Object Relational Mapping for Database Integration

Erico Neves, Ms.C.
(enevesita@yahoo.com.br)
State University of Amazonas – UEA

Laurindo Campos, Ph.D.
(lcampos@inpa.gov.br)
National Institute of Research of Amazonia – INPA

Brazil

Presentation Scope

- Introduction
- Objectives
- Previous works
- ORM Models
- Simulations Results
- Conclusions

Introduction

- Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of this data;
- An old problem, but still not solved;
- Integration is divided into two main approaches:
  - Schema integration reconciles schema elements;
  - Instance integration matches tuples and attribute values.

Introduction

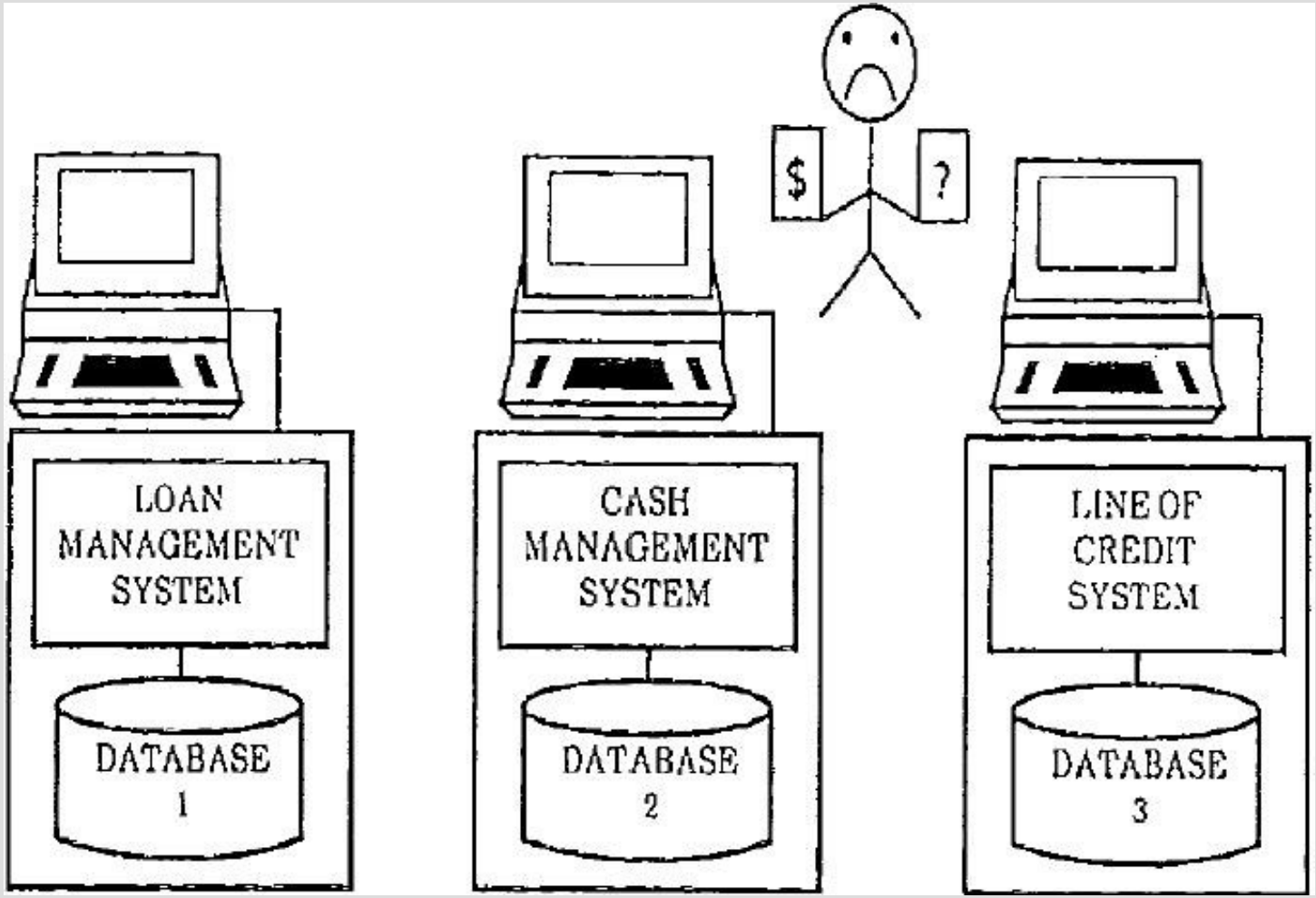
- Developers have many difficulties to integrate data from different sources in their applications:
  - Usually, the computer languages offer a basic support to execute queries;
  - Most part of work is left to developer hands;
  - Relational databases must be “translated” to Object Oriented applications;
  - Data type conversion;
  - Possible solution with XML.
- One approach is use database views are used to create integration.

Objectives

- Present an API to create data integration on Instance level;
  - The API will be used in application program;
  - Two requisites to execute this are:
    - **No temporary views from databases will be used;**
    - **Databases will be used just to retrieve and store data ⇨ no requisites to know other databases;**
  - This API will also offer an approach to allow Object materialization.

Previous Works

- In 1986, Frank, Madnick and Wang proposed the following problem:



Previous Works

- There are some implementations of Object to Relational Mapping (ORM) for Java:

- All base their Schema Mappings using XML;
  - Developer creates a XML file that informs how the application will “see” the tables in database;
  - This XML file also informs the relations between the table's attributes and the application's object attributes.

- Examples:
  - Hibernate;
  - Java Data Objects (JDO); etc...

Previous Works

- Example Hibernate:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="someclass" table="tableindatabase">
    <id name="classattribute" column="columnNameInDatabase"/>
    <property name="date" type="timestamp"
column="EVENT_DATE"/> (OPTIONAL)
  </class>
</hibernate-mapping>
```

Previous Works

- Disadvantages:
  - All application classes, that reflect informations from database's tables, must have a information reflecting in this XML;
  - Any changes in the database, implies in change in application XML file;
  - Developer may use different names to represent attributes in databases and objects;
  - Main focus is Object to Relational Mapping, not data integration.

ORM Models

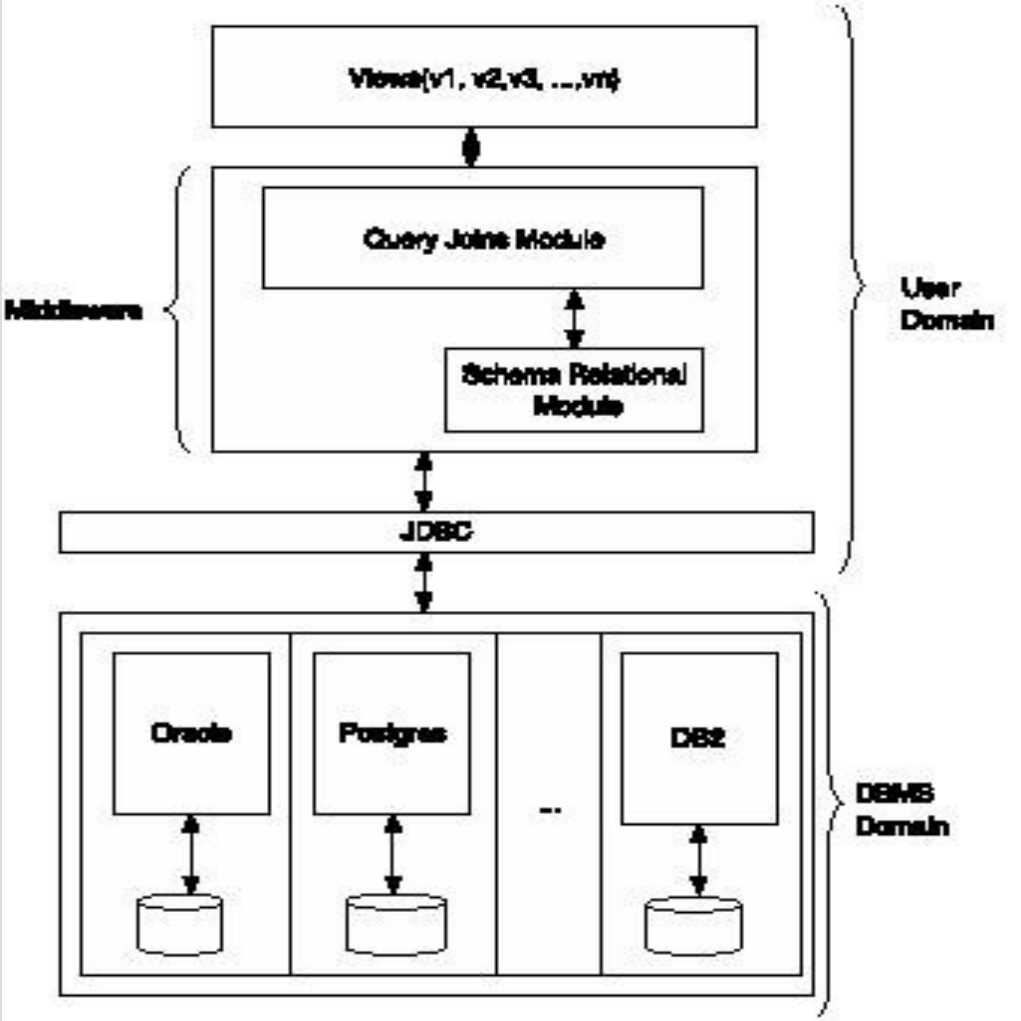
- Java offers a standard access to SQL databases, using JDBC;
- JDBC does not implement any ORM;
- JDBC is a flexible tool to create new approaches to ORM;

ORM Models

- As Java offers tools to create objects dynamically in application, we can use any object created by user;
- In this Application, we wish to follow the steps:
  - After query execution, a view in application is created, representing the query's result;
  - These “application views” can be integrated in application.

ORM Model

- Architecture

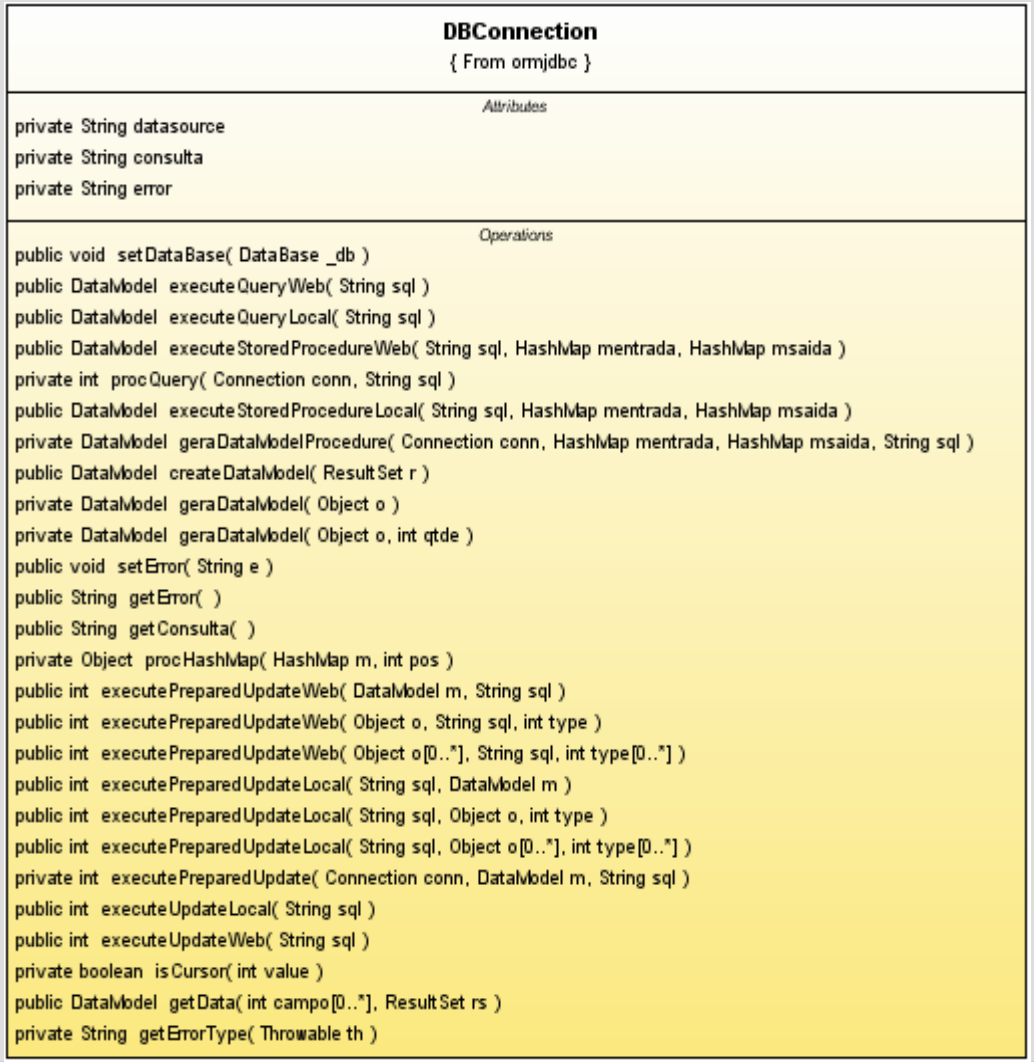


ORM Model

- Structure to access data:

ORM Model

- `DBConnection` class in detail:



ORM Model

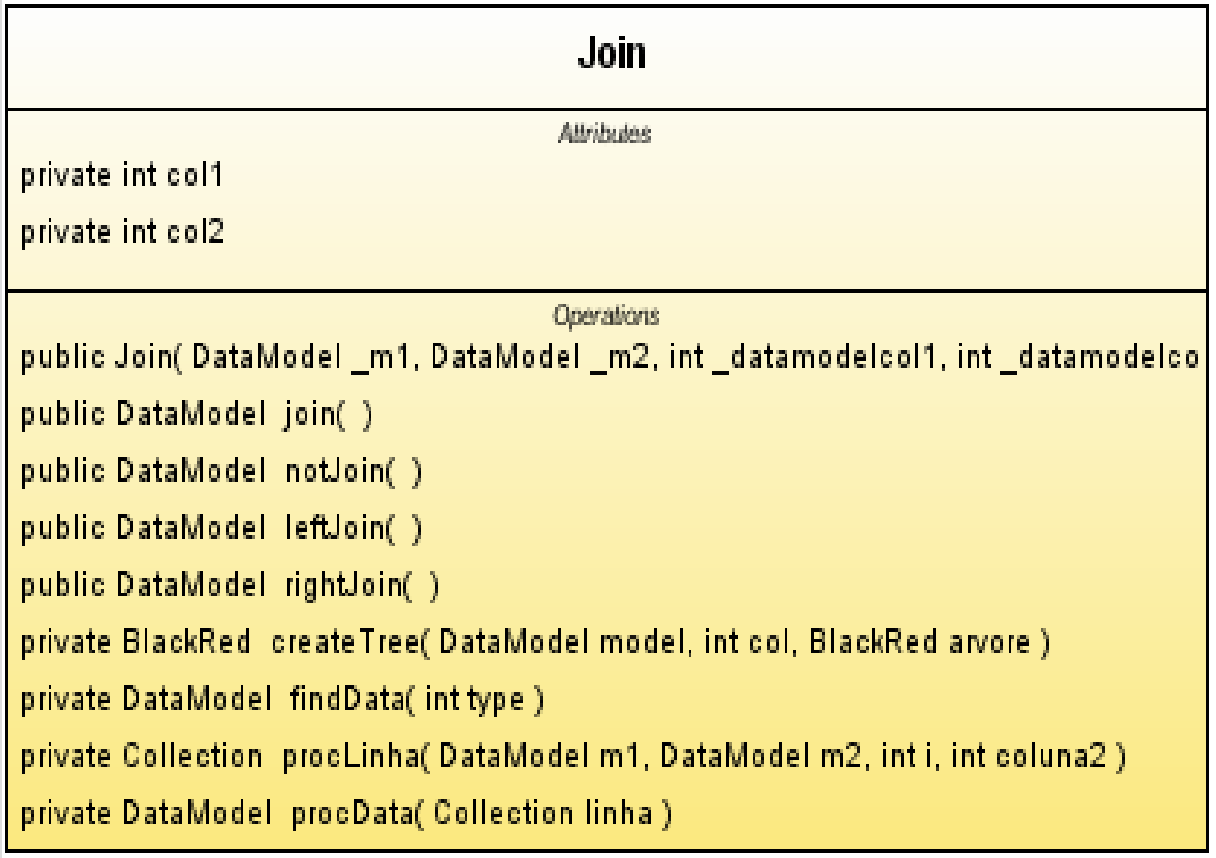
- A Class `Join` is used to create and manipulate the index;
- The Joins between tables from different datasources is implemented using the classes `BlackRed` and `BlackRedNode`;
- These classes implements a Black-Red tree;

ORM Model

- Developer chooses *one* field from each table, and an index will be created;
- The functions allowed to relate data are:
  - Join,
  - Not Join,
  - Left Join and
  - Right Join.

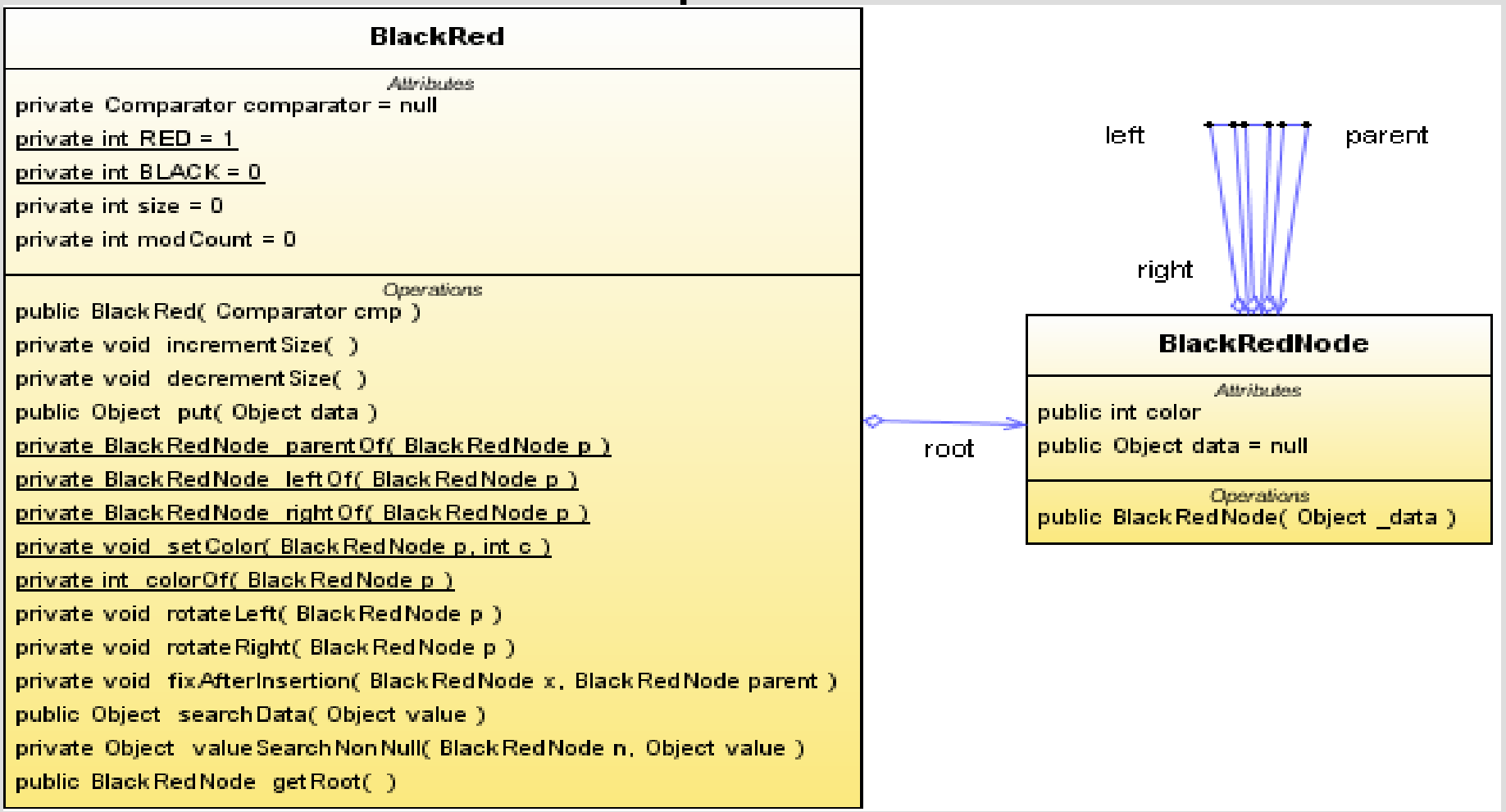
ORM Model

- Join class used to create and use the index:



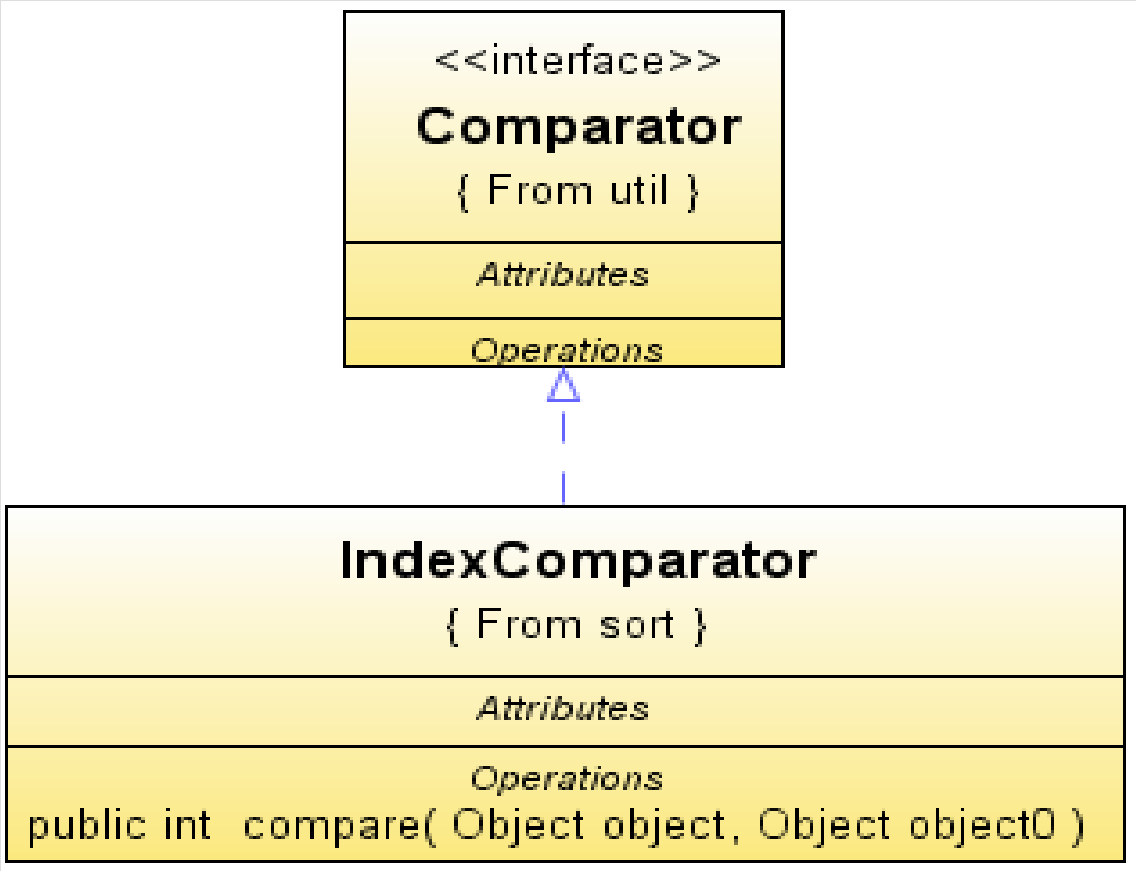
ORM Model

- The Index relationship:



ORM Model

- Comparator used to implement Object type conversion:



ORM Model

- The Data Types problem:
  - Developers execute a query, and a **`DataModel`** object is created;
  - In this object, the types described in database are transformed to Objects;
  - If the each column from both tables are numeric
  - The Numeric Objects are transformed to **`java.math.BigDecimal;`**
  - If one Object is numeric and the other is a String object, both are converted to **`java.math.BigDecimal;`**
  - If both are Strings, no conversion is required.

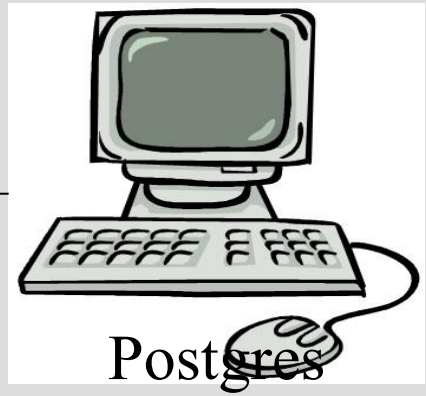
ORM Model

- This API also works with LOB data types;
- If the selection brings an information, which data type is a CLOB or BLOB type, two Objects are created:
  - **`CLOBType`**
  - **`BLOBType`**
- The **`CLOBType`** also implements a “like” search, that implements the same function as SQL “like” command.

Simulations

- Four situations where tested using the following network structure:

Oracle Server
NetFinity 7100 –
2 Pentiun III 700
Mhz
DataBase 1



Postgres
Server
mobile AMD
Athlon 1.8
Ghz
DataBase 2

Client
Machine
Linux
Pentium III
700 Mhz

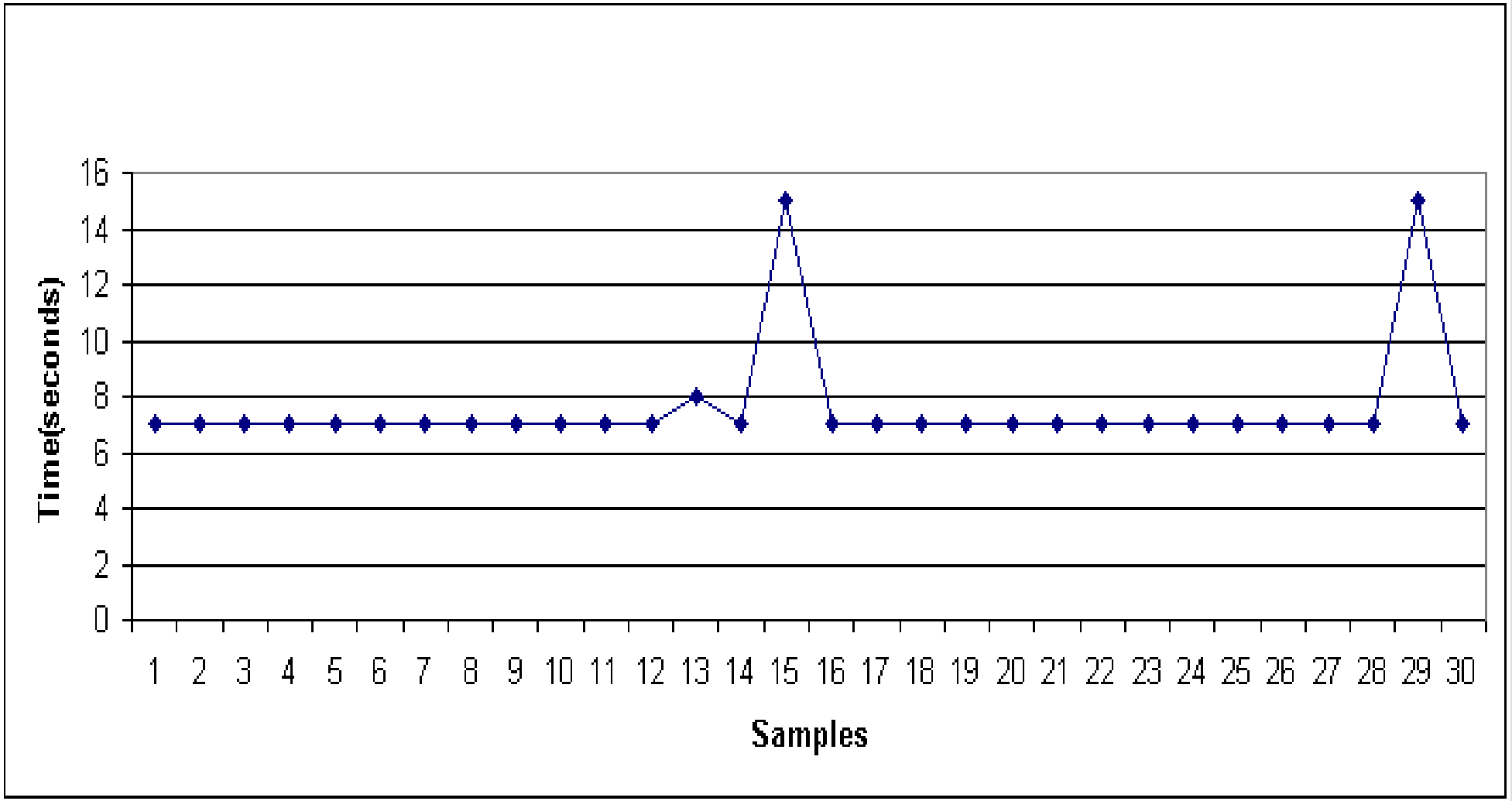
Ehternet Network

Simulations

- The simulations were divided in 4 types:
  - *Situation 1* - Only one column and one row from a table in DataBase 1 (Oracle) was related with one column from DataBase 2;
  - *Situation 2* - Only one column and one row from table in DataBase 1 was related with all columns from DataBase 2;
  - *Situation 3* - All columns and one row from table in DataBase 1 related with all columns from DataBase 2;
  - *Situation 4* - All columns and rows from Database 1 related with all rows and columns from Database 2.

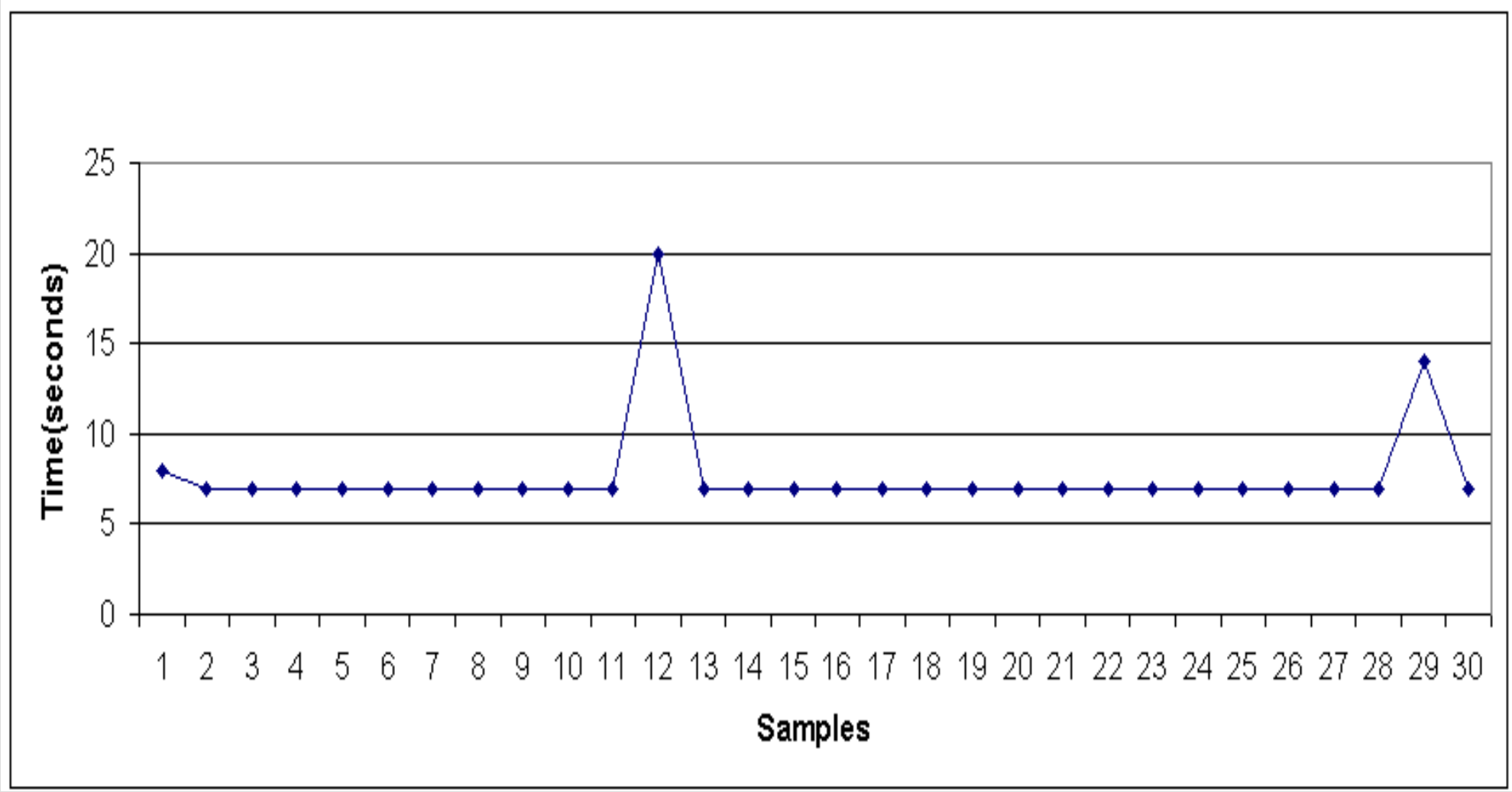
Simulations

- Results for Simulation 1:



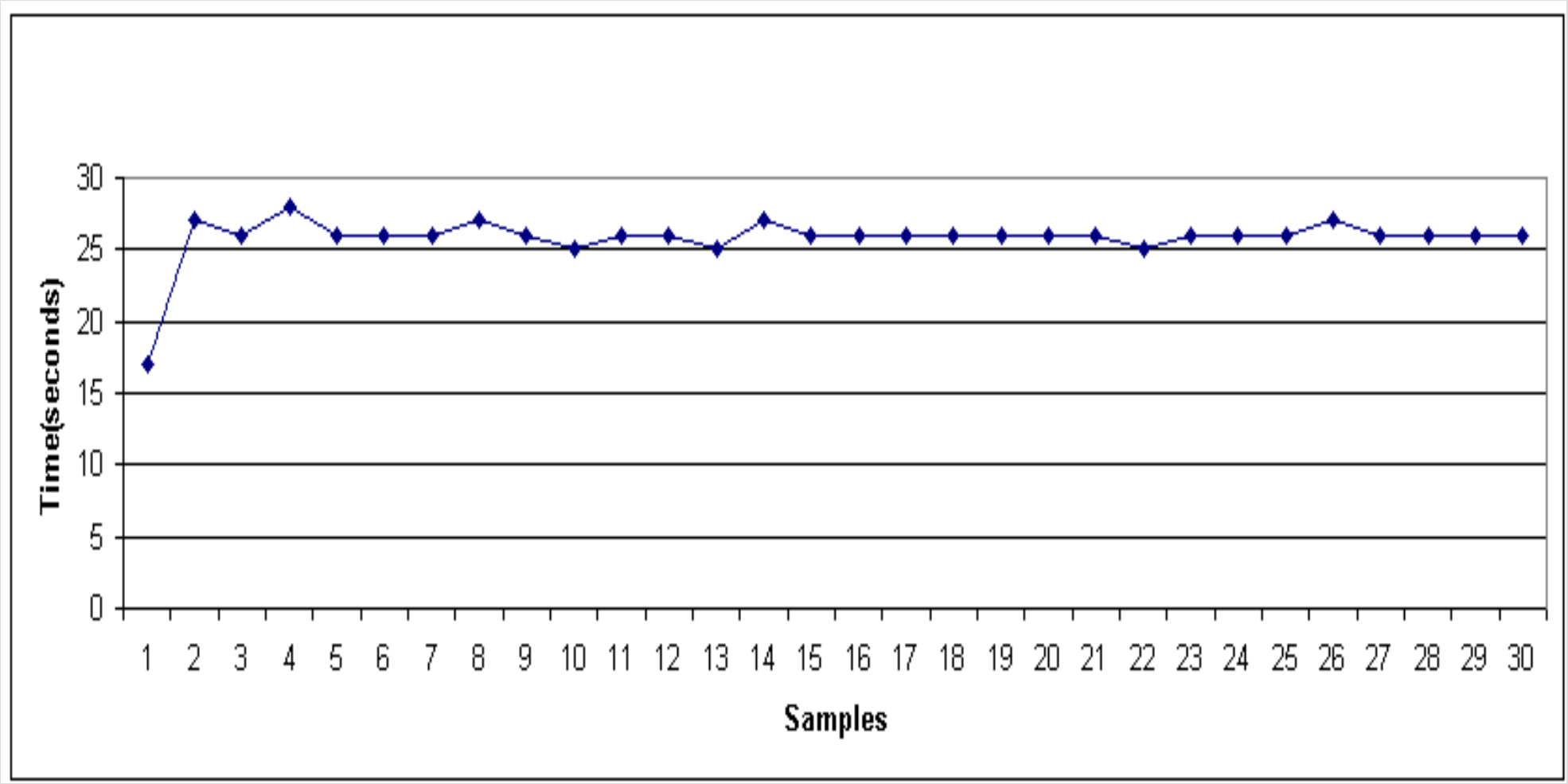
Simulations

- Result for Simulation 2:



Simulations

- Results for Situation 3:



Simulations

- The results for Situation 4 were not analysed, because memory failures;
- Main reason is the high number of objects allocated in memory
  - Possible solution: Create a cache system to store in hard drive data not used.
- If queries do not require a high number of columns, the system does not get any error.

Conclusion

- This presentation showed a API to execute a ORM using JDBC;
- Tests were executed with two different databases;
- This program does not identify homonyms or synonyms;

Conclusion

- This API still demands the presence of manufacturer's driver to access database;
- This API executes the data integration in application environment – no overload to database servers;
- This presentation did not present the Object materialization;

Conclusion

- It is necessary implement a cache to **`DataModel`** Objects;
  - Avoid memory limitations
- This API needs an approach to work with XML databases;
- This program has been in use to integrate data from different databases for almost 1,5 year at State University of Amazonas - Brazil.
- This program also was used to access LDAP servers and integrate their informations with our Academic system – 6 months.