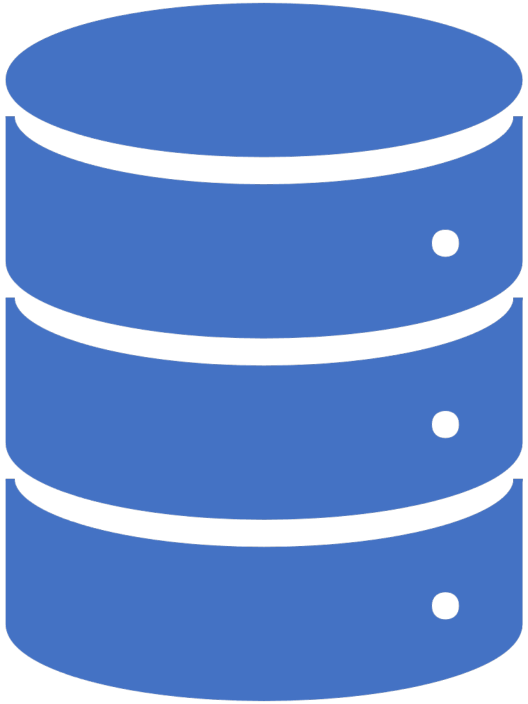


Lecture 11

Object Oriented Databases

Object Oriented Databases

What can be stored?

- Multimedia information (images, movies)
- Space Data (GIS)
- Biological Data
- Technological projects (CAD data)
- Virtual worlds
- Games
- Data streams
- Data types defined by the user

Handling the new data

- a TV channel needs to store and access quick the videos, radios interviews, multimedia documents, …
- a movie director needs to store movies, data about actors, cinemas, …
- a biological lab needs to store complex data related to the molecules, chronozones, …

Object Oriented Databases

Needs for DBMSs

- increase the quantity of the accessed data in the applications and also reduce the time need to develop these applications
  - Object oriented programming
  - DBMSs options: query optimisation, data recovery, concurrent access control, indexing, …

Disadvantages of the relational databases

- no collection attributes
- no inheritance
- no complex objects (except, BLOG – binary large objects)
- the conceptual difference between the data access (declarative: SQL) and the programming language (procedural: C++, C#, Java, …)

Solutions?

Object Oriented Databases

Object Databases = storage of persistent objects

- Object-Oriented Database Systems – alternative to the relational systems
- Object-Relational Database Systems – extension of the the relational systems
- Object-Oriented Database Systems: ObjectStore, GemStone, Wakanda, Realm

Object Data Model

= represents the fundamental of the Object –Oriented Databases

Relational Model

= represents the fundamental of the Relational Databases

- Database - contains an object collection
- Each object has an unique ID (OID) and the collection of the objects that have similar properties is called *class*
- The properties of an object are specified by using an ODL (Object Description Language) and the objects are accessed by using OML (Object Manipulation Language)

Object Oriented Databases

Object Properties

- ***Attributes*** = have atomic or structural types (*set, bag, list, array*)
- ***Relationships*** = reference to an object or set of objects
- ***Methods*** = functions that can be applied to the objects of a class

Abstract Data Types

- key functionality : create new data types by users
- a new data type has methods to access (type + methods = abstract data type)
- a DBMS has predefined types

Encapsulation

- Encapsulation = data structures + operations
- Encapsulation allows to hide the internal details of an abstract data type
- DBMS does not need to know the way of storing the data or the works way of an abstract data type; it is need to be known the available methods and their call details (I/O types)

Object Oriented Databases

Inheritance

- ***a value has a type***
- ***an object is part of a class***
- hierarchy types – can be defined new types based on the existing ones; a subtype inherit all the properties of the supertype
- hierarchy class – a subclass C’ of a class C is a collection of objects in which each object of the class C’ is in the same time also object of the class C; an object of the class C’ inherits all the properties from C

Object Oriented Database

- the aim of an DBMS OO is to be integrated in an OOP language (e.g. C++, C#, Java, …)
- ODL = Object Description Language (like, DDL in SQL)
- OML = Object Manipulation Language (like, DML in SQL)

ODL in OO Databases

- ODL is used to define persistent classes with their objects permanent stored in the database
- defining classes with ODL is an extension of the host Object – Oriented language

Object Oriented Databases

ODL in OO Databases

- Class declaration include
  - class name
  - optional key declaration
  - extent declaration = the name of the set of all the objects that are part of the class
  - elements declarations (an element can be an attribute, a relationship or a method)

class <name> {

<list of element declarations, separated by semicolons>}

Attributes and Methods declarations

- usually, the attributes are declared through name and type (type does not represent a class)

attribute <type> <name>;

- the information from a method declaration contains
  - the returned type (if necessarily)
  - method name
  - (in, out, inout) pair and the argument type (without name)
  - the exceptions can be throw by the method

***float* grade_average(in string) raises (noGrade);**

Object Oriented Databases

Relationships declaration

- the relationships connect an object of a class with one or multiple objects of another class
- the relationships are saved as pairs of pointers switched (A reference B and B reference A)
- the relationships are maintained automatically by the system (if A is eliminated, the B pointer will be automatically initialized with NULL)
- Relationships types: one-to-one, one-to-many, many-to-many

relationship <type> <name> inverse <relationship>;

Object Oriented Databases

Relationships declaration - Example:

```
class Movie{
        attribute date start;
        attribute date end;
        attribute string moviename;
        relationship Set<Cinema> shownAt inverse
                                Cinema::nowShowing;
}

class Cinema{
        attribute string cinemaName;
        attribute string address;
        attribute integer ticketPrice;
        relationship Set<Movie> nowShowing inverse
                                Movie::shownAt;
        float numshowing() raises (errorCountingMovies);
}
```

Relationship type

Operator :: connect a name to a context

Object Oriented Databases

Relationships types

- the type of a relationship can be
  - a class (e.g. Movie) – an object with this type of relationship can be connected with a single object *Movie*
  - Set<*Movie*>*:* the object is connected to a set of objects *Movie*
  - Bag<*Movie*>*,* List<*Movie*>*,* Array<*Movie*>*:* the object is connected to a set of duplicates, list or array of objects *Movie*.

The Multiplicity of the relationships

- All ODL relationships are binary
- Many-to-many relationships have *Collection* as type of the relationship and are reverted
- One-to-many relationships have *Collection*<…> in the relationship declaration in the "one" object and just one class in the relationship declaration "many" object
- One-to-one relationships have class relationship in both directions

Object Oriented Databases

The Multiplicity of the relationships – example

```
class Confectioner { …
        relationship Set<Cookie> likes inverse
                                    Cookie::fans;
        relationship Cookie favCookie inverse
                                    Cookie::superfans;
}
```

many-to-many

```
class Cookie { …
        relationship Set<Confectioner> fans inverse
                                    Confectioner::likes;
        relationship Set<Confectioner> superfans inverse
                                    Confectioner::favCookie;
}
```

one-to-many

Object Oriented Databases

The Multiplicity of the relationships – example

```
class Student {
        attribute …;

        relationship Student boy inverse girl;
        relationship Student girl inverse boy;

        relationship Set<Student> buddies
                                inverse buddies;

}
```

boy and girl are one-to-one relationships and represent the opposite of the other

buddies is many-to-many relationship and represent its own opposite

Object Oriented Databases

Connecting classes

- connect class X, Y and Z through a relation R
  - a class C is created – the objects have the form (x, y, z) and correspond to classes X, Y, Z
  - 3 relationships many-to-many will be created – from (x, y, z) to each of x, y and z

Example:

- Let the classes *BookStore* and *Book*. Memorize the price with which every library (object of the *BookStore*) sells a book.
  - a relation many-to-many between BookStore and Book is not good because there cannot be defined attributes connected to this relation
  - Solution 1: a class *Price* and a connected class *BBP* can be created to represent the relationship between the library, book and price.
  - Solution 2: because the objects *Price* contain just a simple number, could be useful to add an attribute *price* to the class BBP and to use many-to-many relationships between an object *BBP* and objects of the *BookStore* and *Book*.

Object Oriented Databases

Example:

- define the class BBP

```
class BBP {
        attribute real price;
        relationship BookStore theBS inverse
                        BookStore::toBBP;
        relationship Book theBook inverse
                        Book::toBBP;
}
```

- *BookStore* and *Book* will be both modified by excluding the relationship called *toBBP*, of type Set<BBP>

Object Oriented Databases

ODL Types

- the base types: *int, real/float, string, enumeration types, classes*
- composed types:
  - *struct* for structures
  - collection types: *Set, Bag, List, Array, Dictinary*
- the relationship types can be only classes or a collection type applied to a class

ODL subclasses

- correspond to the known subclasses from the OOP

```
class Student:Person
{
        attribute string code;
        …
}
```

Object Oriented Databases

Keys and extensions in ODL

- for a class can be declared as many key as desired

  { key <list of keys>}

- each class has an *extent* – that represent the set of all objects of the corresponded class
- an extension is declared after the name of the class together with the keys

  { extent <extent name> … }

  - convention: common noun at singular for class name and at plural the corresponding extensions

Example:

```
class Book
        ( key name) { … }

class Course
        ( key dept, number),
        (room, hours)) { … }

class Student
        ( extent Students key code) { … }
```

Object Oriented Databases

OML in OO DBMS

- the OML implements are not very efficient (the optimizations of query language are poor)
- the most used query language is OQL (Object Query Language) and has a similar structure with SQL
- OQL is an extension of the SQL
  - include the clauses select, from, where and group by
  - have been added new elements to access the objects properties and operators for the complex data

Object Oriented Databases

OML in OO DBMS

Example:
```
class Movie (extent Movies key movieName) {
        attribute date start;
        attribute date end;
        attribute string movieName;
        relationship Set<Cinema> shownAt inverse
                            Cinema::nowShowing;
    }
    class Cinema (extent Cinemas key cinemaName) {
        attribute string cinemaName;
        attribute string address;
        attribute integer ticketPrice;
        relationship Set<Movie> nowShowing inverse
                            Movie::shownAt;
        float numshowing() raises (erroCountingMovies)
    }
```

Object Oriented Databases

Accessing the properties of the objects (path expresssions)

- Let x be an object of the class C
  - if a is an attribute of C then x.a is the value of that attribute
  - if r is a relationship of C then x.r is the object or the collection of objects with which x is connected through r
  - if m is a C method then x.m (…) is the result of applying m to x

OQL: Select-From-Where (OQL – Object Query Language)

- A phrase OQL has the syntax

SELECT <list of values>
FROM <list of collections and names for typical members>
WHERE <condition>

- Each term of the FROM clause is

<collection> <member name>

- A collection can be an extension of a class or an expression that is evaluated to a collection
- To change the name of a field, this one must be preceded by ":"

Object Oriented Databases

OQL Example

Return the cinemas in which is played more than a movie and these movies.

```
SELECT mname: M.movieName, cname: C.cinemaName
FROM Movie M, M.shownAt C
WHERE C.numshowing() >1
```

The type of a query result

- default – the type of a structure select-from-where is a *Bag* of *Struct*
  - *Struct* has a field for each term of the SELECT clause. The name and the type are provided by the last element of the path expression
- if the query result has just one term, this is going to be a structure with one field
- DISTINCT can be added after SELECT and the result can have the type *Set*, where the duplicates are eliminated
- On the ORDER BY clause the result will be a list of structures ordered by the fields from ORDER BY
  - the order is ascending (ASC – default) or descending (DESC)
  - the elements of the list can be accessed by using the indexes ([1], [2], …) – similar as for the SQL cursors

Object Oriented Databases

Subqueries / Nested queries

- an expression select-from-where can be used as a nested query in multiple ways
  - in a FROM clause, as a collection
  - in a logical expression used in the WHERE clause
    - FOR ALL x IN <collection> : <condition>
    - EXISTS x IN <collection> : <condition>

Example

Return the name of the movies that are played in at least one cinema where the ticket price >5

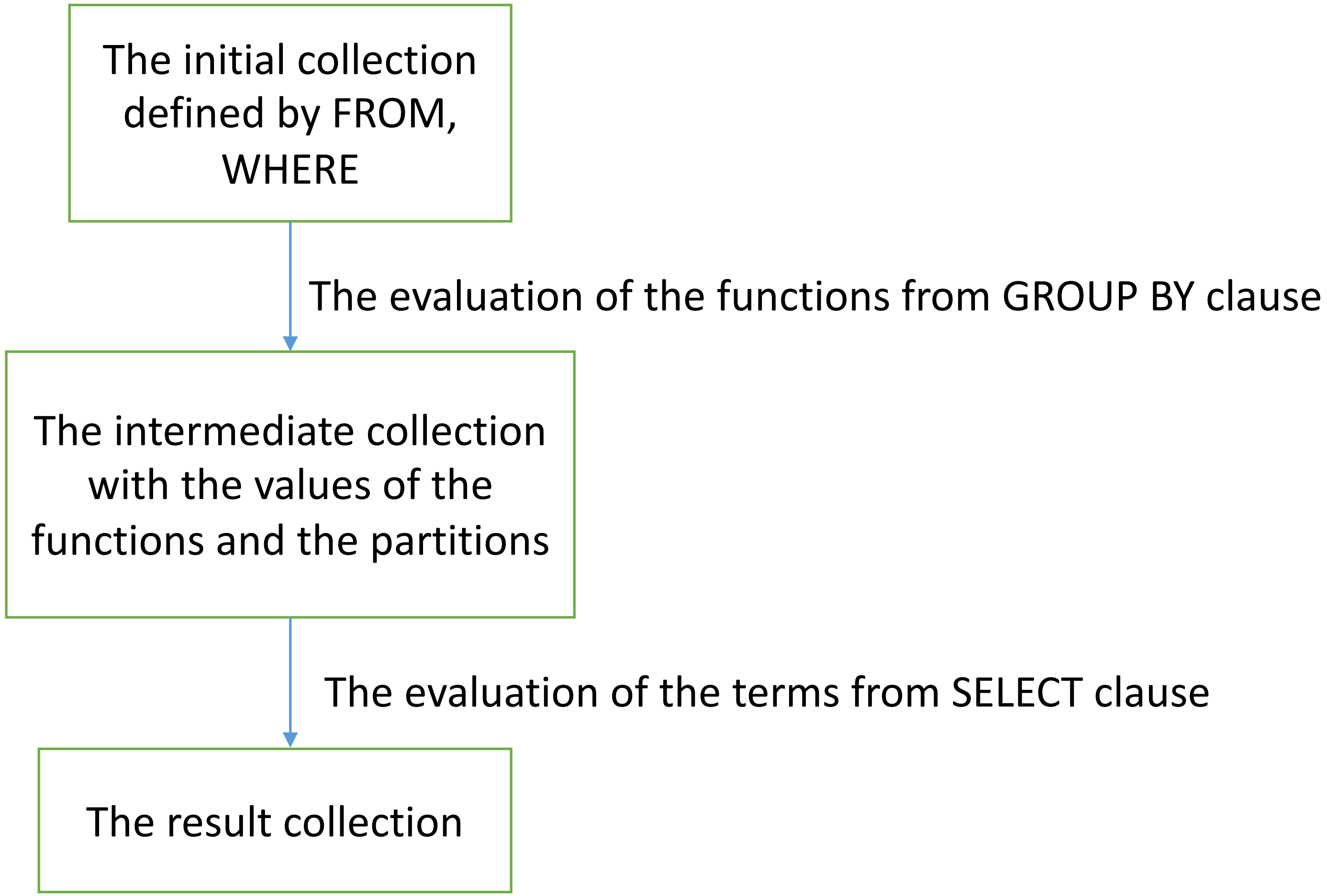
```
SELECT m.name
FROM Movie m
WHERE EXISTS c IN m.shownAt: c.ticketPrice >5
```

Group the data in OQL

- OQL extend the group
  - all the collections can be split into groups
  - the groups can have a base of any function of the objects from the initial collection
- AVG, SUM, MIN, MAX, COUNT – can be applied to all the collections (when need it)

Object Oriented Databases

Group the data in OQL



Object Oriented Databases

GROUP BY example

Return all the distinct price values used in cinemas and the average number of the movies played with the sold tickets on the corresponding price.

```
SELECT C.ticketPrice, avgNum:AVG(SELECT P.C.numshowing()
                                 FROM partition P)
FROM Cinema C
GROUP BY C.ticketPrice
```

Partion in OQL

<u>Example: initial collection</u>

- Based on FROM and WHERE (that missing): FROM Cinemas C
- the initial collection is a *Bag* of structures with only one field for each element from the FROM clause
- in particular, the collection represent a *Bag* of structures with the form Struct(c:obj), where obj is a Cinema object

<u>Example: intermediate collection</u>

- in general, it is a *bag* of structures with a component for each function from the GROUP BY clause and a supplementary component called *partition*
- the value of the component *partition* is given by the se of all objects from the initial collection that are part of the group represented by the structure

Object Oriented Databases

GROUP BY example

<u>Example: intermediate collection</u>

```
SELECT C.ticketPrice,
        avgNum:AVG(
        SELECT P.C.numshowing()
        FROM partition P)
FROM Cinema C
GROUP BY C.ticketPrice
```

A group function contains:
- Name – *ticketPrice*
- Type - *integer*

The intermediate collection is a *set* of structures with the fields
- *ticketPrice*: string
- *partition:* Set<Struct{c: Cinema}>

- an element of the intermediate collection is
  Struct(*ticketPrice=5,*
  partition={c_1*,* c_2*, …,* c_n*})*
- each element of the partition is an object c_i of the class *Cinema*, for each c_i*.ticketPrice*=5

<u>Example: final collection</u>

- the result collection is given by the SELECT clause which is evaluated on the intermediate collection

Object Oriented Databases

GROUP BY example

Example: final collection

```
SELECT C.ticketPrice, avgNum:AVG(
        SELECT P.C.numshowing() FROM partition P)
```

Extract the field *ticketPrice* from the structure of a group

For each P element from *partition*, it is accessed the attribute C (object of *Cinema*), from where access the number of projections

The average of the numbers returned by the functions *numshowing()* stored in the field *avgNume* of the structures from the final collection

Element example:
Struct(ticketPrice = 5, avgNum=9.5)

Object Oriented Databases

DBMS evolution

- Object Oriented DBMSs have failed because they couldn't offer the efficiency obtained by the relational DBMS.
- the object-relational extensions applied to the relational DBMSs keeps a good part of the advantages of OO, but the fundamental abstraction remains the relationship

DBMSs classification

	Simple Data	Complex Data
Queries	Relational DBMSs	Relational-Object DBMSs
Without Queries	File System	DBMSs Object-Oriented

References:

- C.J. Date, *An Introduction to Database Systems (8th Edition)*, Addison-Wesley, 2003.
- H. Garcia-Molina, J. Ullman, J. Widom, *Database Systems: The Complete Book, Prentice Hall Press*, 2008.
- G. Hansen, J. Hansen, *Database Management And Design (2nd Edition),* Prentice Hall, 1996.
- R. Ramakrishnan, J. Gehrke, *Database Management Systems*, McGraw- Hill, 2007. http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- R. Ramakrishnan, J. Gehrke, *Database Management Systems (2nd Edition),* McGraw-Hill, 2000.
- A. Silberschatz, H. Korth, S. Sudarshan, *Database System Concepts*, McGraw-Hill, 2010. http://codex.cs.yale.edu/avi/db-book/
- L. Țâmbulea, *Curs Baze de date*, Facultatea de Matematică și Informatică, UBB, 2013-2014.
- J. Ullman, J. Widom, *A First Course in Database Systems*, http://infolab.stanford.edu/~ullman/fcdb.html