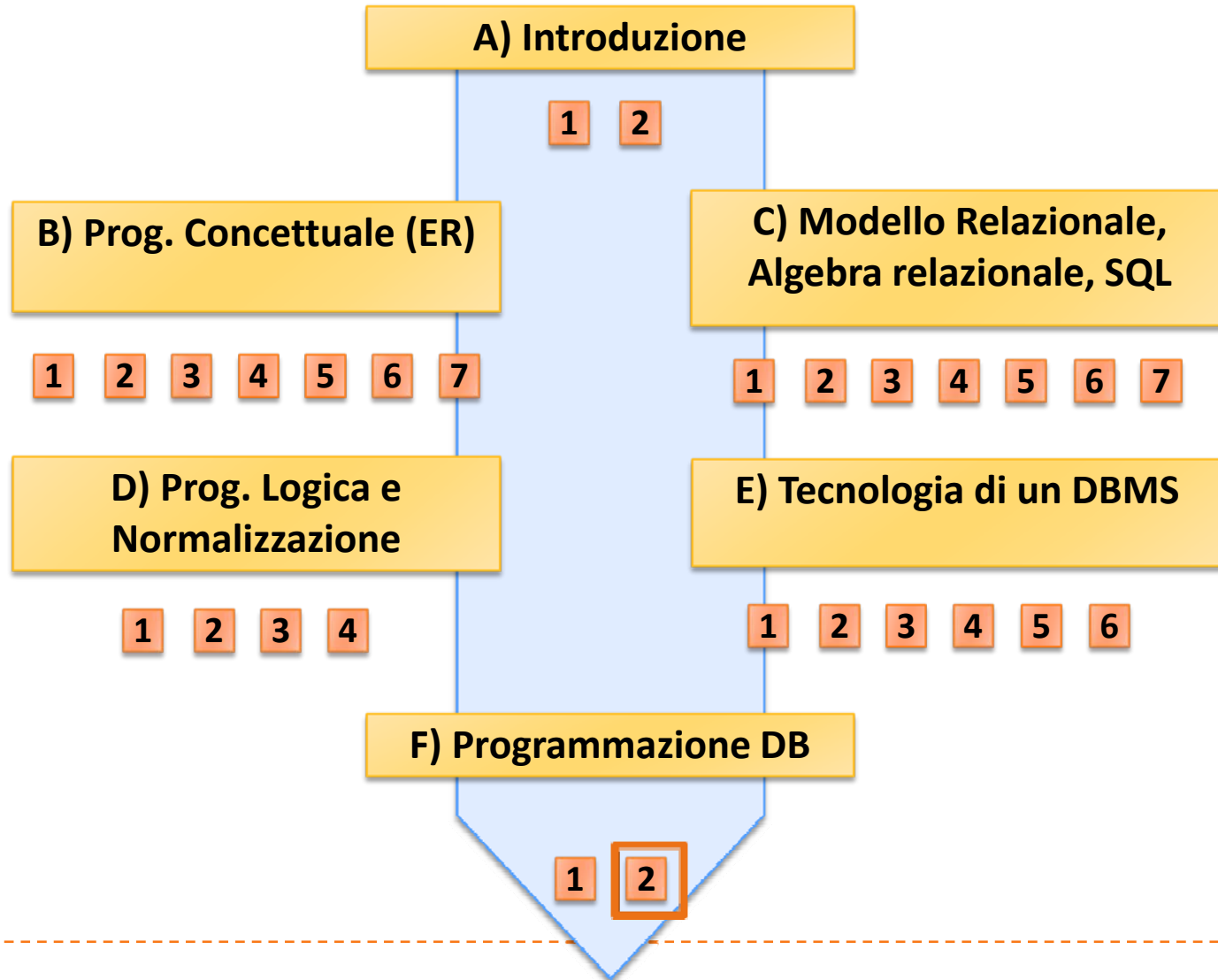


Basi di Dati

jdbc

Basi di Dati – Dove ci troviamo?



Download

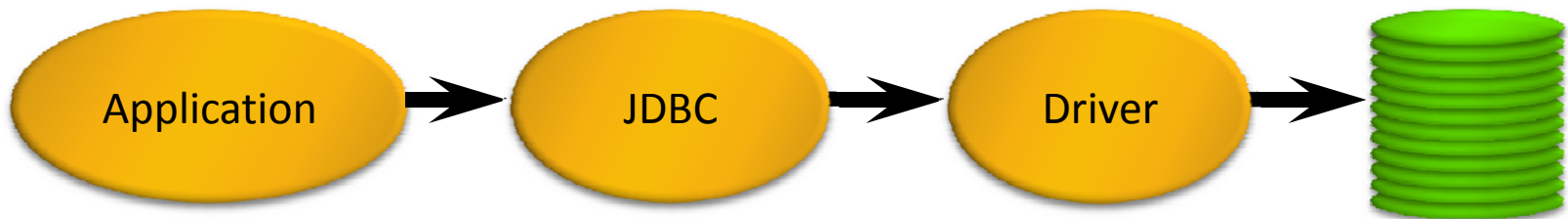
- ▶ Download PostgreSQL
 - ▶ <http://www.postgresql.org/download/>
- ▶ Download jdbc driver for PostgreSQL
 - ▶ <http://jdbc.postgresql.org/download.html>
- ▶ Create database
 - ▶ Esami
- ▶ Create
 - ▶ User 'user', Password 'pw'

JDBC Goals

- ▶ SQL-Level
- ▶ 100% Pure Java
- ▶ Keep it simple
- ▶ High-performance
- ▶ Leverage existing database technology
 - ▶ [why reinvent the wheel?](#)
- ▶ Use strong, static typing wherever possible
- ▶ Use multiple methods to express multiple functionality

JDBC Architecture

- ▶ Java code calls JDBC library
- ▶ JDBC loads a *driver*
- ▶ Driver talks to a particular database
- ▶ Can have more than one driver -> more than one database
- ▶ Ideal: can change database engines without changing any application code



java.sql

- ▶ JDBC is implemented via classes in the `java.sql` package

DriverManager

- ▶ DriverManager tries all the drivers
- ▶ Uses the first one that works
- ▶ When a driver class is first loaded, it registers itself with the DriverManager
- ▶ Therefore, to register a driver, just load it!

Registering a Driver

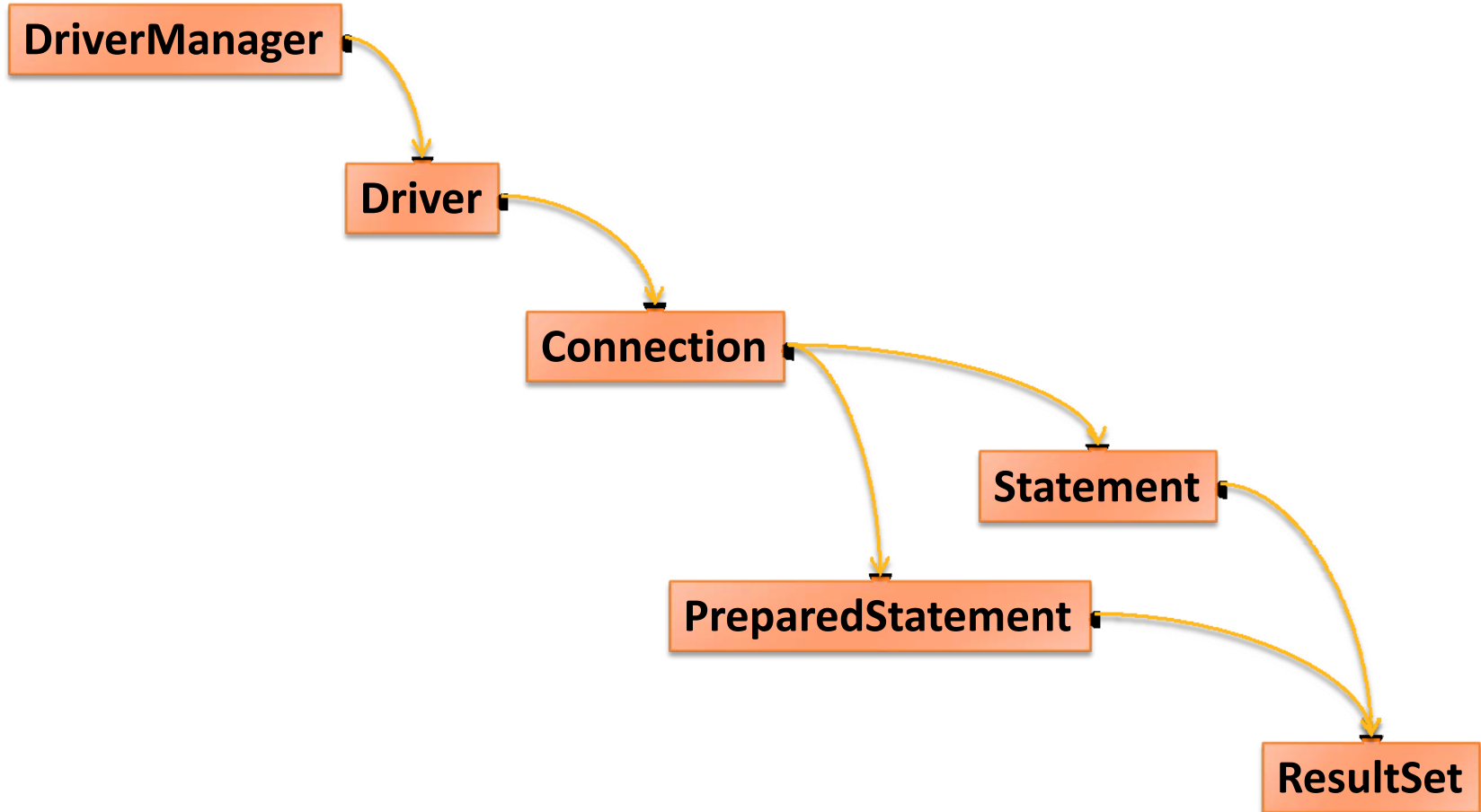
- ▶ Statically load driver

```
Class.forName("org.postgresql.Driver");  
Connection c =  
    DriverManager.getConnection(...);
```


JDBC Object Classes

- ▶ DriverManager
 - ▶ Loads, chooses drivers
- ▶ Driver
 - ▶ connects to actual database
- ▶ Connection
 - ▶ a series of SQL statements to and from the DB
- ▶ Statement/PreparedStatement
 - ▶ a single SQL statement
- ▶ ResultSet
 - ▶ the records returned from a Statement/PreparedStatement

JDBC Class Usage



JDBC URLs

jdbc:subprotocol:source

- ▶ each driver has its own subprotocol
- ▶ each subprotocol has its own syntax for the source

jdbc:odbc:DataSource

- ▶ e.g. `jdbc:odbc:Northwind`

jdbc:mysql://host[:port]/database

- ▶ e.g. `jdbc:mysql://foo.nowhere.com:4333/accounting`

jdbc:postgresql://host:port/database

- ▶ e.g. `jdbc:postgresql://localhost:5432/esami`

DriverManager

`Connection getConnection`

`(String url, String user, String password)`

- ▶ Connects to given JDBC URL with given user name and password
- ▶ Throws `java.sql.SQLException`
- ▶ Returns a `Connection` object

Connection

- ▶ A Connection represents a session with a specific database.
- ▶ Within the context of a Connection, SQL statements are executed and results are returned.
- ▶ Can have multiple connections to a database
 - ▶ NB: Some drivers don't support serialized connections
 - ▶ Fortunately, most do (now)
- ▶ Also provides “metadata” - information about the database, tables, and fields
- ▶ Also methods to deal with transactions

Obtaining a Connection

```
try{
    Class.forName ("org.postgresql.Driver"); // Load the Driver
    Connection conn = DriverManager.getConnection
        ("jdbc:postgresql://localhost:5432/esami", "user", "pw" );
    Statement stmt = conn.createStatement();
    //...
    stmt.close();
    conn.close();
}
catch (ClassNotFoundException e) {
    e.printStackTrace();
}
catch (SQLException e) {
    e.printStackTrace();
}
```

Connection Methods

`Statement createStatement()`

- ▶ returns a new Statement object

`PreparedStatement prepareStatement(String sql)`

- ▶ returns a new PreparedStatement object

Statement

- ▶ A Statement object is used for executing a static SQL statement and obtaining the results produced by it.

Statement Methods

ResultSet executeQuery(String)

- ▶ Execute a SQL statement that returns a single ResultSet.

int executeUpdate(String)

- ▶ Execute a SQL INSERT, UPDATE or DELETE statement. Returns the number of rows changed.

boolean execute(String)

- ▶ Execute a SQL statement that may return multiple results.

Statement Methods

...

```
String sql =  
    "CREATE TABLE STUDENTI  
    (matr integer primary key, cognome varchar, nome varchar)";  
stmt.executeUpdate(sql);
```

```
sql =  
    "INSERT INTO STUDENTI VALUES(1, 'rossi', 'mario'),  
    (2, 'bianchi', 'sergio)";  
stmt.executeUpdate(sql);
```

```
sql =  
    "SELECT * FROM STUDENTI";  
ResultSet rs = stmt.executeQuery(sql);  
...
```



ResultSet

- ▶ A ResultSet provides access to a table of data generated by executing a Statement.
- ▶ Only one ResultSet per Statement can be open at once.
- ▶ The table rows are retrieved in sequence.
- ▶ A ResultSet maintains a cursor pointing to its current row of data.
- ▶ The 'next' method moves the cursor to the next row.
 - ▶ you can't rewind

ResultSet Methods

- ▶ **boolean next()**
 - ▶ activates the next row
 - ▶ the first call to next() activates the first row
 - ▶ returns false if there are no more rows
- ▶ **void close()**
 - ▶ disposes of the ResultSet
 - ▶ allows you to re-use the Statement that created it
 - ▶ automatically called by most Statement methods

ResultSet Methods

- ▶ *Type* `getType(int columnIndex)`
 - ▶ returns the given field as the given type
 - ▶ fields indexed starting at 1 (not 0)
- ▶ *Type* `getType(String columnName)`
 - ▶ same, but uses name of field
 - ▶ less efficient
- ▶ `int findColumn(String columnName)`
 - ▶ looks up column index given column name

ResultSet Methods

- ▶ String getString(int columnIndex)
- ▶ boolean getBoolean(int columnIndex)
- ▶ byte getByte(int columnIndex)
- ▶ short getShort(int columnIndex)
- ▶ int getInt(int columnIndex)
- ▶ long getLong(int columnIndex)
- ▶ float getFloat(int columnIndex)
- ▶ double getDouble(int columnIndex)
- ▶ Date getDate(int columnIndex)
- ▶ Time getTime(int columnIndex)
- ▶ Timestamp getTimestamp(int columnIndex)

ResultSet Methods

- ▶ `String getString(String columnName)`
- ▶ `boolean getBoolean(String columnName)`
- ▶ `byte getByte(String columnName)`
- ▶ `short getShort(String columnName)`
- ▶ `int getInt(String columnName)`
- ▶ `long getLong(String columnName)`
- ▶ `float getFloat(String columnName)`
- ▶ `double getDouble(String columnName)`
- ▶ `Date getDate(String columnName)`
- ▶ `Time getTime(String columnName)`
- ▶ `Timestamp getTimestamp(String columnName)`

ResultSet Methods

...

```
sql = "SELECT * FROM STUDENTI";
```

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

```
while (rs.next()) {
```

```
    int matr = rs.getInt("matr");
```

```
    String cognome = rs.getString("cognome");
```

```
    String nome = rs.getString("nome");
```

```
    System.out.println(matr+" "+cognome+" "+nome);
```

```
}
```

```
rs.close();
```

```
stmt.close();
```

...

Mapping Java Types to SQL Types

<u>SQL type</u>	<u>Java Type</u>
CHAR, <u>VARCHAR</u> , LONGVARCHAR	String
<u>NUMERIC</u> , DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, <u>DOUBLE</u>	double
BINARY, <u>VARBINARY</u> , LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp



PreparedStatement motivation

- ▶ Suppose we would like to run the query

```
SELECT * FROM STUDENT1  
WHERE name='sergio';
```

- ▶ But we would like to run this for all students (separately), not only 'sergio'...
- ▶ Could we create a variable instead of 'sergio' which would get a different name every time??..

PreparedStatement

- ▶ PreparedStatement prepareStatement(String)
 - ▶ returns a new PreparedStatement object
- ▶ Prepared Statements are used for queries that are executed many times with possibly different contents.
- ▶ A PreparedStatement object includes the query and is prepared for execution (precompiled).
- ▶ Question marks can be inserted as variables.
 - ▶ `setString(i, value)`
 - ▶ `setInt(i, value)`

} The i-th question mark is set to the given value.

PreparedStatement

```
...
sql = "SELECT * FROM STUDENTI WHERE nome = ? and matr > ?";
PreparedStatement preparedStatement = conn.prepareStatement(sql);
preparedStatement.setString(1, "sergio");
preparedStatement.setInt(2, 0);
rs = preparedStatement.executeQuery();
while (rs.next()) {
    int matr = rs.getInt(1);
    String cognome = rs.getString(2);
    String nome = rs.getString(3);
    System.out.println(matr+" "+cognome+" "+nome);
}
rs.close();
preparedStatement.close();
conn.close();
}
...
```

PreparedStatement

- ▶ Will this work?

```
PreparedStatement pstmt =  
    con.prepareStatement("select * from ?");  
pstmt.setString(1, "Sailors");
```

- ▶ No! We may put ? only instead of values

JDBC Class Diagram

