EXAMPLE 1:

suppliers

snum	sname	status	city
S1	Suzi	3	Orlando
S2	Jim	2	Tampa
S3	Debi	5	Oviedo

shipments

snum	pnum	jnum	quantity
S1	P1	J1	90
S2	P2	J2	140
S1	P2	J1	40
S2	P3	J4	85

User command: insert into shipments values ('S1','P4','J2',350)

Result: shipments

snum	pnum	jnum	quantity
S1	P1	J1	90
S2	P2	J2	140
S1	P2	J1	40
S2	P3	J4	85
S1	P4	J2	350

Basic (brute force) business logic triggers the following changes to the suppliers table (since both S1 and S2 have a shipment with quantity>100:

suppliers

snum	sname	status	city
S1	Suzi	8	Orlando
S2	Jim	7	Tampa
S3	Debi	5	Oviedo

Correct business logic triggers the following change to the suppliers table (since only supplier S1 was directly impacted by the update):

suppliers

snum	sname	status	city
S1	Suzi	8	Orlando
S2	Jim	2	Tampa
S3	Debi	5	Oviedo

EXAMPLE 2:

suppliers

snum	sname	status	city
S1	Suzi	3	Orlando
S2	Jim	2	Tampa
S3	Debi	5	Oviedo

shipments

snum	pnum	jnum	quantity
S1	P1	J1	90
S2	P2	J2	120
S1	P2	J1	40
S2	P3	J4	85

User command: update shipments set quantity = quantity + 12

Result: shipments

snum	pnum	jnum	quantity
S1	P1	J1	102
S2	P2	J2	132
S1	P2	J1	52
S2	P3	J4	97

Basic business logic triggers the following changes in suppliers table

suppliers

snum	sname	status	city
S1	Suzi	8	Orlando
S2	Jim	7	Tampa
S3	Debi	5	Oviedo

Correct business logic triggers the following changes in the suppliers table

suppliers

snum	sname	status	city
S1	Suzi	8	Orlando
S2	Jim	2	Tampa
S3	Debi	5	Oviedo

(Basic Business Logic: Non-bonus version) Updating command for suppliers table:

run user issued command first – if business logic is triggered then run

this update command:

```
update suppliers
    set status = status + 5
    where snum in (select snum
                   from shipments
                   where quantity > 100)
```

Updating commands in MySQL:

insert into <table> values (<value-list>)

update <table> set <value-list> where <predicate>

replace into <table> values (<value-list>)

replace into <table> set <value-list>

Bonus Version Business Logic Update

Before issuing the user command (which you know will trigger the business logic) do the following:

```
create table beforeShipments like shipments;
insert into beforeShipments
    select * from shipments;
```

Then run user update command:

Then do the following:

```
update suppliers
    set status = status + 5
    where snum in (
        select distinct snum
        from shipments left join beforeShipments
            using (snum, pnum,jnum,quantity)
        where beforeShipments.snum is null and quantity > 100
    );
```

Note that this technique works for any updating command, whether it be a single row update or a multi-row update.

JSP Hints

In HTML file

```
<%-- start scriplet --%>
<%
  String sqlStatement = (String) session.getAttribute("sqlStatement");
  if (sqlStatement == null) sqlStatement = "select * from suppliers";
  String message = (String) session.getAttribute("message");
  if (message == null) message = " ";
 %>



<table>
        <%-- JSP expression to access servlet variable: message --%>
        <%=message%>
</table>
```

In Servlet (in doPost() method)

For queries:

ResultSet resultSet = statement.executeQuery(sqlStatement);

message = SQLFormatter.getHtmlRows(resultSet);

For updates:

MySQL returns an integer value for all cases.

before return

Need to set the session attributes, of which there are two, the "sqlStatement" which represents the SQL command entered by the user and the "message" which represents the return object from MySQL.

You will also need to set up a RequestDispatcher object which is an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server. The servlet container creates the RequestDispatcher object, which is used as a wrapper around a server resource located at a particular path or given by a particular name, in our case, it needs to reference the index.jsp front-end page where the results are to be directed. This dispatcher object needs to be "pointed" to the correct resource. This is done using the interface ServletConfig which contains a method named getServletContext() which returns a ServletContext object. This reference object invokes the getRequestDispatcher() method. The RequestDispatcher object acts as a wrapper for the resource located at the given path. A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

The pathname must begin with a / and is interpreted as relative to the current context root. For more details see: http://docs.oracle.com/javaee/5/api/javax/servlet/ServletContext.html#getRequestDispatcher(java.lang.String)

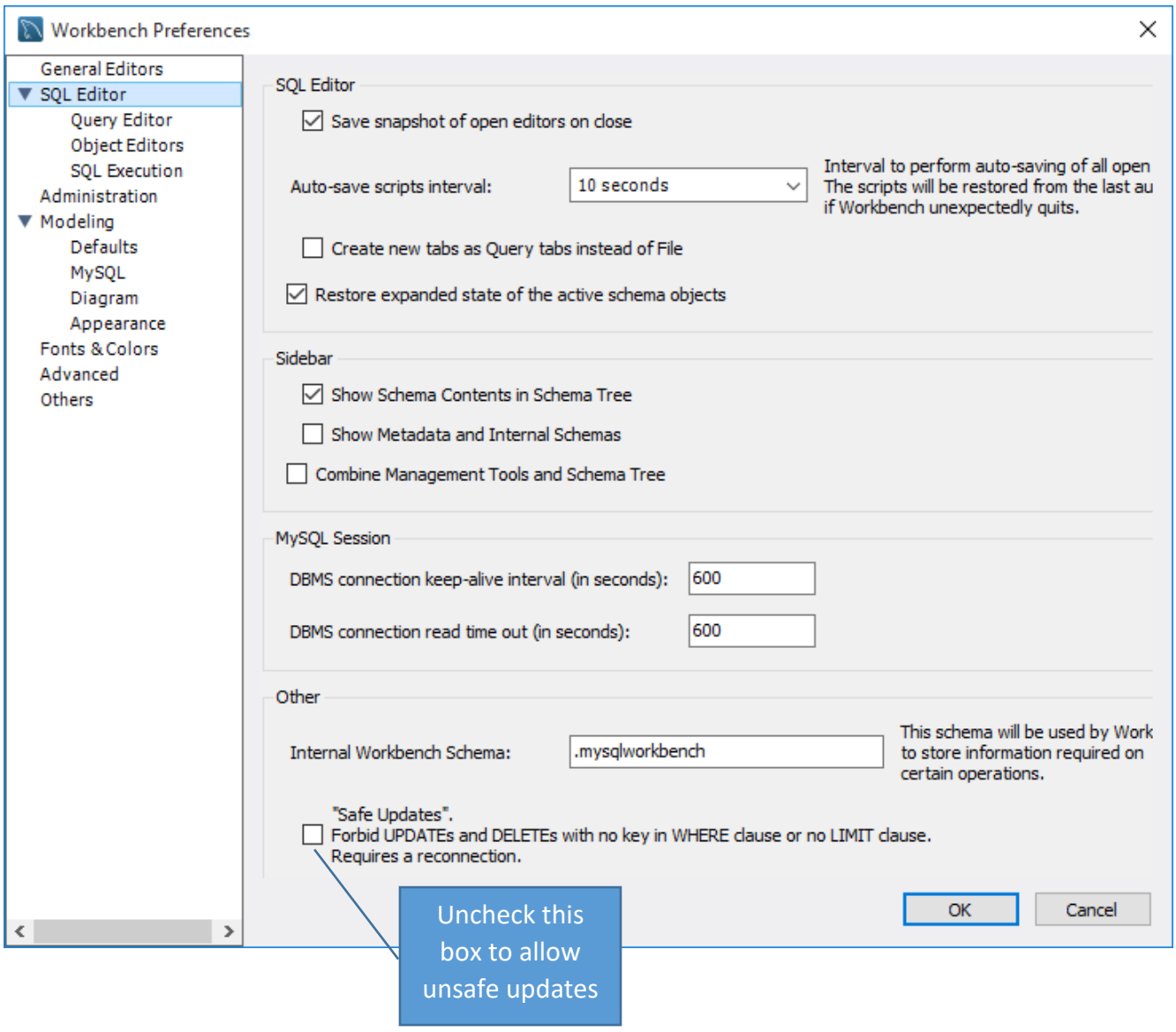
The “forward” method of the RequestDispatcher object forwards a request from a servlet to another resource (another servlet, a JSP file, or an HTML file) on the server.

```
HttpSession session = request.getSession();
session.setAttribute("message", message);
session.setAttribute("sqlStatement", sqlStatement);

RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher("/index.jsp");
dispatcher.forward(request, response);
```

Safe Update Mode in MySQL

Updates to tables using SQL statements without a key field in a WHERE clause or a LIMIT clause are generally considered unsafe. MySQL installations normally have safe mode turned on by default to prevent this sort of update from occurring. To enable non-safe updates you can set the server using Edit → Preferences → SQL Editor



You can also add to a script the following command: set sql_safe_updates=0;