# MyShopDB = MyShop + MySQL

- Adattiamo l'applicazione MyShop in modo da poter utilizzare un database come fonte di dati, invece dei soliti file testuali.
- I nuovi sorgenti si trovano nel file **`MyShopDB.zip`** scaricabile da:

  **`www.dimi.uniud.it/scagnett/LabTecLS3`**

  **`mitel.dimi.uniud.it/med`**

# La base di dati (I)

- Su **`latoserver.dimi.uniud.it`** è possibile utilizzare il proprio database personale a cui bisogna aggiungere le seguenti tabelle:
  - **`prodotti`**,
  - **`intestazioni_ordini`**,
  - **`righe_ordini`**,
  - **`utenti_backoffice`.**
- Lo script per creare e "popolare" le tabelle (tramite il client a linea di comando **`mysql`**) è scaricabile da

  **`www.dimi.uniud.it/scagnett/LabTecLS3`**

  **`mitel.dimi.uniud.it/med`**

# La base di dati (II)

- La tabella prodotti contiene i 3 elementi descritti nel file **`prodotti.txt`** della vecchia versione di MyShop:

```
[scagnetto@latoserver scagnetto]$ > mysql
mysql> use scagnetto
mysql> select * from prodotti;
+----+------------------+-----------------------------------------------+-------
-+
| ID | Prodotto         | Descrizione                                   | Prezzo
 |
+----+------------------+-----------------------------------------------+-------
-+
|  1 | Hard Disk 400 GB | 7200 giri, 16MB cache, Controller Serial Ata |    230
 |
|  2 | Monitor CRT      | 17''                                          |    130
 |
|  3 | Modulo RAM 512 MB | Modulo SDRAM 512 MB PC-133                   |     70
 |
+----+------------------+-----------------------------------------------+-------
-+
3 rows in set (0.00 sec)
```

# Il package **`myShop.jar`**

- Per maggiore “pulizia” del codice inseriamo in un package apposito le classi utili a modellare gli oggetti maggiormente utilizzati nell’applicazione.
- Tutte le pagine JSP che avranno bisogno di tali classi potranno utilizzarle utilizzando la direttiva l’attributo **`import`** della direttiva **`page`** nel modo seguente:

```
<%@ page import="myShop.*" %>
```

# Le classi del package **`myShop.jar`**

- **`Carrello`**: gli oggetti di questo tipo rappresentano i carrelli virtuali dei clienti del sito (con i metodi per aggiungere prodotti, eliminarli, aggiornare le quantità ecc.).
- **`Ordine`**: rappresenta un ordine completo di intestazione e righe corrispondenti ai prodotti ordinati.
- **`Prodotto`**: rappresenta un prodotto (con ID, nome, descrizione e prezzo).
- **`ProdottoSelezionato`**: estende la classe **`Prodotto`**, aggiungendo l'informazione sulla quantità selezionata dal cliente.
- **`Utente`**: rappresenta un utente dell'area riservata.

# Compilazione del package

- Compilazione delle singole classi:

```
javac Carrello.java Ordine.java Prodotto.java
ProdottoSelezionato.java Utente.java
```

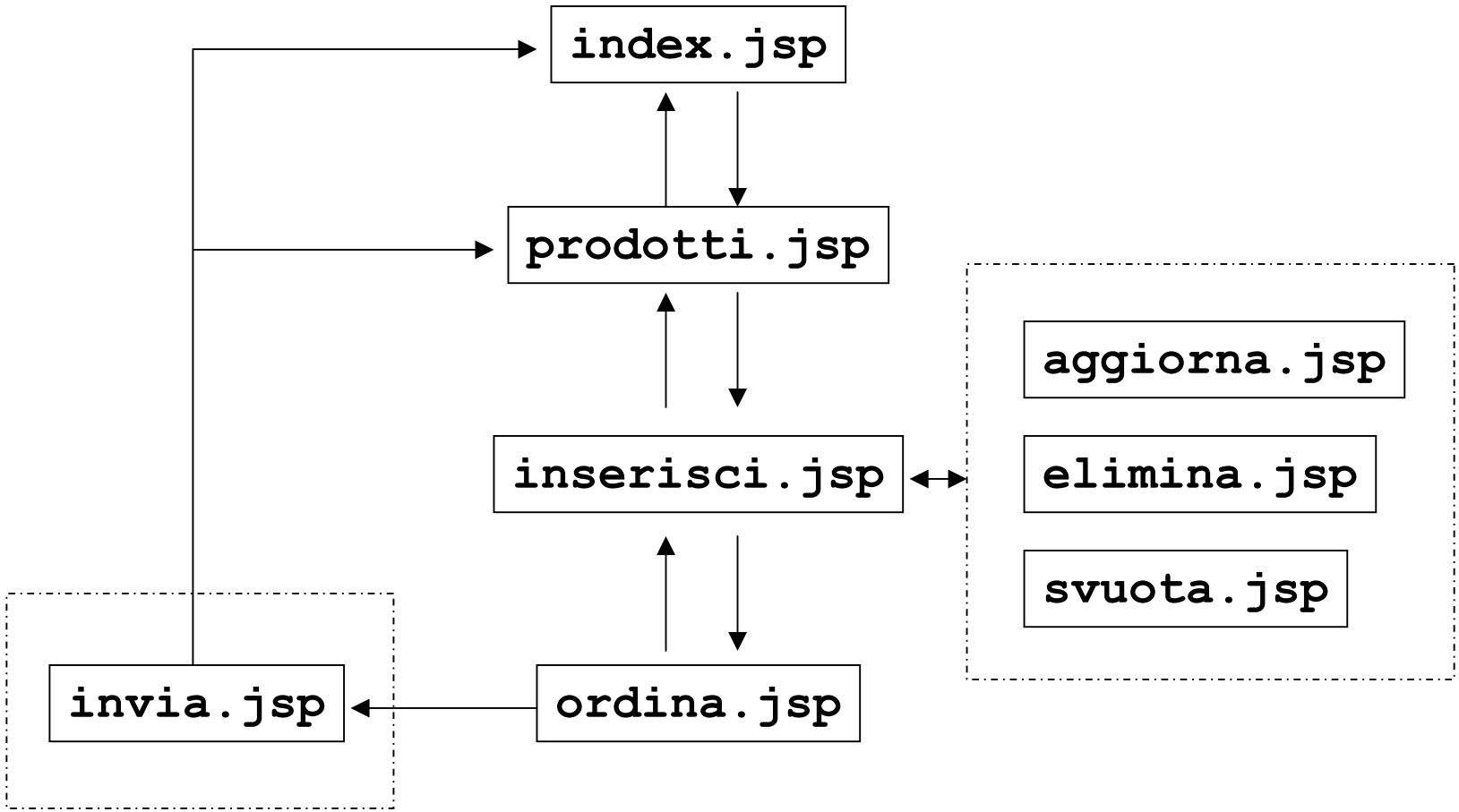
- Creazione dell'archivio **`jar`**:

```
jar cvf myShop.jar Prodotto.class
                   ProdottoSelezionato.class
                   Carrello.class
                   Ordine.class
                   Utente.class
```

# Deployment del package

- In locale:
  - creare le directory **`WEB-INF`** e **`WEB-INF/lib`** all'interno della directory principale dell'applicazione che ospita le pagine JSP di MyShop,
  - copiare **`myShop.jar`** in **`WEB-INF/lib`**.
- Su **`latoserver.dimi.uniud.it`**:
  - creare (se non esiste già) la directory **`/home/<nome utente>/servlets/WEB-INF/lib`**,
  - copiare **`myShop.jar`** in **`/home/<nome utente>/servlets/ WEB-INF/lib`**.
- Riavviare l'applicazione.

# L'architettura di MyShopDB rimane invariata



# File include

- Ai file utilizzati nelle direttive include se ne aggiunge uno nuovo:
  - **`sql.jsp`**
- In **`sql.jsp`** dichiariamo dei valori e delle funzioni utili per l'utilizzo di MySQL:
  - stringa di connessione,
  - nome utente e password per accedere a MySQL,
  - funzione per codificare gli apici nelle query SQL.

# Il problema degli apici (I)

- Supponiamo di voler inserire un nuovo record nella tabella anagrafica (composta da tre campi: **`Nome`**, **`Cognome`**, **`DataNascita`**):

```
INSERT INTO anagrafica (Nome, Cognome, DataNascita) VALUES ('Mario', 'Rossi', '1987-10-05')
```

- Le stringhe di caratteri (come **`Mario`**) devono essere racchiuse fra apici.
- Quindi per inserire una stringa, al cui interno compaiono degli apici come valore di un campo, bisogna "codificarli" per evitare che l'interprete SQL "spezzi" la stringa in più parti con gli errori di sintassi che ne conseguirebbero.

# Il problema degli apici (II)

- Gli apici in SQL vengono codificati raddoppiandoli:
  - es.: la stringa

    **`l'articolo`**

    viene codificata in

    **`l''articolo`**
- In questo modo è possibile delimitare la stringa con gli apici senza problemi nella sintassi SQL:

  **`'l''articolo'`**

# Il problema degli apici (III)

- In Java è sufficiente scrivere la funzione seguente per la codifica:

```
String CodificaApici(String s) {
        String codifica="";
        for(int i=0; i<s.length(); i++)
                if(s.charAt(i)=='\'')
                        codifica+="''";
                else
                        codifica+=s.charAt(i);
        return codifica;
}
```

# Il file `sql.jsp` (I)

```
<%@ page import="java.sql.*" %>
<%!
  String stringaConnessione="jdbc:mysql://localhost/<nome_db>";
  String utenteSQL="<utente>";
  String passwordSQL="<password>";
  int lunghezzaMassima=100;

  String CodificaApici(String s) {
        String codifica="";
        for(int i=0; i<s.length(); i++)
                if(s.charAt(i)=='\'')
                        codifica+="''";
                else
                        codifica+=s.charAt(i);
        return codifica;
  }
%>
```

Parametri per la connessione al server MySQL

# Il file `sql.jsp` (II)

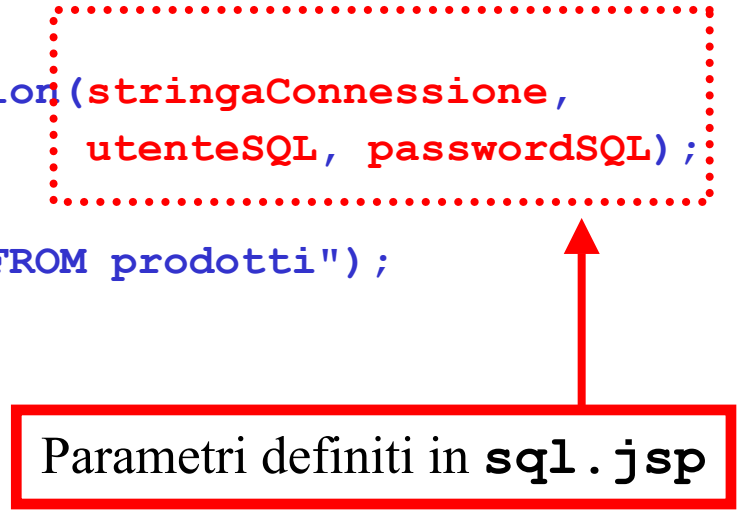
```
...
<%
  try {
        Class.forName("com.mysql.jdbc.Driver");
  } catch (ClassNotFoundException e) {
        response.sendRedirect("errore.jsp");
  }
%>
```

La pagina **`errore.jsp`** visualizza un messaggio di avvertimento per l'utente.

# Il file `prodotti.jsp` (I)

```
...
<%try {
  Connection c = DriverManager.getConnection(stringaConnessione,
                                             utenteSQL, passwordSQL);

  Statement s = c.createStatement();
  ResultSet r = s.executeQuery("SELECT * FROM prodotti");
%>
  ...
  <table border="0" cellpadding="5">
    <tr>
      <td class="intestazione_tabella">
        Prodotto
      </td>
      ...
      <td class="intestazione_tabella">
        Inserisci nel carrello
      </td>
    </tr>
```



# Il file `prodotti.jsp` (II)

```
<%
  NumberFormat fmt=NumberFormat.getInstance(Locale.ITALIAN);
  fmt.setMinimumFractionDigits(2);
  fmt.setMaximumFractionDigits(2);
  int riga=0;
  while(r.next()) {
    riga++;
    String stile=riga%2==0 ? "riga2_tabella" : "riga1_tabella";%>
    <tr>
      <form method="post" action="inserisci.jsp">
      <td class='<%= stile%>'><%= r.getString(2)%></td>
      <td class='<%= stile%>'><%= r.getString(3)%></td>
      <td class='<%= stile%>'><%= fmt.format(r.getFloat(4))%></td>
```

# Il file `prodotti.jsp` (III)

```
    <td class='<%= stile%>'>
      <input type="text" name="qt" value="1" size="2" />
    </td>
    <td class='<%= stile%>'>
      <input type="submit" name="ordina" value="&gt;&gt;" />
      <input type="hidden" name="pid" value="<%= r.getInt(1)%>" />
    </td>
  </form>
</tr>
<%}%>
</table>
<%r.close(); s.close(); c.close();
} catch (SQLException e) {
    response.sendRedirect("errore.jsp");
}%>
```

# Il file `inserisci.jsp`

- Il codice della pagina risulta ora notevolmente semplificato; ad esempio:
  - controllo del carrello vuoto:

```
Carrello carrello=null;
if(session.getValue("myShop.carrello")==null) {
  carrello=new Carrello();
  session.putValue("myShop.carrello",carrello);
} else
  carrello=(Carrello)session.getValue("myShop.carrello");
...
boolean carrelloVuoto=carrello.Vuoto();
```

  - controllo della presenza di un prodotto nel carrello:

```
if(carrello.TrovaIndiceProdotto(Integer.parseInt(idProdotto))==-1)
 ...
 else
 ...
```

# Il file svuota.jsp

```
<%@ page import="myShop.*" %>

<%
  Carrello carrello=null;

  if(session.getValue("myShop.carrello")==null)
    carrello=new Carrello();
  else
    carrello=(Carrello)session.getValue("myShop.carrello");

  carrello.Svuota();
  response.sendRedirect("inserisci.jsp?stampa=1");
%>
```

# Il file `elimina.jsp`

```
<%@ page import="myShop.*" %>
<%
  String idProdotto=request.getParameter("id");
  Carrello carrello=null;

  if(session.getValue("myShop.carrello")==null)
    carrello=new Carrello();
  else
    carrello=(Carrello)session.getValue("myShop.carrello");

  try {
    carrello.EliminaProdotto(Integer.parseInt(idProdotto));
  } catch (Exception e) {
        response.sendRedirect("errore.jsp");
  }
  response.sendRedirect("inserisci.jsp?stampa=1");
%>
```

# Il file aggiorna.jsp (I)

```
<%@ page import="myShop.*" %>
<%int numeroRighe=0;
  // Recupero del numero di righe (prodotti) del carrello.
  try { numeroRighe = Integer.parseInt(request.getParameter("righe"));
  } catch (NullPointerException e) {
          numeroRighe=0;
  } catch (NumberFormatException e) {
          numeroRighe=0;
  }
  Carrello carrello=null;
  // Recupero del carrello dalla sessione corrente
  if(session.getValue("myShop.carrello")==null)
    carrello=new Carrello();
  else
    carrello=(Carrello)session.getValue("myShop.carrello");
```

# Il file `aggiorna.jsp` (II)

```
try {

 for(int i=0;i<numeroRighe;i++) {
   // Recupero dell'id del prodotto
   String idProdotto=request.getParameter("id"+
                                     (new Integer(i)).toString());
   // Recupero della quantità di prodotto selezionata dal cliente
   int qtProdotto=0;
   try {
     qtProdotto = Integer.parseInt(request.getParameter("qt"+
                                    (new Integer(i)).toString()));
   } catch (NullPointerException e) {
     qtProdotto=0;
   } catch (NumberFormatException e) {
     qtProdotto=0;
   }
```

# Il file `aggiorna.jsp` (III)

```
carrello.ModificaQuantita(Integer.parseInt(idProdotto),
  qtProdotto);
        }
  // Eventuali errori...
  } catch (Exception e) {
        response.sendRedirect("errore.jsp");
  }
  /* Redirezionamento del browser per la nuova
   visualizzazione del carrello */
  response.sendRedirect("inserisci.jsp?stampa=1");
%>
```

# Il file **`invia.jsp`**

- E' presente la direttiva

  **`<%@ page isThreadSafe="false" %>`**
- Perché?
  - Il motivo risiede nel fatto che bisogna inserire un record nella tabella **`intestazioni_ordini`** ed uno o più record collegati dallo stesso **`ID`** nella tabella **`righe_ordini`**.
  - L'**`ID`** del nuovo ordine viene calcolato come il **`MAX(ID)`** dei vecchi ordini **`+1`**.
  - Quindi è necessario che soltanto un thread (richiesta) alla volta possa accedere al DB per calcolare correttamente l'ID del nuovo ordine (altrimenti potrebbero essere generati ID duplicati dalla pagina **`invia.jsp`**).

# Transazioni

- Siccome la direttiva

  ```
  <%@ page isThreadSafe="false" %>
  ```

  non fornisce garanzie assolute (Tomcat potrebbe creare più istanze in memoria della servlet corrispondente alla pagina JSP), utilizziamo il meccanismo delle transazioni offerto da JDBC.
- Una transazione permette di raggruppare atomicamente più query.
- Ciò significa che le query suddette vengono trattate come se fossero un'unica operazione indivisibile.
- Se anche una sola delle varie interrogazioni genera un errore, viene annullato anche l'effetto di tutte le altre.

# Transazioni (codice)

```
Connection c = DriverManager.getConnection(stringaConnessione,
   utenteSQL, passwordSQL);
     Statement s = c.createStatement();
…
try {
   c.setAutoCommit(false); /* disabilitazione del commit
    automatico */
   // query
   c.commit(); // se tutte le query vanno a buon fine
}
catch(SQLException e) {
   c.rollback(); /* in caso di errori in una query, viene
    annullato l'effetto anche delle query precedentemente
    andate a buon fine. */
   response.sendRedirect("errore.jsp");
}
```

# Esercizio

- Modificare la base di dati e l'applicazione in modo da visualizzare anche la quantità disponibile a magazzino per ogni prodotto.
- Scrivere una pagina che visualizzi le intestazioni di ogni ordine (numero d'ordine, data, nome, cognome, indirizzo, pagamento, consegna, spese di spedizione).

  La query da utilizzare è:

```
SELECT ID, Data, Nome, Cognome, Indirizzo,
Pagamento, Consegna, SpeseSpedizione FROM
intestazioni_ordini
```

# Esercizio

- Oltre all'intestazione visualizzare le righe (corrispondenti ai prodotti ordinati) di ogni ordine, usando la query seguente (da ripetere per ogni singolo ordine):

```
SELECT P.Prodotto , R.Prezzo,
R.Quantita FROM prodotti AS P,
righe_ordini AS R WHERE
P.ID=R.IDProdotto AND
R.IDOrdine=<valore del campo ID
estratto usando la query precedente>
```