each object has an unique Object_id
  - objects can be given (persistent) names
  - two types of inheritance: (extends and ISA)
  - forced to use factory objects

Object-Relational Databases

Object-Relational DBMS

- Attempt to get the best of:
  - Object-oriented models with complex data types
  - Relational model with high-level query language
- Object-oriented features in SQL'99 and above
  - Possibility to define classes
  - Inheritance and polymorphism
  - Other features
- Still: SQL dialects of various commercial DBMS differ in object-relational features

User-Defined Types (UDTs)

- A class declaration: attributes + methods
- Main use:
  - As **attribute type**
  - As a **row type**: type of a table
- Definition of UDTs without methods:
  - `CREATE TYPE` *T* `AS <primitive type>;`

```
CREATE TYPE StudentID AS Int;
```

  - `CREATE TYPE` *T* `AS (<attribute declarations>);`

```
CREATE TYPE Address AS (
        street CHAR(50),
        city   CHAR(20)
);
```

UDTs As Attribute Types

- UDTs can be used as attribute types
- Example:

```
CREATE TYPE Address AS (
       street CHAR(50),
       city   CHAR(20)
);

CREATE TABLE Students AS (
       name          CHAR(50),
       address       Address
);
```

UDTs As Row Types

- UDTs can also be used as row types (types of tables)
- Syntax: `CREATE TABLE <table name> AS <type name>`
- Example:

```
CREATE TABLE Students AS Student;

CREATE TYPE Student AS (
      first_name  CHAR(20),
      last_name   CHAR(20),
      student_id  INT
);
```

Students

obj1 = Student(‘Anna’, ‘A.’, 123)
obj2 = Student(‘Ben’, ‘B.’, 456)
…

- Creates a relation with one column whose tuples are objects of type Student

Queries

- To query an object-relational database, use normal SQL syntax
- Example:

```
CREATE TYPE Student AS (
        first_name    CHAR(20),
        last_name     CHAR(20),
        student_id    INT
);
```

```
CREATE TABLE Students AS Student;
```

Students

Student('Anna', 'A.', 123)
Student('Ben', 'B.', 456)
…

```
SELECT * FROM Student S
```

Returns: Student('Anna', 'A', 123), Student('Ben', 'B.', 456)

Observer Methods

- How do we access the values (first_name, …) of a Student object in table Students?

```
CREATE TYPE Student AS (
        first_name    CHAR(20),
        last_name     CHAR(20),
        student_id    INT
);

CREATE TABLE Students AS Student;
```

Students

Student('Anna', 'A.', 123)
Student('Ben', 'B.', 456)
…

- Answer: use observer methods first_name(), …

```
SELECT S.first_name() FROM Students S;
```

Returns: 'Anna', 'Ben'

Adding Methods to UDTs

- **Declaration** via `METHOD <name>() RETURNS <type>;`
  - Part of UDT

```
CREATE TYPE Address AS (
        street    CHAR(50),
        city      CHAR(20)
)
METHOD houseNumber() RETURNS Char(10);
```

- **Definitions** are separate from the UDT:

```
CREATE METHOD houseNumber() RETURNS CHAR(10)
FOR Address
BEGIN
        …
END
```

Inheritance

- Inheritance is implemented via table inheritance
  - One table can inherit the attributed of another table
  - Keyword: UNDER …
- Example:

```
CREATE TABLE Cities (
        name            Text,
        population      Float,
        altitude        Int
);

CREATE TABLE Capitals UNDER Cities (
        state           Char(2)
);
```

Different in PostgreSQL

References

- References are essential for object-oriented models
- To say that **attribute A** is a reference to objects of **type T** in **relation R**, use:

```
A REF(T) SCOPE R
```

- Example:

```
CREATE TABLE Student (
      name          CHAR(50),
      programme     REF(Programme) SCOPE Programmes
);
```

Generating references

- Generate reference

```
REF IS progammeId SYSTEM GENERATED
```

Can also be DERIVED in which case it comes from primary key

- Example:

```
CREATE TABLE Programme (
      REF IS progammeId SYSTEM GENERATED,
      name          CHAR(50)
);
```

How to follow references

- if x is a reference to a tuple t with attribute a then x->a returns the attribute and DEREF(x) gives t

- Example:
- Schema enrolledIn(student,module)
  - Student and module is a reference to a student and a module respectively
- SELECT DEREF(module)
FROM enrolledIn
WHERE student->name='Anna';

Other Features

- Various other features that have not been covered
- Examples:
  - Generator and mutator methods
  - Ordering
  - Implementation of methods in programming languages like C, …
  - …
- See SQL standard, or better: documentation of PostgreSQL, Oracle, …

Summary

- Object-oriented DBMS provide much flexibility and solve the impedance mismatch problem for object-oriented programming
- Disadvantages, but used in some areas
- Object-relational DBMS try to combine the best of object-oriented DBMS and relational DBMS
  – Relational model
  – On top: object-oriented concepts such as classes (UDTs), objects, inheritance, etc.