PRIMER – JDBC MS SQL Server

URL za uspostavljanje konekcije:

```
jdbc:sqlserver://[serverName[\instanceName][:portNumber]][;property=value]
```

applicationIntent, applicationName, authenticationScheme, databaseName, database, disableStatementPooling, encrypt, failoverPartner, hostNameInCertificate, instanceName, integratedSecurity, lastUpdateCount, lockTimeout, loginTimeout, multiSubnetFailover, packetSize, password, portNumber, port, responseBuffering, selectMethod, sendStringParametersAsUnicode, sendTimeAsDatetime, serverName, server, trustServerCertificate, trustStore, trustStorePassword, userName, user, workstationID, xopenStates

Primeri:

```
jdbc:sqlserver://localhost:1433;databaseName=AdventureWorks;integratedSecurity=true;

jdbc:sqlserver://MyServer;loginTimeout=90;integratedSecurity=true;

jdbc:sqlserver://MyServer;applicationName=MYAPP.EXE;integratedSecurity=true;
```

Upotreba **sqljdbc_auth.dll**

```
-Djava.library.path=C:\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_<version>\enu\auth\x86
```

Upotreba IPv6

```
jdbc:sqlserver://;serverName=3ffe:8311:eeee:f70f:0:5eae:10.203.31.9\\instance1;integra
tedSecurity=true;
```

odnosno

```
Properties pro = new Properties();

pro.setProperty("serverName",
"serverName=3ffe:8311:eeee:f70f:0:5eae:10.203.31.9\\instance1");

Connection con = DriverManager.getConnection
                                ("jdbc:sqlserver://;integratedSecurity=true;", pro);
```

Primer fail-over:

```
import java.sql.*;

public class clientFailover {

   public static void main(String[] args) {

      // Create a variable for the connection string.
      String connectionUrl = "jdbc:sqlserver://serverA:1433;" +
         "databaseName=AdventureWorks;integratedSecurity=true;" +
         "failoverPartner=serverB";

      // Declare the JDBC objects.
      Connection con = null;
      Statement stmt = null;

      try {
         // Establish the connection to the principal server.
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
         con = DriverManager.getConnection(connectionUrl);
         System.out.println("Connected to the principal server.");

         // Note that if a failover of serverA occurs here, then an
         // exception will be thrown and the failover partner will
         // be used in the first catch block below.

         // Create and execute an SQL statement that inserts some data.
         stmt = con.createStatement();

         // Note that the following statement assumes that the
         // TestTable table has been created in the AdventureWorks
         // sample database.
         stmt.executeUpdate("INSERT INTO TestTable (Col2, Col3) VALUES ('a', 10)");
      }

      // Handle any errors that may have occurred.
      catch (SQLException se) {
         try {
            // The connection to the principal server failed,
            // try the mirror server which may now be the new
            // principal server.
            System.out.println("Connection to principal server failed, " +
            "trying the mirror server.");
            con = DriverManager.getConnection(connectionUrl);
            System.out.println("Connected to the new principal server.");
            stmt = con.createStatement();
            stmt.executeUpdate("INSERT INTO TestTable (Col2, Col3) VALUES ('a', 10)");
         }
         catch (Exception e) {
            e.printStackTrace();
         }
      }
      catch (Exception e) {
         e.printStackTrace();
      }
      // Close the JDBC objects.
      finally {
         if (stmt != null) try { stmt.close(); } catch(Exception e) {}
         if (con != null) try { con.close(); } catch(Exception e) {}
      }
   }
}
```

SQL Server Types	JDBC Types (java.sql.Types)	Java Language Types
bigint	BIGINT	long
binary	BINARY	byte[]
bit	BIT	boolean
char	CHAR	String
date	DATE	java.sql.Date
datetime	TIMESTAMP	java.sql.Timestamp
datetime2	TIMESTAMP	java.sql.Timestamp
datetimeoffset	microsoft.sql.Types.DATETIMEOFFSET	microsoft.sql.DateTimeOffset
decimal	DECIMAL	java.math.BigDecimal
float	DOUBLE	double
image	LONGVARBINARY	byte[]
int	INTEGER	int
money	DECIMAL	java.math.BigDecimal
nchar	CHAR NCHAR (Java SE 6.0)	String
ntext	LONGVARCHAR LONGNVARCHAR (Java SE 6.0)	String
numeric	NUMERIC	java.math.BigDecimal
nvarchar	VARCHAR NVARCHAR (Java SE 6.0)	String
nvarchar(max)	VARCHAR NVARCHAR (Java SE 6.0)	String
real	REAL	float
smalldatetime	TIMESTAMP	java.sql.Timestamp
smallint	SMALLINT	short
smallmoney	DECIMAL	java.math.BigDecimal
text	LONGVARCHAR	String
time	TIME	java.sql.Time
timestamp	BINARY	byte[]
tinyint	TINYINT	short
udt	VARBINARY	byte[]
uniqueidentifier	CHAR	String
varbinary	VARBINARY	byte[]
varbinary(max)	VARBINARY	byte[]
varchar	VARCHAR	String
varchar(max)	VARCHAR	String
xml	LONGVARCHAR LONGNVARCHAR (Java SE 6.0)	String SQLXML

Primer dohvatanja podataka kao string:

```
String SQL = "SELECT TOP 10 * FROM Person.Contact";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(SQL);

while (rs.next()) {
   System.out.println(rs.getString(4) + " " + rs.getString(6));
}
rs.close();
stmt.close();
```

Primer dohvatanja podataka na osnovu informacije o tipu podataka:

```
ResultSet rs = stmt.executeQuery("SELECT lname, job_id FROM employee
   WHERE (lname = 'Brown')");
rs.next();
short empJobID = rs.getShort("job_id");
rs.close();
stmt.close();
```

Primer promene podataka:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT lname, job_id FROM employee
   WHERE (lname = 'Brown')");
rs.next();
short empJobID = rs.getInt(2);
empJobID++;
rs.first();
rs.updateInt(2, empJobID);
rs.updateRow();
rs.close();
stmt.close();
```

Primer promene podataka kroz parametrizovani upit:

```
PreparedStatement pstmt = con.prepareStatement("UPDATE employee SET
   fname = ? WHERE (lname = 'Brown')");
String first = "Bob";
pstmt.setString(1, first);
int rowCount = pstmt.executeUpdate();
pstmt.close();
```

Primer prosledjivanja parametara proceduri:

```
CallableStatement cstmt = con.prepareCall("{call employee_jobid(?)}");
String lname = "Brown";
cstmt.setString(1, lname);
Resultset rs = cstmt.executeQuery();
rs.close();
cstmt.close();
```

Primer dohvatanja parametara iz procedure:

```
CallableStatement cstmt = con.prepareCall("{call employee_jobid (?, ?)}");
cstmt.registerOutParameter(2, java.sql.Types.SMALLINT);
String lname = "Brown";
cstmt.setString(1, lname);
Resultset rs = cstmt.executeQuery();
short empJobID = cstmt.getShort(2);
rs.close();
cstmt.close();
```

Upotreba velikih podataka (preko 8KB)
varchar, nvarchar, varbinary mogu biti 2^{31} bajtova

SQL Server Types	JDBC Types (java.sql.Types)	Java Language Types
varbinary(max) image	LONGVARBINARY	byte[] (default), Blob, InputStream, String
text varchar(max)	LONGVARCHAR	String (default), Clob, InputStream
ntext nvarchar(max)	LONGVARCHAR LONGNVARCHAR (Java SE 6.0)	String (default), Clob, NClob (Java SE 6.0)
xml	LONGVARCHAR SQLXML (Java SE 6.0)	String (default), InputStream, Clob, byte[],Blob, SQLXML (Java SE 6.0)
Udt1	VARBINARY	String (default), byte[], InputStream

Primer dohvatanja velikog podatka (**`varchar(max)`**):

```
ResultSet rs = stmt.executeQuery("SELECT TOP 1 * FROM Test1");
rs.next();
Reader reader = rs.getCharacterStream(2);
```

Primer dohvatanja velikog podatka (**`varbinary(max)`**):

```
ResultSet rs = stmt.executeQuery("SELECT photo FROM mypics");
rs.next();
InputStream is = rs.getBinaryStream(2);
```

Ili:

```
ResultSet rs = stmt.executeQuery("SELECT photo FROM mypics");
rs.next();
byte [] b = rs.getBytes(2);
```

Primer promene velikog podatka (**`varchar(max)`**):

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO test1 (c1_id, c2_vcmax)
VALUES (?, ?)");
pstmt.setInt(1, 1);
pstmt.setString(2, htmlStr);
pstmt.executeUpdate();
```

Primer promene velikog podatka (**`varbinary(max)`**):

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO test1 (Col1, Col2)
VALUES(?,?)");
File inputFile = new File("CLOBFile20mb.jpg");
FileInputStream inStream = new FileInputStream(inputFile);
int id = 1;
pstmt.setInt(1,id);
pstmt.setBinaryStream(2, inStream);
pstmt.executeUpdate();
inStream.close();
```

Primer promene reči u velikom tekstualnom podatku (**`clob`**):

```
String SQL = "SELECT * FROM test1;";
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery(SQL);
rs.next();

Clob clob = rs.getClob(2);
long pos = clob.position("dog", 1);
clob.setString(pos, "cat");
rs.updateClob(2, clob);
rs.updateRow();
```

Primer upotrebe procedure bez parametara:

```
CREATE PROCEDURE GetContactFormalNames
AS
BEGIN
   SELECT TOP 10 Title + ' ' + FirstName + ' ' + LastName AS FormalName
   FROM Person.Contact
END

-------------------------------------------------------------

public static void executeSprocNoParams(Connection con) {
   try {
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("{call dbo.GetContactFormalNames}");

      while (rs.next()) {
         System.out.println(rs.getString("FormalName"));
      }
      rs.close();
      stmt.close();
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Primer upotrebe procedure sa ulaznim parametrima:

```
public static void executeSprocInParams(Connection con) {
   try {
      PreparedStatement pstmt = con.prepareStatement("{call
dbo.uspGetEmployeeManagers(?)}");
      pstmt.setInt(1, 50);
      ResultSet rs = pstmt.executeQuery();

      while (rs.next()) {
         System.out.println("EMPLOYEE:");
         System.out.println(rs.getString("LastName") + ", " +
rs.getString("FirstName"));
         System.out.println("MANAGER:");
         System.out.println(rs.getString("ManagerLastName") + ", " +
rs.getString("ManagerFirstName"));
         System.out.println();
      }
      rs.close();
      pstmt.close();
   }

   catch (Exception e) {
      e.printStackTrace();
    }
}
```

Primer upotrebe procedure sa izlaznim parametrima:

```
CREATE PROCEDURE GetImmediateManager
   @employeeID INT,
   @managerID INT OUTPUT
AS
BEGIN
   SELECT @managerID = ManagerID
   FROM HumanResources.Employee
   WHERE EmployeeID = @employeeID
END

-----------------------------------------------------------------

public static void executeStoredProcedure(Connection con) {
   try {
      CallableStatement cstmt = con.prepareCall("{call
dbo.GetImmediateManager(?, ?)}");
      cstmt.setInt(1, 5);
      cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
      cstmt.execute();
      System.out.println("MANAGER ID: " + cstmt.getInt(2));
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Ili pristup po nazivu parametra:

```
public static void executeStoredProcedure(Connection con) {
   try {
      CallableStatement cstmt = con.prepareCall("{call
dbo.GetImmediateManager(?, ?)}");
      cstmt.setInt("employeeID", 5);
      cstmt.registerOutParameter("managerID", java.sql.Types.INTEGER);
      cstmt.execute();
      System.out.println("MANAGER ID: " + cstmt.getInt("managerID"));
      cstmt.close();
   }
   catch (Exception e) {
      e.printStackTrace();
   }
```

Primer upotrebe procedure sa izlaznim statusnim parametrom:

```
CREATE PROCEDURE CheckContactCity
   (@cityName CHAR(50))
AS
BEGIN
   IF ((SELECT COUNT(*)
   FROM Person.Address
   WHERE City = @cityName) > 1)
   RETURN 1
ELSE
   RETURN 0
END

-----------------------------------------------------------------------

public static void executeStoredProcedure(Connection con) {
   try {
      CallableStatement cstmt = con.prepareCall("{? = call dbo.CheckContactCity(?)}");
      cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
      cstmt.setString(2, "Atlanta");
      cstmt.execute();
      System.out.println("RETURN STATUS: " + cstmt.getInt(1));
      cstmt.close();
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Primer upotrebe procedure sa izlaznom informacijom o broju ažuriranih redova:

```
CREATE TABLE TestTable
   (Col1 int IDENTITY,
    Col2 varchar(50),
    Col3 int);

CREATE PROCEDURE UpdateTestTable
   @Col2 varchar(50),
   @Col3 int
AS
BEGIN
   UPDATE TestTable
   SET Col2 = @Col2, Col3 = @Col3
END;
INSERT INTO dbo.TestTable (Col2, Col3) VALUES ('b', 10);

-----------------------------------------------------------------

public static void executeUpdateStoredProcedure(Connection con) {
   try {
      CallableStatement cstmt = con.prepareCall("{call dbo.UpdateTestTable(?, ?)}");
      cstmt.setString(1, "A");
      cstmt.setInt(2, 100);
      cstmt.execute();
      int count = cstmt.getUpdateCount();
      cstmt.close();

      System.out.println("ROWS AFFECTED: " + count);
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Primer upotrebe upita koji vraća veći broj rezultata:

```
public static void executeStatement(Connection con) {
   try {
      String SQL = "SELECT TOP 10 * FROM Person.Contact; " +
                   "SELECT TOP 20 * FROM Person.Contact";
      Statement stmt = con.createStatement();
      boolean results = stmt.execute(SQL);
      int rsCount = 0;

      //Loop through the available result sets.
     do {
        if(results) {
           ResultSet rs = stmt.getResultSet();
           rsCount++;

           //Show data from the result set.
           System.out.println("RESULT SET #" + rsCount);
           while (rs.next()) {
              System.out.println(rs.getString("LastName") + ", " +
rs.getString("FirstName"));
           }
           rs.close();
        }
        System.out.println();
        results = stmt.getMoreResults();
        } while(results);
      stmt.close();
      }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Primer upotrebe kompleksnih iskaza:

```
public static void executeComplexStatement(Connection con) {
   try {
      String sqlStringWithUnknownResults = "{call
dbo.uspGetEmployeeManagers(50)};SELECT TOP 10 * FROM Person.Contact";
      Statement stmt = con.createStatement();
      boolean results = stmt.execute(sqlStringWithUnknownResults);
      int count = 0;
      do {
         if (results) {
            ResultSet rs = stmt.getResultSet();
            System.out.println("Result set data displayed here.");
            rs.close();
         } else {
            count = stmt.getUpdateCount();
            if (count >= 0) {
               System.out.println("DDL or update data displayed here.");
            } else {
               System.out.println("No more results to process.");
            }
         }
         results = stmt.getMoreResults();
      } while (results || count != -1);
      stmt.close();
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Primer upotrebe sa autogenerišučim primarnim ključem:

```
CREATE TABLE TestTable
   (Col1 int IDENTITY,
    Col2 varchar(50),
    Col3 int);

-----------------------------------------------------------------

public static void executeInsertWithKeys(Connection con) {
   try {
      String SQL = "INSERT INTO TestTable (Col2, Col3) VALUES ('S', 50)";
      Statement stmt = con.createStatement();
      int count = stmt.executeUpdate(SQL, Statement.RETURN_GENERATED_KEYS);
      ResultSet rs = stmt.getGeneratedKeys();

      ResultSetMetaData rsmd = rs.getMetaData();
      int columnCount = rsmd.getColumnCount();
      if (rs.next()) {
         do {
            for (int i=1; i<=columnCount; i++) {
               String key = rs.getString(i);
               System.out.println("KEY " + i + " = " + key);
            }
         } while(rs.next());
      }
      else {
         System.out.println("NO KEYS WERE GENERATED.");
      }
      rs.close();
      stmt.close();
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Primer upotrebe sa paketnom obradom:

```
CREATE TABLE TestTable
   (Col1 int IDENTITY,
    Col2 varchar(50),
    Col3 int);

-----------------------------------------------------------------------

public static void executeBatchUpdate(Connection con) {
   try {
      Statement stmt = con.createStatement();
      stmt.addBatch("INSERT INTO TestTable (Col2, Col3) VALUES ('X', 100)");
      stmt.addBatch("INSERT INTO TestTable (Col2, Col3) VALUES ('Y', 200)");
      stmt.addBatch("INSERT INTO TestTable (Col2, Col3) VALUES ('Z', 300)");
      int[] updateCounts = stmt.executeBatch();
      stmt.close();
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Result Set (Cursor) Type	Characteristics
TYPE_FORWARD_ONLY (CONCUR_READ_ONLY)	Forward-only, read-only
TYPE_FORWARD_ONLY (CONCUR_READ_ONLY)	Forward-only, read-only
TYPE_FORWARD_ONLY (CONCUR_READ_ONLY)	Forward-only, read-only
TYPE_FORWARD_ONLY (CONCUR_UPDATABLE)	Forward-only, updatable
TYPE_SCROLL_INSENSITIVE	Scrollable, not updateable. External row updates, inserts, and deletes are not visible.
TYPE_SCROLL_SENSITIVE (CONCUR_READ_ONLY)	Scrollable, read-only. External row updates are visible, and deletes appear as missing data. External row inserts are not visible.
TYPE_SCROLL_SENSITIVE (CONCUR_UPDATABLE, CONCUR_SS_SCROLL_LOCKS, CONCUR_SS_OPTIMISTIC_CC, CONCUR_SS_OPTIMISTIC_CCVAL)	Scrollable, updatable. External and internal row updates are visible, and deletes appear as missing data; inserts are not visible.
TYPE_SS_DIRECT_FORWARD_ONLY	Forward-only, read-only
TYPE_SS_SERVER_CURSOR_FORWARD_ONLY	Forward-only
TYPE_SS_SCROLL_STATIC	Other users’ updates are not reflected.
TYPE_SS_SCROLL_KEYSET (CONCUR_READ_ONLY)	Scrollable, read-only. External row updates are visible, and deletes appear as missing data. External row inserts are not visible.
TYPE_SS_SCROLL_KEYSET (CONCUR_UPDATABLE, CONCUR_SS_SCROLL_LOCKS, CONCUR_SS_OPTIMISTIC_CC, CONCUR_SS_OPTIMISTIC_CCVAL)	Scrollable, updatable. External and internal row updates are visible, and deletes appear as missing data; inserts are not visible.
TYPE_SS_SCROLL_DYNAMIC (CONCUR_READ_ONLY)	Scrollable, read-only. External row updates and inserts are visible, and deletes appear as transient missing data in the current fetch buffer.
TYPE_SS_SCROLL_DYNAMIC (CONCUR_UPDATABLE, CONCUR_SS_SCROLL_LOCKS, CONCUR_SS_OPTIMISTIC_CC, CONCUR_SS_OPTIMISTIC_CCVAL)	Scrollable, updatable. External and internal row updates and inserts are visible, and deletes appear as transient missing data in the current fetch buffer.

Primer upotrebe transakcija:

```
public static void executeTransaction(Connection con) {
   try {
      //Switch to manual transaction mode by setting
      //autocommit to false. Note that this starts the first
      //manual transaction.
      con.setAutoCommit(false);
      Statement stmt = con.createStatement();
      stmt.executeUpdate("INSERT INTO Production.ScrapReason(Name) VALUES('Wrong
size')");
      stmt.executeUpdate("INSERT INTO Production.ScrapReason(Name) VALUES('Wrong
color')");
      con.commit(); //This commits the transaction and starts a new one.
      stmt.close(); //This turns off the transaction.
      System.out.println("Transaction succeeded. Both records were written to the
database.");
   }
   catch (SQLException ex) {
      ex.printStackTrace();
      try {
         System.out.println("Transaction failed.");
         con.rollback();
      }
      catch (SQLException se) {
         se.printStackTrace();
      }
   }
}
```

Isolation Level	Dirty Read	Non Repeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

```
con.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);

con.setTransactionIsolation(SQLServerConnection.TRANSACTION_SNAPSHOT);
```

Primer upotrebe delimičnog opravka transakcije:

```
public static void executeTransaction(Connection con) {
   try {
      con.setAutoCommit(false);
      Statement stmt = con.createStatement();
      stmt.executeUpdate("INSERT INTO Production.ScrapReason(Name) VALUES('Correct
width')");
      Savepoint save = con.setSavepoint();
      stmt.executeUpdate("INSERT INTO Production.ScrapReason(Name) VALUES('Wrong
width')");
      con.rollback(save);
      con.commit();
      stmt.close();
      System.out.println("Transaction succeeded.");
   }
   catch (SQLException ex) {
      ex.printStackTrace();
      try {
         System.out.println("Transaction failed.");
         con.rollback();
      }
      catch (SQLException se) {
         se.printStackTrace();
      }
   }
}
```

Primer upotrebe metapodataka:

```
public static void getDatabaseMetaData(Connection con) {
   try {
      DatabaseMetaData dbmd = con.getMetaData();
      System.out.println("dbmd:driver version = " + dbmd.getDriverVersion());
      System.out.println("dbmd:driver name = " + dbmd.getDriverName());
      System.out.println("db name = " + dbmd.getDatabaseProductName());
      System.out.println("db ver = " + dbmd.getDatabaseProductVersion());
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}

public static void getResultSetMetaData(Connection con) {
   try {
      String SQL = "SELECT TOP 10 * FROM Person.Contact";
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(SQL);
      ResultSetMetaData rsmd = rs.getMetaData();

      // Display the column name and type.
      int cols = rsmd.getColumnCount();
      for (int i = 1; i <= cols; i++) {
         System.out.println("NAME: " + rsmd.getColumnName(i) + " " + "TYPE: " +
rsmd.getColumnTypeName(i));
      }
      rs.close();
      stmt.close();
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}

public static void getParameterMetaData(Connection con) {
   try {
      CallableStatement cstmt = con.prepareCall("{call
HumanResources.uspUpdateEmployeeHireInfo(?, ?, ?, ?, ?)}");
      ParameterMetaData pmd = cstmt.getParameterMetaData();
      int count = pmd.getParameterCount();
      for (int i = 1; i <= count; i++) {
         System.out.println("TYPE: " + pmd.getParameterTypeName(i) + " MODE: " +
pmd.getParameterMode(i));
      }
      cstmt.close();
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

Primer 1:

```
CREATE TABLE DataTypesTable
   (Col1 int IDENTITY,
    Col2 char,
    Col3 varchar(50),
    Col4 bit,
    Col5 decimal(18, 2),
    Col6 money,
    Col7 datetime,
    Col8 date,
    Col9 time,
    Col10 datetime2,
    Col11 datetimeoffset
    );

INSERT INTO DataTypesTable
VALUES ('A', 'Some text.', 0, 15.25, 10.00, '01/01/2006 23:59:59.991', '01/01/2006',
'23:59:59', '01/01/2006 23:59:59.12345', '01/01/2006 23:59:59.12345 -1:00')

-----------------------------------------------------------------

import java.sql.*;

import com.microsoft.sqlserver.jdbc.SQLServerResultSet;
import microsoft.sql.DateTimeOffset;

public class basicDT {
   public static void main(String[] args) {

      // Create a variable for the connection string.
      String connectionUrl =
"jdbc:sqlserver://localhost:1433;databaseName=AdventureWorks;integratedSecurity=true;";

      // Declare the JDBC objects.
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;

      try {
         // Establish the connection.
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
         con = DriverManager.getConnection(connectionUrl);

         // Create and execute an SQL statement that returns some data
         // and display it.
         String SQL = "SELECT * FROM DataTypesTable";
         stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_UPDATABLE);
         rs = stmt.executeQuery(SQL);
         rs.next();
         displayRow("ORIGINAL DATA", rs);

         // Update the data in the result set.
         rs.updateString(2, "B");
         rs.updateString(3, "Some updated text.");
         rs.updateBoolean(4, true);
         rs.updateDouble(5, 77.89);
         rs.updateDouble(6, 1000.01);
         long timeInMillis = System.currentTimeMillis();
         Timestamp ts = new Timestamp(timeInMillis);
         rs.updateTimestamp(7, ts);
         rs.updateDate(8, new Date(timeInMillis));
         rs.updateTime(9, new Time(timeInMillis));
```

```
         rs.updateTimestamp(10, ts);

         //-480 indicates GMT - 8:00 hrs
         ((SQLServerResultSet)rs).updateDateTimeOffset(11, DateTimeOffset.valueOf(ts,
-480));

         rs.updateRow();

         // Get the updated data from the database and display it.
         rs = stmt.executeQuery(SQL);
         rs.next();
         displayRow("UPDATED DATA", rs);
      }

      // Handle any errors that may have occurred.
      catch (Exception e) {
         e.printStackTrace();
      }

      finally {
         if (rs != null)
         try {
         rs.close();
         }
         catch(Exception e) {}

         if (stmt != null)
         try { stmt.close();
         }
         catch(Exception e) {}

         if (con != null)
         try {
         con.close();
         }
         catch(Exception e) {}
      }
   }

   private static void displayRow(String title, ResultSet rs) {
      try {
         System.out.println(title);
         System.out.println(rs.getInt(1) + " , " +  // SQL integer type.
               rs.getString(2) + " , " +            // SQL char type.
               rs.getString(3) + " , " +            // SQL varchar type.
               rs.getBoolean(4) + " , " +           // SQL bit type.
               rs.getDouble(5) + " , " +            // SQL decimal type.
               rs.getDouble(6) + " , " +            // SQL money type.
               rs.getTimestamp(7) + " , " +            // SQL datetime type.
               rs.getDate(8) + " , " +              // SQL date type.
               rs.getTime(9) + " , " +              // SQL time type.
               rs.getTimestamp(10) + " , " +            // SQL datetime2 type.
               ((SQLServerResultSet)rs).getDateTimeOffset(11)); // SQL datetimeoffset
type.

         System.out.println();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Primer 2 - XML:

```
import java.sql.*;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;

import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.sax.SAXResult;
import javax.xml.transform.sax.SAXTransformerFactory;

import org.xml.sax.*;

public class sqlxmlExample {

   public static void main(String[] args) {

      // Create a variable for the connection string.
      String connectionUrl = "jdbc:sqlserver://localhost:1433;" +
            "databaseName=AdventureWorks;integratedSecurity=true;";

      // Declare the JDBC objects.
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;

      try {
         // Establish the connection.
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
         con = DriverManager.getConnection(connectionUrl);

         // Create initial sample data.
         createSampleTables(con);

         // The showGetters method demonstrates how to parse the data in the
         // SQLXML object by using the SAX, ContentHandler and XMLReader.
         showGetters(con);

         // The showSetters method demonstrates how to set the xml column
         // by using the SAX, ContentHandler, and ResultSet.
         showSetters(con);

         // The showTransformer method demonstrates how to get an XML data
         // from one table and insert that XML data to another table
         // by using the SAX and the Transformer.
         showTransformer(con);
      }
      // Handle any errors that may have occurred.
      catch (Exception e) {
         e.printStackTrace();
      }
      finally {
         if (rs != null) try { rs.close(); } catch(Exception e) {}
         if (stmt != null) try { stmt.close(); } catch(Exception e) {}
         if (con != null) try { con.close(); } catch(Exception e) {}
      }
   }

   private static void showGetters(Connection con) {

      try {
          // Create an instance of the custom content handler.
          ExampleContentHandler myHandler = new ExampleContentHandler();

          // Create and execute an SQL statement that returns a
```

```
          // set of data.
          String SQL = "SELECT * FROM TestTable1";
          Statement stmt = con.createStatement();
          ResultSet rs = stmt.executeQuery(SQL);

          rs.next();

          SQLXML xmlSource = rs.getSQLXML("Col3");

          // Send SAX events to the custom content handler.
          SAXSource sxSource =  xmlSource.getSource(SAXSource.class);
          XMLReader xmlReader = sxSource.getXMLReader();
          xmlReader.setContentHandler(myHandler);

            System.out.println("showGetters method: Parse an XML data in TestTable1 =>
");
          xmlReader.parse(sxSource.getInputSource());

      } catch (Exception e) {
          e.printStackTrace();
      }
   }

   private static void showSetters(Connection con) {

      try {
         // Create and execute an SQL statement, retrieving an updatable result set.
         String SQL = "SELECT * FROM TestTable1;";
         Statement stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_UPDATABLE);
         ResultSet rs = stmt.executeQuery(SQL);

         // Create an empty SQLXML object.
         SQLXML sqlxml = con.createSQLXML();

         // Set the result value from SAX events.
         SAXResult sxResult = sqlxml.setResult(SAXResult.class);
         ContentHandler myHandler = sxResult.getHandler();

         // Set the XML elements and attributes into the result.
         myHandler.startDocument();
         myHandler.startElement(null, "contact", "contact", null);
         myHandler.startElement(null, "name", "name", null);
         myHandler.endElement(null, "name", "name");
         myHandler.startElement(null, "phone", "phone", null);
         myHandler.endElement(null, "phone", "phone");
         myHandler.endElement(null, "contact", "contact");
         myHandler.endDocument();

         // Update the data in the result set.
         rs.moveToInsertRow();
         rs.updateString("Col2", "C");
         rs.updateSQLXML("Col3", sqlxml);
         rs.insertRow();

         // Display the data.
         System.out.println("showSetters method: Display data in TestTable1 => ");
         while (rs.next()) {
           System.out.println(rs.getString("Col1") + " : " + rs.getString("Col2"));
           SQLXML xml = rs.getSQLXML("Col3");
           System.out.println("XML column : " + xml.getString());
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
```

```
   }

   private static void showTransformer(Connection con) {

      try {
          // Create and execute an SQL statement that returns a
          // set of data.
          String SQL = "SELECT * FROM TestTable1";
          Statement stmt = con.createStatement();
          ResultSet rs = stmt.executeQuery(SQL);

          rs.next();

          // Get the value of the source SQLXML object from the database.
          SQLXML xmlSource = rs.getSQLXML("Col3");

          // Get a Source to read the XML data.
          SAXSource sxSource =  xmlSource.getSource(SAXSource.class);

         // Create a destination SQLXML object without any data.
         SQLXML xmlDest = con.createSQLXML();

         // Get a Result to write the XML data.
         SAXResult sxResult = xmlDest.setResult(SAXResult.class);

         // Transform the Source to a Result by using the identity transform.
         SAXTransformerFactory stf = (SAXTransformerFactory)
TransformerFactory.newInstance();
         Transformer identity = stf.newTransformer();
         identity.transform(sxSource, sxResult);

         // Insert the destination SQLXML object into the database.
         PreparedStatement psmt =
           con.prepareStatement(
              "INSERT INTO TestTable2" + " (Col2, Col3, Col4, Col5) VALUES
(?, ?, ?, ?)");
         psmt.setString(1, "A");
         psmt.setString(2, "Test data");
         psmt.setInt(3, 123);
         psmt.setSQLXML(4, xmlDest);
         psmt.execute();

         // Execute the query and display the data.
         SQL = "SELECT * FROM TestTable2";
         stmt = con.createStatement();
         rs = stmt.executeQuery(SQL);

         System.out.println("showTransformer method : Display data in TestTable2 => ");
         while (rs.next()) {
            System.out.println(rs.getString("Col1") + " : " + rs.getString("Col2"));
            System.out.println(rs.getString("Col3") + " : " + rs.getInt("Col4"));

            SQLXML xml = rs.getSQLXML("Col5");
            System.out.println("XML column : " + xml.getString());
         }
     } catch (Exception e) {
        e.printStackTrace();
     }
   }

   private static void createSampleTables(Connection con) {

      try {
         Statement stmt = con.createStatement();
```

```
          // Drop the tables.
         stmt.executeUpdate("if exists (select * from sys.objects where name =
'TestTable1')" +
                "drop table TestTable1" );

          stmt.executeUpdate("if exists (select * from sys.objects where name =
'TestTable2')" +
                "drop table TestTable2" );

          // Create empty tables.
          stmt.execute("CREATE TABLE TestTable1 (Col1 int IDENTITY, Col2 char, Col3
xml)");
          stmt.execute("CREATE TABLE TestTable2 (Col1 int IDENTITY, Col2 char, Col3
varchar(50), Col4 int, Col5 xml)");

          // Insert two rows to the TestTable1.
         String row1 = "<contact><name>Contact Name 1</name><phone>XXX-XXX-
XXXX</phone></contact>";
         String row2 = "<contact><name>Contact Name 2</name><phone>YYY-YYY-
YYYY</phone></contact>";

         stmt.executeUpdate("insert into TestTable1" + " (Col2, Col3) values('A', '" +
row1 +"')");
         stmt.executeUpdate("insert into TestTable1" + " (Col2, Col3) values('B', '" +
row2 +"')");

       } catch (Exception e) {
          e.printStackTrace();
       }
   }
}

class ExampleContentHandler implements ContentHandler {

  public void startElement(String namespaceURI, String localName, String qName,
Attributes atts)
  throws SAXException {
       System.out.println("startElement method: localName => " + localName);
  }
  public void characters(char[] text, int start, int length) throws SAXException {
       System.out.println("characters method");
  }
  public void endElement(String namespaceURI, String localName, String qName) throws
SAXException {
     System.out.println("endElement method: localName => " + localName);
  }
  public void setDocumentLocator(Locator locator) {
       System.out.println("setDocumentLocator method");
  }
  public void startDocument() throws SAXException {
       System.out.println("startDocument method");
  }
  public void endDocument() throws SAXException {
       System.out.println("endDocument method");
  }
  public void startPrefixMapping(String prefix, String uri) throws SAXException {
       System.out.println("startPrefixMapping method: prefix => " + prefix);
  }
  public void endPrefixMapping(String prefix) throws SAXException {
       System.out.println("endPrefixMapping method: prefix => " + prefix);
  }
  public void skippedEntity(String name) throws SAXException {
       System.out.println("skippedEntity method: name => " + name);
  }
```

```
  public void ignorableWhitespace(char[] text, int start, int length) throws
SAXException {
        System.out.println("ignorableWhiteSpace method");
  }
  public void processingInstruction(String target, String data) throws SAXException {
        System.out.println("processingInstruction method: target => " + target);
  }
}
```

Primer 3:

```
import java.sql.*;

public class updateRS {

   public static void main(String[] args) {

      // Create a variable for the connection string.
      String connectionUrl = "jdbc:sqlserver://localhost:1433;" +
            "databaseName=AdventureWorks;integratedSecurity=true;";

      // Declare the JDBC objects.
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;

      try {

         // Establish the connection.
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
         con = DriverManager.getConnection(connectionUrl);

         // Create and execute an SQL statement, retrieving an updateable result set.
         String SQL = "SELECT * FROM HumanResources.Department;";
         stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
         rs = stmt.executeQuery(SQL);

         // Insert a row of data.
         rs.moveToInsertRow();
         rs.updateString("Name", "Accounting");
         rs.updateString("GroupName", "Executive General and Administration");
         rs.updateString("ModifiedDate", "08/01/2006");
         rs.insertRow();

         // Retrieve the inserted row of data and display it.
         SQL = "SELECT * FROM HumanResources.Department WHERE Name = 'Accounting';";
         rs = stmt.executeQuery(SQL);
         displayRow("ADDED ROW", rs);

         // Update the row of data.
         rs.first();
         rs.updateString("GroupName", "Finance");
         rs.updateRow();

         // Retrieve the updated row of data and display it.
         rs = stmt.executeQuery(SQL);
         displayRow("UPDATED ROW", rs);

         // Delete the row of data.
         rs.first();
         rs.deleteRow();
         System.out.println("ROW DELETED");
      }

      // Handle any errors that may have occurred.
      catch (Exception e) {
         e.printStackTrace();
      }

      finally {
         if (rs != null) try { rs.close(); } catch(Exception e) {}
         if (stmt != null) try { stmt.close(); } catch(Exception e) {}
```

```
         if (con != null) try { con.close(); } catch(Exception e) {}
      }
   }

   private static void displayRow(String title, ResultSet rs) {
      try {
         System.out.println(title);
         while (rs.next()) {
            System.out.println(rs.getString("Name") + " : " +
rs.getString("GroupName"));
            System.out.println();
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Primer 4:

```
import java.sql.*;
import com.microsoft.sqlserver.jdbc.SQLServerResultSet;

public class cacheRS {

   public static void main(String[] args) {

      // Create a variable for the connection string.
      String connectionUrl = "jdbc:sqlserver://localhost:1433;" +
            "databaseName=AdventureWorks;integratedSecurity=true;";

      // Declare the JDBC objects.
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;

      try {

         // Establish the connection.
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
         con = DriverManager.getConnection(connectionUrl);

         // Create and execute an SQL statement that returns a large
         // set of data and then display it.
         String SQL = "SELECT * FROM Sales.SalesOrderDetail;";
         stmt =
con.createStatement(SQLServerResultSet.TYPE_SS_SERVER_CURSOR_FORWARD_ONLY, +
               SQLServerResultSet.CONCUR_READ_ONLY);

         // Perform a fetch for every row in the result set.
         rs = stmt.executeQuery(SQL);
         timerTest(1, rs);
         rs.close();

         // Perform a fetch for every tenth row in the result set.
         rs = stmt.executeQuery(SQL);
         timerTest(10, rs);
         rs.close();

         // Perform a fetch for every 100th row in the result set.
         rs = stmt.executeQuery(SQL);
         timerTest(100, rs);
         rs.close();

         // Perform a fetch for every 1000th row in the result set.
         rs = stmt.executeQuery(SQL);
         timerTest(1000, rs);
         rs.close();

         // Perform a fetch for every 128th row (the default) in the result set.
         rs = stmt.executeQuery(SQL);
         timerTest(0, rs);
         rs.close();
      }

      // Handle any errors that may have occurred.
      catch (Exception e) {
         e.printStackTrace();
      }

      finally {
         if (rs != null) try { rs.close(); } catch(Exception e) {}
```

```
            if (stmt != null) try { stmt.close(); } catch(Exception e) {}
            if (con != null) try { con.close(); } catch(Exception e) {}
        }
    }

    private static void timerTest(int fetchSize, ResultSet rs) {
        try {

            // Declare the variables for tracking the row count and elapsed time.
            int rowCount = 0;
            long startTime = 0;
            long stopTime = 0;
            long runTime = 0;

            // Set the fetch size then iterate through the result set to
            // cache the data locally.
            rs.setFetchSize(fetchSize);
            startTime = System.currentTimeMillis();
            while (rs.next()) {
               rowCount++;
            }
            stopTime = System.currentTimeMillis();
            runTime = stopTime - startTime;

            // Display the results of the timer test.
            System.out.println("FETCH SIZE: " + rs.getFetchSize());
            System.out.println("ROWS PROCESSED: " + rowCount);
            System.out.println("TIME TO EXECUTE: " + runTime);
            System.out.println();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Primer 5:

```
import java.sql.*;
import java.io.*;
import com.microsoft.sqlserver.jdbc.SQLServerStatement;

public class readLargeData {

   public static void main(String[] args) {

      // Create a variable for the connection string.
      String connectionUrl =
           "jdbc:sqlserver://localhost:1433;" +
            "databaseName=AdventureWorks;integratedSecurity=true;";

      // Declare the JDBC objects.
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;

      try {
          // Establish the connection.
          Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
          con = DriverManager.getConnection(connectionUrl);

          // Create test data as an example.
          StringBuffer buffer = new StringBuffer(4000);
          for (int i = 0; i < 4000; i++)
              buffer.append( (char) ('A'));

          PreparedStatement pstmt = con.prepareStatement(
                    "UPDATE Production.Document " +
                     "SET DocumentSummary = ? WHERE (DocumentID = 1)");

          pstmt.setString(1, buffer.toString());
          pstmt.executeUpdate();
          pstmt.close();

          // In adaptive mode, the application does not have to use a server cursor
          // to avoid OutOfMemoryError when the SELECT statement produces very large
          // results.

          // Create and execute an SQL statement that returns some data.
          String SQL = "SELECT Title, DocumentSummary " +
                          "FROM Production.Document";
          stmt = con.createStatement();

          // Display the response buffering mode.
          SQLServerStatement SQLstmt = (SQLServerStatement) stmt;
          System.out.println("Response buffering mode is: " +
             SQLstmt.getResponseBuffering());

          // Get the updated data from the database and display it.
          rs = stmt.executeQuery(SQL);

          while (rs.next()) {
               Reader reader = rs.getCharacterStream(2);
               if (reader != null)
               {
                  char output[] = new char[40];
                  while (reader.read(output) != -1)
                  {
                      // Do something with the chunk of the data that was
                      // read.
```

```
                }

                System.out.println(rs.getString(1) +
                    " has been accessed for the summary column.");
                // Close the stream.
                reader.close();
             }
          }
      }
      // Handle any errors that may have occurred.
      catch (Exception e) {
         e.printStackTrace();
      }
      finally {
          if (rs != null) try { rs.close(); } catch(Exception e) {}
          if (stmt != null) try { stmt.close(); } catch(Exception e) {}
         if (con != null) try { con.close(); } catch(Exception e) {}
      }
   }
}
```

Primer 6:

```
CREATE PROCEDURE GetLargeDataValue
  (@Document_ID int,
   @Document_ID_out int OUTPUT,
   @Document_Title varchar(50) OUTPUT,
   @Document_Summary nvarchar(max) OUTPUT)

AS
BEGIN
   SELECT @Document_ID_out = DocumentID,
          @Document_Title = Title,
          @Document_Summary = DocumentSummary
    FROM  Production.Document
    WHERE DocumentID = @Document_ID
END


------------------------------------------------------------------------

import java.sql.*;
import java.io.*;
import com.microsoft.sqlserver.jdbc.SQLServerCallableStatement;

public class executeStoredProcedure {

    public static void main(String[] args) {
        // Create a variable for the connection string.
        String connectionUrl =
           "jdbc:sqlserver://localhost:1433;" +
           "databaseName=AdventureWorks;integratedSecurity=true;";

        // Declare the JDBC objects.
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
          // Establish the connection.
          Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
          con = DriverManager.getConnection(connectionUrl);

          // Create test data as an example.
          StringBuffer buffer = new StringBuffer(4000);
          for (int i = 0; i < 4000; i++)
             buffer.append( (char) ('A'));

          PreparedStatement pstmt = con.prepareStatement(
               "UPDATE Production.Document " +
               "SET DocumentSummary = ? WHERE (DocumentID = 1)");

          pstmt.setString(1, buffer.toString());
          pstmt.executeUpdate();
          pstmt.close();

          // Query test data by using a stored procedure.
          CallableStatement cstmt =
             con.prepareCall("{call dbo.GetLargeDataValue(?, ?, ?, ?)}");

          cstmt.setInt(1, 1);
          cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
          cstmt.registerOutParameter(3, java.sql.Types.CHAR);
          cstmt.registerOutParameter(4, java.sql.Types.LONGVARCHAR);
```

```
            // Display the response buffering mode.
            SQLServerCallableStatement SQLcstmt = (SQLServerCallableStatement) cstmt;
            System.out.println("Response buffering mode is: " +
                 SQLcstmt.getResponseBuffering());

            SQLcstmt.execute();
            System.out.println("DocumentID: " + cstmt.getInt(2));
            System.out.println("Document_Title: " + cstmt.getString(3));

            Reader reader = SQLcstmt.getCharacterStream(4);

            // If your application needs to re-read any portion of the value,
            // it must call the mark method on the InputStream or Reader to
            // start buffering data that is to be re-read after a subsequent
            // call to the reset method.
            reader.mark(4000);

            // Read the first half of data.
            char output1[] = new char[2000];
            reader.read(output1);
            String stringOutput1 = new String(output1);

            // Reset the stream.
            reader.reset();

            // Read all the data.
            char output2[] = new char[4000];
            reader.read(output2);
            String stringOutput2 = new String(output2);

            // Close the stream.
            reader.close();
        }
        // Handle any errors that may have occurred.
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            if (rs != null) try { rs.close(); } catch(Exception e) {}
            if (stmt != null) try { stmt.close(); } catch(Exception e) {}
            if (con != null) try { con.close(); } catch(Exception e) {}
        }
    }
}
```

Primer 7:

```
import java.sql.*;
import java.io.*;
import com.microsoft.sqlserver.jdbc.SQLServerStatement;

public class updateLargeData {

   public static void main(String[] args) {

      // Create a variable for the connection string.
      String connectionUrl =
            "jdbc:sqlserver://localhost:1433;" +
            "databaseName=AdventureWorks;integratedSecurity=true;";

      // Declare the JDBC objects.
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;
      Reader reader = null;

      try {
          // Establish the connection.
          Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
          con = DriverManager.getConnection(connectionUrl);

          stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_UPDATABLE);

          // Since the summaries could be large, make sure that
          // the driver reads them incrementally from a database,
          // even though a server cursor is used for the updatable result sets.

          // The recommended way to access the Microsoft JDBC Driver for SQL Server
          // specific methods is to use the JDBC 4.0 Wrapper functionality.
          // The following code statement demonstrates how to use the
          // Statement.isWrapperFor and Statement.unwrap methods
          // to access the driver specific response buffering methods.

          if  (stmt.isWrapperFor(com.microsoft.sqlserver.jdbc.SQLServerStatement.class))
{
              SQLServerStatement SQLstmt =
                 stmt.unwrap(com.microsoft.sqlserver.jdbc.SQLServerStatement.class);

              SQLstmt.setResponseBuffering("adaptive");
              System.out.println("Response buffering mode has been set to " +
                 SQLstmt.getResponseBuffering());
          }

          // Select all of the document summaries.
          rs = stmt.executeQuery("SELECT Title, DocumentSummary FROM
Production.Document");

          // Update each document summary.
          while (rs.next()) {

               // Retrieve the original document summary.
               reader = rs.getCharacterStream("DocumentSummary");

               if (reader == null)
               {
                   // Update the document summary.
                   System.out.println("Updating " + rs.getString("Title"));
                   rs.updateString("DocumentSummary", "Work in progress");
```

```
                rs.updateRow();
            }
            else
            {
                 // Do something with the chunk of the data that was
                // read.
                System.out.println("reading " + rs.getString("Title"));
                reader.close();
                reader = null;
            }
        }
    }
    // Handle any errors that may have occurred.
    catch (Exception e) {
       e.printStackTrace();
    }
    finally {
        if (reader != null) try { reader.close(); } catch(Exception e) {}
        if (rs != null) try { rs.close(); } catch(Exception e) {}
        if (stmt != null) try { stmt.close(); } catch(Exception e) {}
         if (con != null) try { con.close(); } catch(Exception e) {}
    }
  }
}
```