teUpdate() становится неэффективным, и в данном случае лучше использовать схему batch-команд:

```
try {
   Employee[] employees = new Employee[10];
   PreparedStatement statement =
   con.prepareStatement("INSERT INTO employee VALUES
                                        (?,?,?,?,?)");
   for (int i = 0; i < employees.length; i++) {
         Employee currEmployee = employees[i];
         statement.setInt(1, currEmployee.getSSN());
         statement.setString(2, currEmployee.getName());
   statement.setDouble(3, currEmployee.getSalary());
   statement.setString(4,currEmployee.getHireDate());
         statement.setInt(5, currEmployee.getLoc_Id());
         statement.addBatch();
   }
   updateCounts = statement.executeBatch();
   } catch (BatchUpdateException e) {
         e.printStackTrace();
   }
```

Транзакции

При проектировании распределенных систем часто возникают ситуации, когда сбой в системе или какой-либо ее периферийной части может привести к потере информации или к финансовым потерям. Простейшим примером может служить пример с перечислением денег с одного счета на другой. Если сбой произошел в тот момент, когда операция снятия денег с одного счета уже произведена, а операция зачисления на другой счет еще не произведена, то система, позволяющая такие ситуации, должна быть признана не отвечающей требованиям заказчика. Или должны выполняться обе операции, или не выполняться вовсе. Такие две операции трактуют как одну и называют транзакцией.

Транзакцию (деловую операцию) определяют как единицу работы, обладающую свойствами ACID:

- Атомарность – две или более операций выполняются все или не выполняется ни одна. Успешно завершенные транзакции фиксируются, в случае неудачного завершения происходит откат всей транзакции.
- Согласованность – при возникновении сбоя система возвращается в состояние до начала неудавшейся транзакции. Если транзакция завершается успешно, то проверка согласованности удостоверяется в успешном завершении всех операций транзакции.
- Изолированность – во время выполнения транзакции все объекты-сущности, участвующие в ней, должны быть синхронизированы.
- Долговечность – все изменения, произведенные с данными во время транзакции, сохраняются, например, в базе данных. Это позволяет восстанавливать систему.

Для фиксации результатов работы SQL-операторов, логически выполняемых в рамках некоторой транзакции, используется SQL-оператор COMMIT. В API JDBC эта операция выполняется по умолчанию после каждого вызова методов **executeQuery()** и **executeUpdate()**. Если же необходимо сгруппировать запросы и только после этого выполнить операцию COMMIT, сначала вызывается метод **setAutoCommit(boolean param)** интерфейса **Connection** с параметром **false**, в результате выполнения которого текущее соединение с БД переходит в режим неавтоматического подтверждения операций. После этого выполнение любого запроса на изменение информации в таблицах базы данных не приведет к необратимым последствиям, пока операция COMMIT не будет выполнена непосредственно. Подтверждает выполнение SQL-запросов метод **commit()** интерфейса **Connection**, в результате действия которого все изменения таблицы производятся как одно логическое действие. Если же транзакция не выполнена, то методом **rollback()** отменяются действия всех запросов SQL, начиная от последнего вызова **commit()**. В следующем примере информация добавляется в таблицу в режиме действия транзакции, подтвердить или отменить действия которой можно, снимая или добавляя комментарий в строках вызова методов **commit()** и **rollback()**.

```
<!-- пример # 3 : вызов сервлета : index.jsp -->
<%@ page language="java" contentType="text/html; char-
set=windows-1251" pageEncoding="windows-1251"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional
//EN">
<html><head>
<meta http-equiv="Content-Type"
      content="text/html; charset=windows-1251">
<title>Simple Transaction Demo</title>
</head>
<body>
<form name="students" method="POST"
      action="SQLTransactionServlet">

      id:<br/>
      <input type="text" name="id" value=""><br/>
      Name:<br/>
      <input type="text" name="name" value=""><br/>

      Course:<br/>
      <select name="course">
        <option>Java SE 6
        <option>XML
        <option>Struts
      </select><br/>

      <input type="submit" value="Submit">
</form>
</body></html>
```

```
/* пример # 4 : выполнение транзакции : метод perform() сервлета
SQLTransactionServlet.java */
public void taskPerform(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {
 response.setContentType("text/html; charset=Cp1251");
 PrintWriter out = null;
 Connection cn = null;
 try {
      out = response.getWriter();
      String id = request.getParameter("id");
      String name = request.getParameter("name");
      String course = request.getParameter("course");
      out.print("ID студента: " + id + ", " + name +<br>");
            cn = getConnection();
            cn.setAutoCommit(false);
            Statement st = cn.createStatement();
      try {
            String upd;
            upd =
      "INSERT INTO student (id, name) VALUES ('"
                              + id  + "', '" + name + "')";
            st.executeUpdate(upd);
            out.print("Внесены данные в students: "
                              + id + ", " + name + "<br>");

            upd =
 "INSERT INTO course(id_student, name_course) VALUES('"
                              + id + "','" + course + "')";
            st.executeUpdate(upd);
            out.print("Внесены данные в course: " + id
                              + ", " + course + "<br>");

            cn.commit();// подтверждение
 out.print("<b>Данные внесены - транзакция завершена"
                                    + "</b><br>");
      } catch (SQLException e) {
            cn.rollback();// откат
            out.println("<b>Произведен откат транзакции:"
                        + e.getMessage()  + "</b>");
      } finally {
            if (cn != null)
            cn.close();
      }
 } catch (SQLException e) {
      out.println("<b>ошибка при закрытии соединения:"
                                    + e.getMessage());
 }
}
```

Если таблицы `student` и `course` базы данных **`db1`** до изменения выглядели, например, следующим образом,

id	name
71	Goncharenko

id_student	name_course
71	Java SE 6

Рис. 20.2. Таблицы до выполнения запроса

то после внесения изменений и их подтверждения они примут вид:

id	name
71	Goncharenko
83	Petrov

id_student	name_course
71	Java SE 6
83	XML

Рис. 20.3. Таблицы после подтверждения выполнения запросов

ID студента: 83, Petrov
Внесены данные в students: 83, Petrov
Внесены данные в course: 83, XML
Данные внесены - транзакция завершена

Приведенный пример в полной мере не отражает принципы транзакции, но демонстрирует способы ее поддержки методами языка Java.

Для транзакций существует несколько типов чтения:

- Грязное чтение (dirty reads) происходит, когда транзакциям разрешено видеть несохраненные изменения данных. Иными словами, изменения, сделанные в одной транзакции, видны вне ее до того, как она была сохранена. Если изменения не будут сохранены, то, вероятно, другие транзакции выполняли работу на основе некорректных данных;
- Непроверяющееся чтение (nonrepeatable reads) происходит, когда транзакция А читает строку, транзакция Б изменяет эту строку, транзакция А читает ту же строку и получает обновленные данные;
- Фантомное чтение (phantom reads) происходит, когда транзакция А считывает все строки, удовлетворяющие **`WHERE`**-условию, транзакция Б вставляет новую или удаляет одну из строк, которая удовлетворяет этому условию, транзакция А еще раз считывает все строки, удовлетворяющие **`WHERE`**-условию, уже вместе с новой строкой или недосчитавшись старой.

JDBC удовлетворяет четырем уровням изоляции транзакций, определенным в стандарте SQL:2003.

Уровни изоляции транзакций определены в виде констант интерфейса **`Connection`** (по возрастанию уровня ограничения):

- **`TRANSACTION_NONE`** – информирует о том, что драйвер не поддерживает транзакции;
- **`TRANSACTION_READ_UNCOMMITTED`** – позволяет транзакциям видеть несохраненные изменения данных, что разрешает грязное, непроверяющееся и фантомное чтения;
- **`TRANSACTION_READ_COMMITTED`** – означает, что любое изменение, сделанное в транзакции, не видно вне неё, пока она не сохранена. Это предотвращает грязное чтение, но разрешает непроверяющееся и фантомное;

- **TRANSACTION_REPEATABLE_READ** – запрещает грязное и непроверяющееся, но фантомное чтение разрешено;
- **TRANSACTION_SERIALIZABLE** – определяет, что грязное, непроверяющееся и фантомное чтения запрещены.

Метод **boolean supportsTransactionIsolationLevel(int level)** интерфейса **DatabaseMetaData** определяет, поддерживается ли заданный уровень изоляции транзакций.

В свою очередь, методы интерфейса **Connection** определяют доступ к уровню изоляции:

int getTransactionIsolation() – возвращает текущий уровень изоляции;

void setTransactionIsolation(int level) – устанавливает нужный уровень.

Точки сохранения

Точки сохранения дают дополнительный контроль над транзакциями. Установкой точки сохранения обозначается логическая точка внутри транзакции, которая может быть использована для отката данных. Таким образом, если произойдет ошибка, можно вызвать метод **rollback()** для отмены всех изменений, которые были сделаны после точки сохранения.

Метод **boolean supportsSavepoints()** интерфейса **DatabaseMetaData** используется для того, чтобы определить, поддерживает ли точки сохранения драйвер JDBC и сама СУБД.

Методы **setSavepoint(String name)** и **setSavepoint()** (оба возвращают объект **Savepoint**) интерфейса **Connection** используются для установки именованной или неименованной точки сохранения во время текущей транзакции. При этом новая транзакция будет начата, если в момент вызова **setSavepoint()** не будет активной транзакции.

```
/* пример # 5 : применение точек сохранения : SavepointServlet.java */
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;

public class SavepointServlet extends HttpServlet {
      protected void doGet(HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException, IOException {
                  performTask(req, resp);
            }
      protected void doPost(HttpServletRequest req,
                  HttpServletResponse resp)
            throws ServletException, IOException {
                  performTask(req, resp);
            }
```