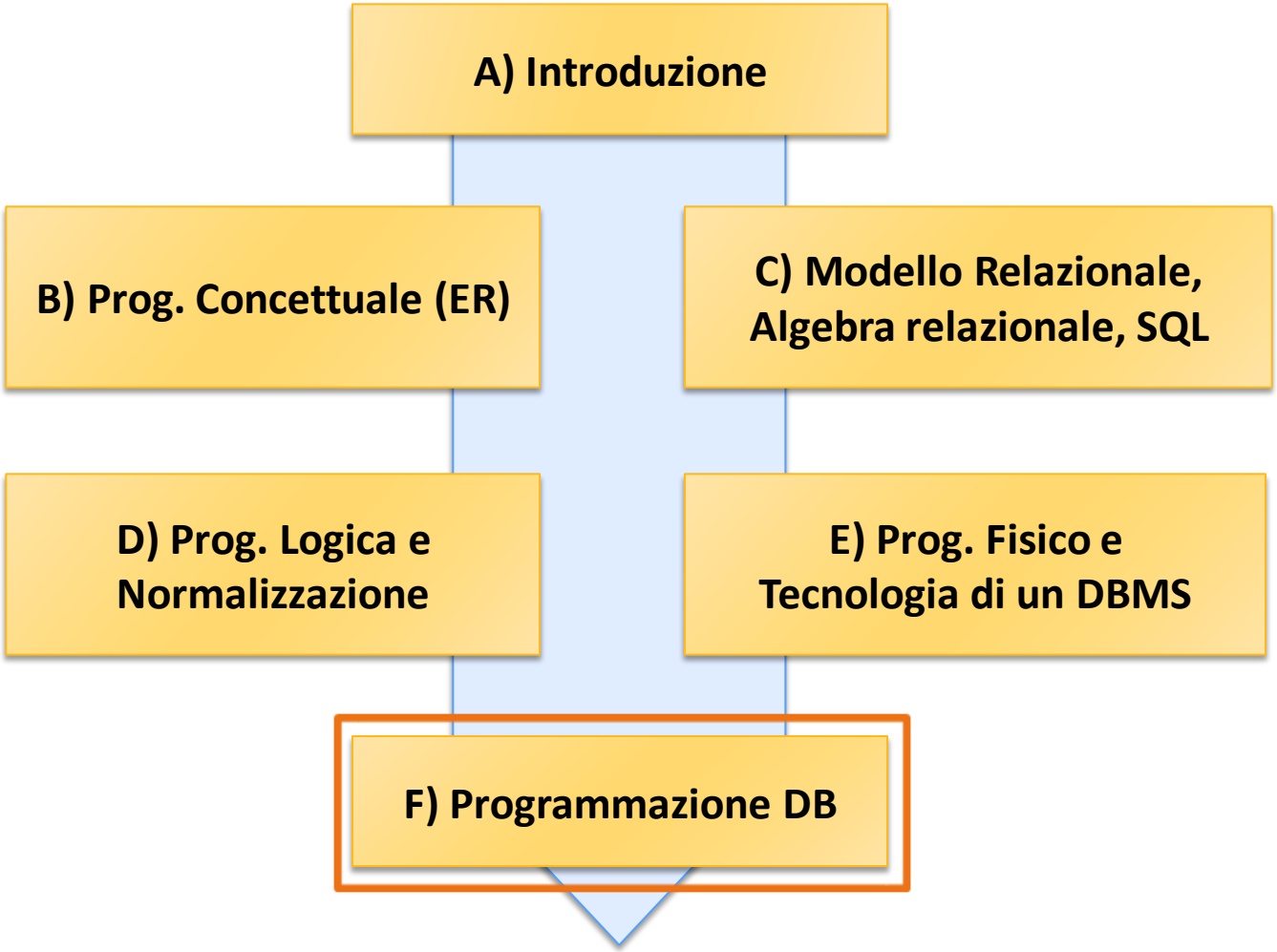
Basi di Dati

JDBC

Basi di Dati – Dove ci troviamo?



Download

- Scaricare PostgresSQL
  - http://www.postgresql.org/download/

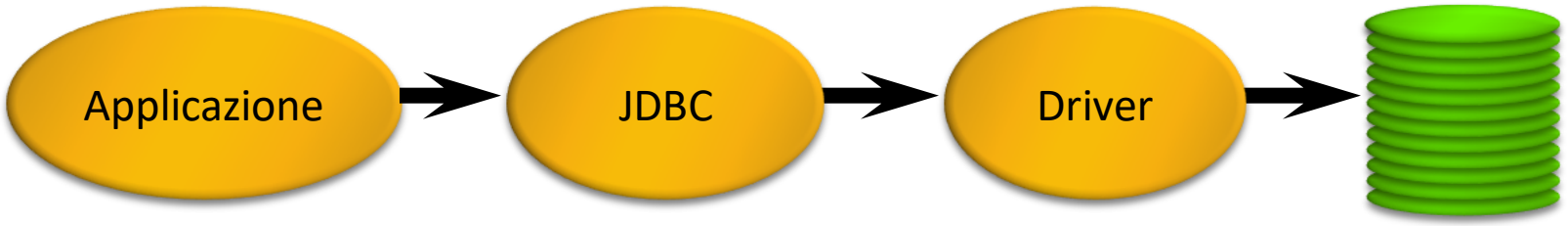
- Scaricare il driver JDBC per PostgresSQL
  - http://jdbc.postgresql.org/download.html

JDBC - Obiettivi

- SQL-Level
- 100% Puro Java
- Semplice
- Elevate prestazioni
- Sfruttare le tecnologie database esistenti
  - why reinvent the wheel?

Architettura JDBC

- Il codice Java richiama la libreria JDBC
- JDBC carica un *driver*
- Il Driver dialoga con un particolare database
- Ci può essere più di un driver -> più di un database
- Ideale: possibilità di cambiare engine database senza cambiare il codice dell'applicazione



java.sql

- JDBC è implementato attraverso le classi del package java.sql

DriverManager

- Il DriverManager prova tutti i driver
- Utilizza il primo funzionaente
- Appena una classe driver è caricata, si registra con il DriverManager
- Quindi, per registrare un driver è sufficiente caricarlo!

Registrare un Driver

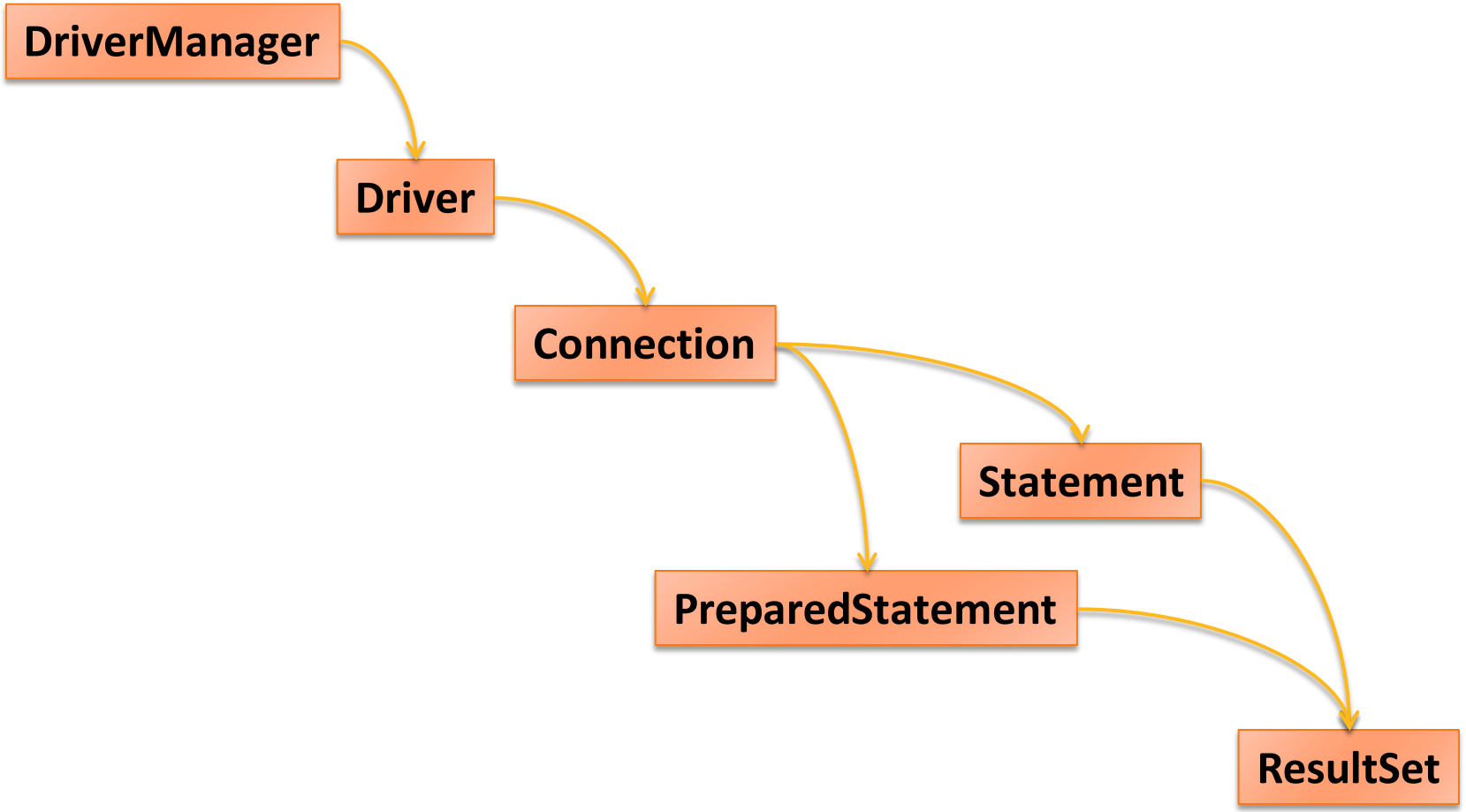
- Caricamento statico di un driver

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

Classi / Oggetti JDBC

- DriverManager
  - Carica / seleziona i driver
- Driver
  - Si connette al database
- Connection
  - Una serie di statement SQL
- Statement/PreparedStatement
  - Un singolo statement SQL
- ResultSet
  - I record restituiti da un Statement/PreparedStatement

Utilizzo delle classi JDBC



URL JDBC

jdbc:*subprotocol*:*source*

- ogni driver ha il suo subprotocol
- ogni subprotocol ha la sua sintassi verso la sorgente dati

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/videogame`

DriverManager

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

- Si connette alla URL JDBC con username e password specificati
- Lancia java.sql.SQLException
- Restituisce un oggetto Connection

Connection

- Una Connection rappresenta una sessione con uno specifico database
- Nel contesto di una Connection, è possible eseguire statement SQL e recuperare risultati.
- E' possibile avere connessioni multiple ad un database
- Fornisce anche "metadati" – informazioni sul database, le tabelle e i campi
- Include anche metodi per la gestione delle transazioni

Stabilire una Connection

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

Connection - Metodi

`Statement createStatement()`

- Restituisce un nuovo oggetto Statement

`PreparedStatement prepareStatement(String sql)`

- Restituisce un nuovo oggetto PreparedStatement

Statement

- Un oggetto Statement è usato per eseguire uno statement SQL statico e per ottenere i risultati da esso prodotti

Statement Methods

`ResultSet executeQuery(String)`

- Esegue uno statement SQL che restituisce un singolo ResultSet.

`int executeUpdate(String)`

- Esegue uno statement SQL INSERT, UPDATE o DELETE. Restituisce il numero di tuple modificate.

Statement Methods

```
...
String sql =
   "CREATE TABLE VIDEOGAME
   (titolo varchar primary key, piattaforma varchar, genere
   varchar)";
stmt.executeUpdate(sql);

sql =
   "INSERT INTO VIDEOGAME VALUES('The Secret of Monkey Island',
   'Amiga', 'Avventura'),
         ('Thimbleweed Park', 'PC', 'Avventura')";
stmt.executeUpdate(sql);

...
```

ResultSet

- Un ResultSet fornisce accesso ad una tabella di dati generata dall'esecuzione di uno Statement.
- Solo un ResultSet per Statement può essere aperto.
- Le tuple della tabella sono restituite in sequenza.
- Un ResultSet mantiene un cursore che punta alla prossima tupla.
- Il metodo 'next' muove il cursore alla tupla seguente.
  - Non è possibile "riavvolgere" il cursore

ResultSet - Metodi

- boolean next()
  - Attiva la prossima tupla
  - La prima chiamata a next() attiva la prima tupla
  - Restituisce false se non ci sono più tuple
- void close()
  - Elimina il ResultSet
  - Permette di riutilizzare lo Statement che lo ha creato

ResultSet - Metodi

- *Type* get*Type*(int columnIndex)
  - Restituisce il campo specificato del tipo specificato
  - I campi sono numerati a partire da 1 (non 0)
- *Type* get*Type*(String columnName)
  - Lo stesso, ma usa il nome del campo
  - Meno efficiente
- int findColumn(String columnName)
  - Restituisce l'indice di colonna dato il nome

ResultSet - Metodi

- 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 - Metodi

- 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 - Metodi

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

while (rs.next()) {
   String titolo = rs.getString(”titolo");
   String piattaforma = rs.getString(”piattaforma");
   String genere = rs.getString(”genere");
   System.out.println(titolo+" "+piattaforma+" "+genere);
}
rs.close();
stmt.close();
...
```

Mapping tra tipi Java e tipi SQL

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

PreparedStatement - motivazione

- Immaginiamo di voler eseguire la seguente query:

```
SELECT * FROM VIDEOGAME
WHERE piattaforma ='Amiga';
```

- Ma vorremmo farlo (separatamente) per ogni piattaforma, non solo per 'Amiga'…
- E' possibile creare una variabile invece di 'Amiga' che prenda di volta in volta un valore differente??

PreparedStatement

- PreparedStatement prepareStatement(String)
  - Restituisce un nuovo oggetto PreparedStatement

- I PreparedStatement sono utilizzati per query da eseguire numerose volte in diversi contesti.
- Un oggetto PreparedStatement include la query ed è "preparato" all'esecuzione (precompilato).
- Punti interrogativi fungono da variabili.
  - setString(i, value)
  - setInt(i, value)

  } L'i-esimo punto interrogativo è impostato al valore dato

PreparedStatement

```
...
sql = "SELECT * FROM VIDEOGAME WHERE piattaforma = ?";
PreparedStatement preparedStatement = conn.prepareStatement(sql);
preparedStatement.setString(1, "Amiga");
rs = preparedStatement.executeQuery();
while (rs.next()) {
   String titolo = rs.getString(1);
   String piattaforma = rs.getString(2);
   String genere = rs.getString(3);
   System.out.println(titolo+" "+piattaforma+" "+genere);
}
rs.close();
preparedStatement.close();
conn.close();
}
...
```

PreparedStatement

- Il seguente codice è corretto?

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

- No! Possiamo utilizzare ? solo al posto di valori

Class Diagram JDBC