INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Programmation Orientée Objet

Sami Yangui, Ph.D.
A. Prof and CNRS LAAS Researcher

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Lecture 4: Java Database Connectivity

October 23, 2019

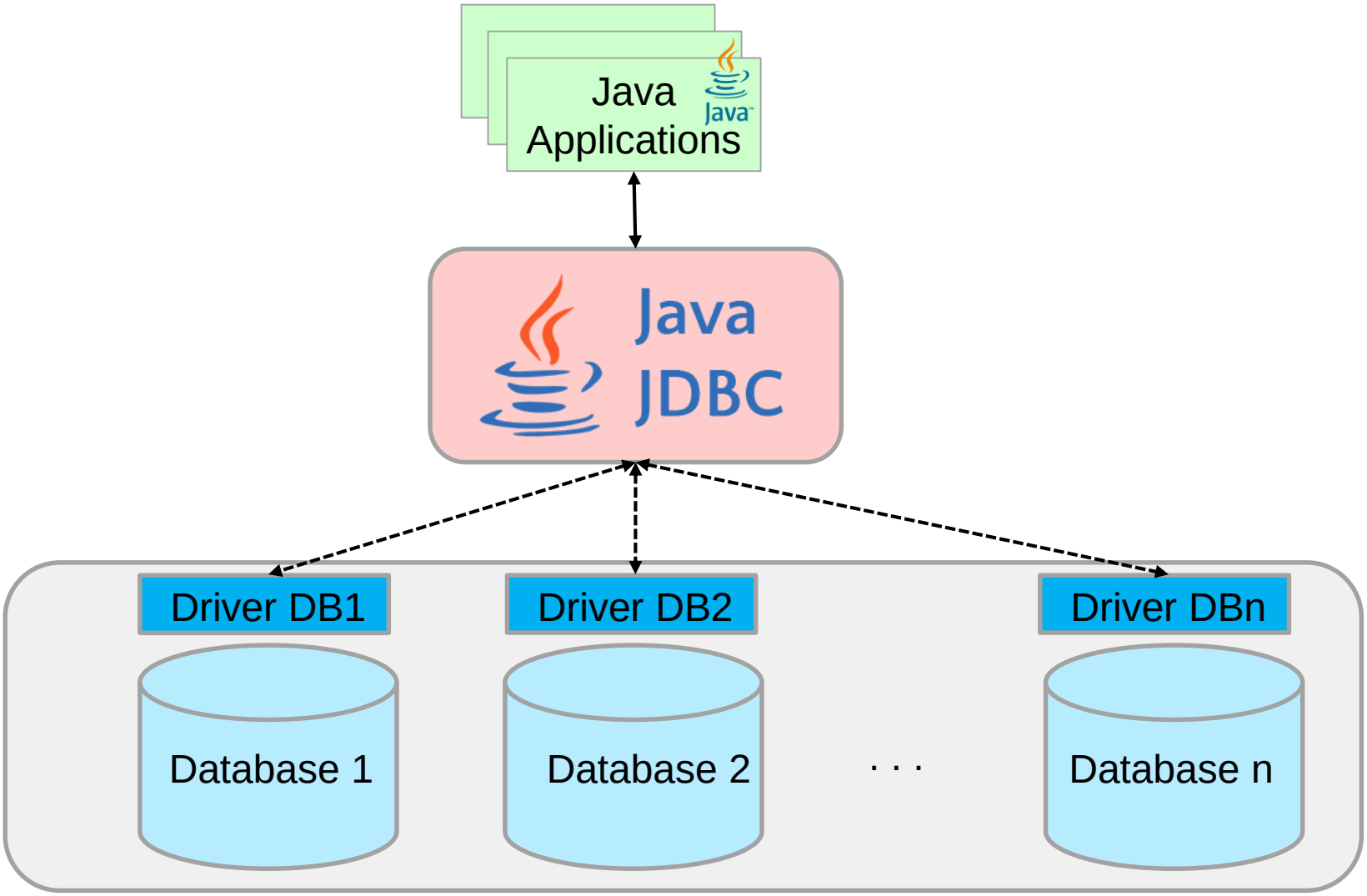
OUTLINE

- Introduction to JDBC
- JDBC System Overview
- JDBC Drivers Types
- JDBC Architecture
- JDBC Basic Ingredients
- Mapping SQL Types to JAVA Types
- TIME in SQL and JAVA

INTRODUCTION TO JDBC

- JDBC is used for accessing databases from Java applications
- Information are transferred from relations to objects and vice-versa
  - *Databases* are optimized for *searching/indexing*
  - *Objects* are optimized for *enginnering/processing*

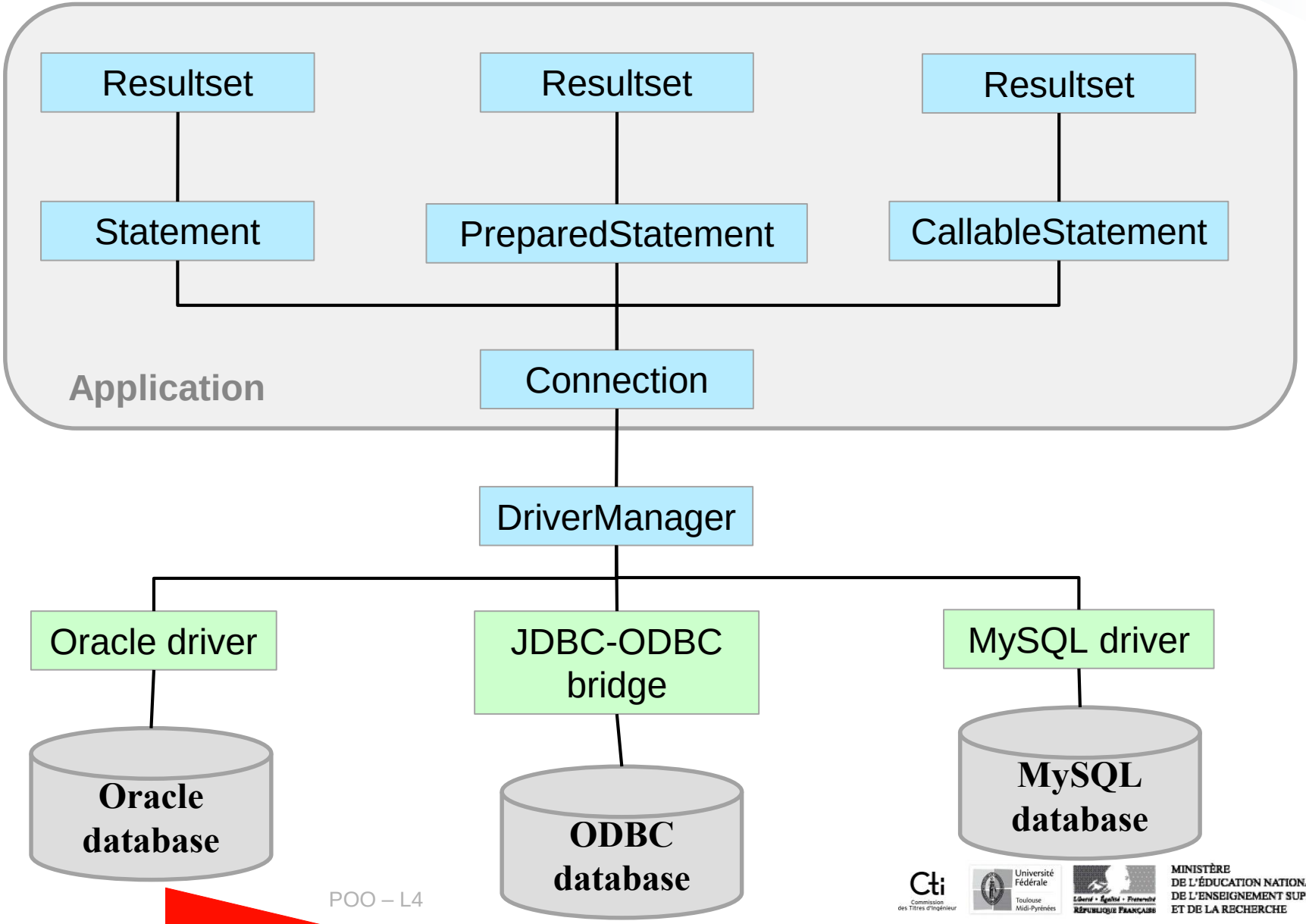
JDBC SYSTEM OVERVIEW



JDBC DRIVER TYPES

Driver type	Description
ODBC/JDBC bridge	Maps JDBC calls to ODBC driver calls on the client
Native API-Java	Maps JDBC calls to native calls on the client
JDBC Network-All Java	Maps JDBC calls to « network » protocol, which calls native methods on server
Native Protocol-All Java	Directly calls RDBMS from the client

JDBC ARCHITECTURE



JDBC BASIC INGREDIENTS

1. Declare a database connection object

```
Connection con=null;
```

2. Load the driver class file

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

3. Make a database connection

```
Con = DriverManager.getConnection("jdbc:odbc:MovieCatalog");
```

4. Create a statement object

```
Statement statement = con.createStatement();
```

5. Execute the statement

```
ResultSet rs = statement.executeQuery ("SELECT *" + FROM table) ;
```

6. Close the database connection

```
Con.close();
```

JDBC BASIC INGREDIENTS

Loading the driver

- Register the driver indirectly

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

  - Class.forName loads the specified class
  - When OracleDriver is loaded, it automatically
    - Creates an instance of itself
    - Registers this instance with the DriveManager

- Register the driver directly
  - The driver class can be given as an argument of the application

```
Driver driver = new ORG.sql.sqlDriver();

DriverManager.registerDriver(driver);
```

JDBC BASIC INGREDIENTS

Connecting to the database

- Implemented by the ***Connection*** class
  - Every database is identified by an URL
- Given an URL, DriverManager looks for the driver that can talk to the corresponding database
- DriverManager tries all registered drivers, until a suitable one is found

JDBC BASIC INGREDIENTS

- Interaction with the database

- There are three different interfaces
  - ***Statement***, ***PreparedStatement***, ***CallableStatement***
- All are interfaces, hence canot be instantiated
  - They are actually created by ***Connection***
- ***Statement*** objects are used to:
  - Query the database
  - Update the database

JDBC BASIC INGREDIENTS

Querying with Statement

- The ***executeQuery()*** method returns a ***ResultSet*** object representing the query result

```
String query = "SELECT *" + " FROM table";

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(query);
```

JDBC BASIC INGREDIENTS

Changing database with Statement

- The ***executeUpdate()*** method is used for data manipulation
  - Insert, delete, update, create table, etc. (Anything other than querying!)
- Returns the number of rows modified

```
String query = "DELETE FROM table where name='noname'";

Statement stmt = con.createStatement();

int delnum = stmt.executeUpdate(query);
```

JDBC BASIC INGREDIENTS

Prepared Statement

- Prepared statements are used for queries that are executed many times
- Parsed (compiled) by the RDBMS only once
- Column values can be set after the compilation
  - Instead of values, use "?" that represents a placeholder to be substituted later with actual values

JDBC BASIC INGREDIENTS

Prepared Statement (Cont.)

```
String query = "SELECT * FROM table where name=?";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setString(1, "noname");
ResultSet rs = pstmt.executeQuery();
```

JDBC BASIC INGREDIENTS

Prepared Statement (Cont.)

```
String query = "DELETE FROM table where name=?";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setString(1, "noname");
int delnum = pstmt.updateQuery();
```

JDBC BASIC INGREDIENTS

- Compare these two codes

```
String val = "abc";
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM R where A=?");
pstmt.setString(1, val);
ResultSet rs = pstmt.executeQuery();
---------
String val = "abc";
Statement stmt = con.createStatement();
ResultSet rs = rs.executeQuery("SELECT * FROM R where A="+val)
```

JDBC BASIC INGREDIENTS

- Will this work?

```
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM ?");

pstmt.setString(1, TABLE);
```

JDBC BASIC INGREDIENTS

- The ***ResultSet*** class
  - It provides access to the tables generated as results of executing a ***Statement*** queries
  - Only one ***ResultSet*** per ***Statement*** can be open at the same time
  - The table rows are retrieved in sequence
    - A ***ResultSet*** maintains a cursor pointing to its current row
    - The ***next()*** method moves the cursor to the next row

JDBC BASIC INGREDIENTS

- The ***ResultSet*** class (methods)
  - The ***next()*** method – returns *Boolean*
    - Activates the next row
    - The first call to ***next()*** activates the first row
    - Returns *false* if there is no more rows
  - The ***close()*** method – returns *Void*
    - Disposes of the ***ResultSet***
    - Enables to re use the ***Statement*** that created it
    - Called by most ***Statement*** methods

JDBC BASIC INGREDIENTS

- The ***ResultSet*** class (methods) – Cont.
  - The ***getType(int columnIndex)*** method – returns *Type*
    - Returns the given field as the given type
    - Indices start at 1 and not 0!
  - The ***getType(String columnName)*** method – returns *Type*
    - Same, but uses name of field
    - Less efficient

E.g. ***getString(columnIndex)***, ***getInt(columnName)***, ***getTime***, ***getBoolean***, etc.

JDBC BASIC INGREDIENTS

- The ***ResultSet*** class (methods) – Cont.

  - The ***findColumn(String columnName)*** method – returns *int*
    - Locks up column index given column name
  - JDBC 2.0 includes scrollable resultsets
    - Additional methods included are: '***first***', '***last***', '***previous***', etc.

JDBC BASIC INGREDIENTS

The *ResultSet* class – Example

```
Statement stmt = con.createStatement();
ResultSet rs = rs.executeQuery("SELECT name, salary FROM employees");
// Print the result
while (rs.next()){
System.out.print(rs.getString(1)+ ":");
System.out.println(rs.getDouble("salary"));
}
```

MAPPING SQL TYPES TO JAVA TYPES

SQL TYPE	JAVA TYPE
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	Java.math.BigDecimal
BIT	Boolean
TINYINT	Byte
SMALLINT	Short
INTEGER	Int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	Byte[]
DATE	Java.sql.Date
TIME	Java.sql.Time
TIMESTAMP	Java.sql.Timestamp

NULL VALUES

- In SQL, NULL means the field is empty
  - Not the same as 0 or " "
- In JDBC, the test for NULL for the last read field should be explicit
  - ResultSet.wasNull(column)

E.g. ***getInt(column)*** will return 0 if the value is either 0 or NULL

RESULTSETMETADATA CLASS

- Implements information about the properties of the columns in a ***ResultSet*** object
  - Example: Write the columns of the result set

```
ResultSetMetaData rsmd = rs.getMetaData();
int numcols = rsmd.getColumnCount();
for (int i=1; i<=numcols; i++) {
System.out.print(rsmd.getColumnLabel(i)+" ");
}
```

TIME IN DATABASE AND JAVA

- Times in SQL are notoriously non-standard
- Java defines three classes to help
  - Java.sql. Date
    - Year, month, day
  - Java.sql.Time
    - Hours, minutes, seconds
  - Java.sql.Timestamp
    - Year, month, day, hours, minutes, seconds, nanoseconds

CLEANING UP AFTER YOURSELF

- Remember to close the ***Connections***, ***Statements***, ***PreparedStatements***, and ***Resultsets***

```
con.close();
stmt.close();
pstmt.close();
rs.close();
```