



Reading and Writing CLOBs

In this chapter, you will examine all aspects of JDBC's CLOB data type. Specifically, this chapter shows how to use JDBC's rich data type CLOB. This data type stores/retrieves large character/text objects such as large text files or Java/HTML/XML/PostScript source files.

The word *clob* has different meanings; I will use CLOB for the SQL data type and Clob for the `java.sql.Clob` interface (which represents a SQL CLOB data type).

This chapter's focus is on the following topics:

- How to define a CLOB data type in a database
- How to insert/write a CLOB into a database
- How to retrieve/read a CLOB from a database
- How to update an existing CLOB in a database
- How to delete an existing CLOB in a database

You can define, retrieve, store, and update the CLOB data type the same way you do other JDBC data types. You use either the `ResultSet.getClob()` method or the `CallableStatement.getClob()` method to retrieve CLOBs, the `PreparedStatement.setClob()` method to store them, and the `ResultSet.updateClob()` method to update them.

8-1. What Is a CLOB?

A CLOB is a Character Large Object in a database. Database vendors (such as Oracle, MySQL, IBM DB2, and so on) implement CLOBs in different ways, but as long as you use the common interface `java.sql.Clob`, you should not care about the internal implementations. CLOBs can be very large, up to 2GB or more, depending on the database. Use CLOBs with care; whenever possible, try to use caching algorithms to improve the overall performance of database applications. Large character data files can be stored as CLOB types.

JDBC's Support for CLOB

JDBC provides a single interface (`java.sql.Clob`) for handling large character/text objects. According to J2SE 5.0, `java.sql.Clob` is defined as follows:

[`java.sql.Clob` is] the mapping in the Java programming language for the SQL CLOB type. An SQL CLOB is a built-in type that stores a Character Large Object as a column value in a row of a database table. By default drivers implement a Clob object using an SQL locator (CLOB), which means that a Clob object contains a logical pointer to the SQL CLOB data rather than the data itself. A Clob object is valid for the duration of the transaction in which it was created. The Clob interface provides methods for getting the length of an SQL CLOB (Character Large Object) value, for materializing a CLOB value on the client, and for searching for a substring or CLOB object within a CLOB value. Methods in the interfaces `ResultSet`, `CallableStatement`, and `PreparedStatement`, such as `getClob` and `setClob`, allow a programmer to access an SQL CLOB value. In addition, this interface has methods for updating a CLOB value.

Table 8-1 describes the `java.sql.Clob` interface.

Table 8-1. `java.sql.Clob` Interface Description (According to J2SE 5.0)

Return Type	Method	Description
InputStream	<code>getAsciiStream()</code>	Retrieves the CLOB value designated by this Clob object as an ASCII stream.
Reader	<code>getCharacterStream()</code>	Retrieves the CLOB value designated by this Clob object as a <code>java.io.Reader</code> object (or as a stream of characters).
String	<code>getSubString(long pos, int length)</code>	Retrieves a copy of the specified substring in the CLOB value designated by this Clob object. Note that Oracle CLOBs can be up to 4GB, which exceeds the maximum “int” limit.
long	<code>length()</code>	Retrieves the number of characters in the CLOB value designated by this Clob object.
long	<code>position(Clob searchstr, long start)</code>	Retrieves the character position at which the specified Clob object <code>searchstr</code> appears in this Clob object.
long	<code>position(String searchstr, long start)</code>	Retrieves the character position at which the specified substring <code>searchstr</code> appears in the SQL CLOB value represented by this Clob object.
OutputStream	<code>setAsciiStream(long pos)</code>	Retrieves a stream to be used to write ASCII characters to the CLOB value that this Clob object represents, starting at position <code>pos</code> .
Writer	<code>setCharacterStream(long pos)</code>	Retrieves a stream to be used to write a stream of Unicode characters to the CLOB value that this Clob object represents, at position <code>pos</code> .
int	<code>setString(long pos, String str)</code>	Writes the given Java String to the CLOB value that this Clob object designates at the position <code>pos</code> .

Return Type	Method	Description
int	setString(long pos, String str, int offset, int len)	Writes len characters of str, starting at the character offset, to the CLOB value that this Clob represents.
void	truncate(long len)	Truncates the CLOB value that this Clob designates to have a length of len characters.

Oracle CLOBs

Oracle has only one type (called CLOB) to support large character/text objects. The `oracle.sql.CLOB` class is the Oracle JDBC driver's implementation of the standard JDBC `java.sql.Clob` interface.

MySQL CLOBs

This is according to the MySQL reference manual:

The four TEXT types TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT correspond to the four CLOB types and have the same maximum lengths and storage requirements. The only difference between BLOB and TEXT types is that sorting and comparison is performed in case-sensitive fashion for BLOB values and case-insensitive fashion for TEXT values. In other words, a TEXT is a case-insensitive BLOB.

MySQL has four kinds of CLOBs (please note that the JDBC calls are the same for these four types). Portions of MySQL's CLOB can be indexed.

- TINYTEXT: A character object that is stored with its length. Cannot be a key. The maximum length is 255 characters (8 bits). Takes the (varying per row) length plus 1 byte in the table.
- TEXT: A character object that is stored with its length. Cannot be a key. The maximum length is 16,535 characters (16 bits). Takes the (varying per row) length plus 2 bytes in the table.
- MEDIUMTEXT: A character object that is stored with its length. Cannot be a key. The maximum length is 16,777,216 characters (24 bits). Takes the (varying per row) length plus 3 bytes in the table.
- LONGTEXT: A character object that is stored with its length. Cannot be a key. The range is 4,294,967,295 characters (32 bits). Takes the (varying per row) length plus 4 bytes in the table.

According to MySQL (<http://dev.mysql.com/doc/connector/j/en/index.html>), “the Clob implementation does not allow in-place modification (they are *copies*, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, you should use the `PreparedStatement.setClob()` method to save changes back to the database.”

8-2. How Do You Define a CLOB Data Type in a Table?

Suppose that in your `DataFiles` table you want to store large text files. Then you might define your table as in the following sections.

Defining the Table: Oracle 9i

The following defines a table based on Oracle 9i:

```
create table DataFiles (
    id INT PRIMARY KEY,
    fileName VARCHAR(20),
    fileBody CLOB
);
```

Defining the Table: MySQL

The following defines a table based on MySQL:

```
create table DataFiles (
    id INT PRIMARY KEY,
    fileName VARCHAR(20),
    fileBody TEXT
);
```

Creating the Table: Oracle 9i

The following creates the table for Oracle 9i:

```
C:\> sqlplus system/password
SQL*Plus: Release 9.2.0.1.0 - Production on Thu Apr 17 08:40:37 2003
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
```

```
SQL> create table DataFiles
2 (id INT PRIMARY KEY,
3  fileName varchar(20),
4  fileBody CLOB
5 );
```

Table created.

```
SQL> describe DataFiles;
```

Name	Null?	Type
ID	NOT NULL	NUMBER(38)
FILENAME		VARCHAR2(20)
FILEBODY		CLOB

Creating the Table: MySQL

The following defines the table for MySQL:

```
mysql> create table DataFiles (
->     id INT PRIMARY KEY,
->     fileName VARCHAR(20),
->     fileBody TEXT
-> );
```

Query OK, 0 rows affected (0.09 sec)

```
mysql> describe DataFiles;
```

Field	Type	Null	Key	Default	Extra
id	int(11)		PRI	0	
fileName	varchar(20)	YES		NULL	
fileBody	text	YES		NULL	

3 rows in set (0.02 sec)

8-3. What Are the Restrictions for Using CLOBs?

For CLOB restrictions (such as indexing and the number of CLOBs that can be used per row), you should consult the database vendor's documentation. In general, restrictions for CLOBs are as follows:

- CLOB columns cannot be keys (primary or foreign).
- One cannot group or sort on CLOB.

The MySQL database allows you to index portions of the CLOB data type. For details, refer to the MySQL reference manual. In the MySQL database, you may also use SQL's LIKE statement for searching keywords and sentences. (You need to understand the performance of SQL's LIKE before using it.)

8-4. How Do You Create a `java.sql.Clob` Object?

You can create a `java.sql.Clob` object in only one way, and that is by using a `ResultSet` object's methods. Consider the `DataFiles` table defined earlier (using the Oracle database):

```
create table DataFiles(
    id INT PRIMARY KEY,
    fileName varchar(20),
    fileBody CLOB
);
```

The `ResultSet` interface has two methods for creating a `java.sql.Clob` object:

- `getClob(int columnPosition)`
- `getClob(String columnName)`

I will show how to create a `java.sql.Clob` object by using the overloaded `getClob()` methods.

`ResultSet.getClob(int columnPosition)`

This shows `ResultSet.getClob(int columnPosition)`:

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    conn = <get-a-valid-connection-object>
    stmt = conn.createStatement();
    rs = stmt.executeQuery("select fileBody from DataFiles");
    while (rs.next()) {
        java.sql.Clob clob = rs.getClob(1); // first column
        // now the Clob object is created and you can apply
        // methods defined in the java.sql.Clob interface
        ...
    }
}
catch(SQLException se) {
    // handle database exception
    ...
}
catch(Exception e) {
    // handle other exceptions
    ...
}
finally {
```

```

    // close resources: rs, stmt, conn
    ...
}

```

ResultSet.getClob(String columnName)

This shows `ResultSet.getClob(int columnPosition)`:

```

Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    conn = <get-a-valid-connection-object>
    stmt = conn.createStatement();
    rs = stmt.executeQuery("select fileBody from DataFiles");
    while (rs.next()) {
        java.sql.Clob clob = rs.getClob("fileBody");
        // now the Clob object is created and you can apply
        // methods defined in the java.sql.Clob interface
        ...
    }
}
catch(SQLException se) {
    // handle database exception
    ...
}
catch(Exception e) {
    // handle other exceptions
    ...
}
finally {
    // close resources: rs, stmt, conn
    ...
}

```

8-5. How Do You Materialize CLOB Data?

A SQL CLOB maps to a `java.sql.Clob` object. If you want to operate on CLOB data, you must first materialize it on the client (that is, retrieve the CLOB value's data and put it in memory on the client in the form of a Java object). The `java.sql.Clob` interface has four methods for materializing CLOB data:

- `getAsciiStream()`: Materializes the CLOB value as an input stream (`java.io.InputStream`)
- `getCharacterStream()`: Materializes the CLOB value as a stream of Unicode characters (`java.io.Reader`)
- `getSubString()`: Materializes all of the CLOB as a `String` object
- `getSubString()`: Materializes part of the CLOB as a `String` object

To materialize the CLOB value, assume that there is a Java method that returns a valid `java.sql.Clob` object:

```

public java.sql.Clob getClob(...)
    throws SQLException {
    ...
}

```

I will use the `getClob(...)` method in the following snippets.

Materializing the CLOB Value As an Input Stream (java.io.InputStream)

This shows how to materialize the CLOB value as an input stream:

```
import jcb.util.DatabaseUtil;
...//
// prints out all of ASCII bytes in the CLOB
//
byte b; // as an ASCII byte
java.sql.Clob clob = null;
java.io.InputStream input = null;
try {
    clob = getClob(...);
    input = clob.getAsciiStream();
    while ((b = input.read()) > -1) {
        // process the ASCII byte
        System.out.println(b);
    }
}
catch(SQLException se) {
    // handle database exception
    ...
}
catch(Exception e) {
    // handle other exceptions
    ...
}
finally {
    // close resources
    DatabaseUtil.close(input);
}
```

Materializing the CLOB Value As a Stream of Unicode Characters (java.io.Reader)

This shows how to materialize the CLOB value as a stream of Unicode characters:

```
import jcb.util.DatabaseUtil;
...
//
// prints out all of Unicode characters in the CLOB
//

// The character read, as an integer
// in the range 0 to 65535 (0x00-0xffff)
int aCharacter;
java.sql.Clob clob = null;
java.io.Reader input = null;
try {
    clob = getClob(...);
    input = clob.getCharacterStream();
    while ((aCharacter = input.read()) > -1) {
        // process the unicode character
        System.out.println(aCharacter);
    }
}
catch(SQLException se) {
    // handle database exception
    ...
}
```

```

    catch(Exception e) {
        // handle other exceptions
        ...
    }
    finally {
        // close resources
        DatabaseUtil close(input);
    }
}

```

Materializing the CLOB As a String Object (Get the Whole CLOB)

This shows how to materialize the CLOB value as a String object:

```

//
// get the whole CLOB as a String object
//
long length;
java.sql.Clob clob = null;
try {
    clob = getClob(...);
    length = clob.length();
    // note that the first character is at position 1
    String wholeClob = clob.getSubString(1, (int) length);
}
catch(SQLException se) {
    // handle database exception
    ...
}
catch(Exception e) {
    // handle other exceptions
    ...
}
}

```

Materializing the CLOB As a String Object (Get Part of the CLOB)

This shows how to materialize the CLOB value as a String object:

```

//
// get a part of the clob as a String object
// get 25 characters starting from position 10
//
long length = 25;
long startingPosition = 10;
java.sql.Clob clob = null;
String partialClobAsString = null;
try {
    clob = getClob(...);
    partialClobAsString = clob.getSubString(startingPosition, length);
}
catch(SQLException se) {
    // handle database exception
    ...
}
catch(Exception e) {
    // handle other exceptions
    ...
}
}

```


You can express this as a Java method:

```

/*
 * Get a part of the clob as a String object get "length"
 * characters starting from position "startingPosition"
 *
 * @param clob a CLOB object
 * @param startingPosition the first character of
 * the substring to be extracted.
 * @param length the number of consecutive characters
 * to be copied
 * @throws SQLException if there is an error accessing
 * the CLOB value
 *
 */
public static String getPartialClob(java.sql.Clob clob,
                                   long length,
                                   long startingPosition)
    throws SQLException {
    if (clob == null) {
        return null;
    }

    return clob.getSubString(startingPosition, length);
}

```

8-6. How Do You Insert a New Record with a CLOB?

Consider the DataFiles table (which has a CLOB column):

```

create table DataFiles(
    id INT PRIMARY KEY,
    fileName varchar(20),
    fileBody CLOB
);

```

You should be able to use JDBC to insert new records (which will contain fileBody as a CLOB data type). Suppose you want to insert the following data:

```

id  fileName      fileBody (content of text file)
--  -
1   file1.txt     c:/temp/data/file1.txt
2   file2.txt     c:/temp/data/file2.txt
3   file3.txt     c:/temp/data/file3.txt
4   file4.txt     c:/temp/data/file4.txt

```

Your goal is to write a program that will accept id, fileName, and fileBody (the content of the text file, as a full filename) and insert them into the DataFiles table. The client interface is as follows:

```

java InsertTextFileToMySQL <id> <fileName> <fileBody>
java InsertTextFileToOracle <id> <fileName> <fileBody>

```

Therefore, you need to develop two classes (InsertTextFileToMySQL.java and InsertTextFileToOracle.java). To insert the four records into a MySQL database, execute the following:

```

java InsertTextFileToMySQL 1 file1.txt c:/temp/data/file1.txt
java InsertTextFileToMySQL 2 file2.txt c:/temp/data/file2.txt
java InsertTextFileToMySQL 3 file3.txt c:/temp/data/file3.txt
java InsertTextFileToMySQL 4 file4.txt c:/temp/data/file4.txt

```

To insert the first three records into an Oracle 9i/10g database, execute the following:

```
java InsertTextFileToOracle 1 file1.txt c:/temp/data/file1.txt
java InsertTextFileToOracle 2 file2.txt c:/temp/data/file2.txt
java InsertTextFileToOracle 3 file3.txt c:/temp/data/file3.txt
java InsertTextFileToOracle 4 file4.txt c:/temp/data/file4.txt
```

MySQL Solution: InsertTextFileToMySQL.java

The following shows the InsertTextFileToMySQL.java solution:

```
import java.io.*;
import java.sql.*;
import jcb.db.DatabaseUtil;

public class InsertTextFileToMySQL {

    private static final String INSERT_TEXT_FILE =
        "insert into DataFiles(id, fileName, fileBody) values (?, ?, ?)";

    private static String trimArgument(String s) {
        if ((s == null) || (s.length() == 0)) {
            return s;
        }
        else {
            return s.trim();
        }
    }

    public static Connection getConnection() throws Exception {
        String driver = "org.gjt.mm.mysql.Driver";
        String url = "jdbc:mysql://localhost/octopus";
        String username = "root";
        String password = "root";
        Class.forName(driver); // load MySQL driver
        return DriverManager.getConnection(url, username, password);
    }

    public static void main(String[] args) {

        if ((args == null) || (args.length != 3)) {
            System.err.println("Usage: java InsertTextFileToMySQL ");
            System.exit(0);
        }

        String id = trimArgument(args[0]);
        String name = trimArgument(args[1]);
        String textFile = trimArgument(args[2]);

        FileInputStream fis = null;
        PreparedStatement pstmt = null;
        Connection conn = null;
        try {
            conn = getConnection();
            conn.setAutoCommit(false);

            File file = new File(textFile);
            fis = new FileInputStream(file);
```

```

        pstmt = conn.prepareStatement(INSERT_TEXT_FILE);
        pstmt.setString(1, id);
        pstmt.setString(2, name);
        pstmt.setAsciiStream(3, fis, (int)file.length());
        pstmt.executeUpdate();
        conn.commit();
    }
    catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
        e.printStackTrace();
    }
    finally {
        DatabaseUtil.close(pstmt);
        DatabaseUtil.close(fis);
        DatabaseUtil.close(conn);
    }
}
}
}

```

Testing InsertTextFileToMySQL.java

Testing the solution involves three steps.

Step 1: Prepare Input Text Files

Step 1 is to prepare the input text files:

```

$ cat c:/temp/data/file1.txt
this is file1.
hello world.
This is the last line.

```

```

$ cat c:/temp/data/file2.txt
import java.util.*;
import java.io.*;
import java.sql.*;

public class TestMySQL {
    public static Connection getConnection() throws Exception {
        String driver = "org.gjt.mm.mysql.Driver";
        String url = "jdbc:mysql://localhost/octopus";
        String username = "root";
        String password = "root";
        Class.forName(driver); // load MySQL driver
        return DriverManager.getConnection(url, username, password);
    }
}
$

```

Step 2: Compile and Run the Program

Step 2 is to compile and run the program:

```

$ javac InsertTextFileToMySQL.java
$ java InsertTextFileToMySQL 10 file1 c:/temp/data/file1.txt
$ java InsertTextFileToMySQL 20 file2 c:/temp/data/file2.txt

```

Step 3: Check the Database Content

Step 3 is to check the database content. Using MySQL's command line, you can view the inserted CLOB data.

```
mysql> use octopus;
Database changed
```

```
mysql> desc DataFiles;
```

Field	Type	Null	Key	Default	Extra
id	int(11)		PRI	0	
fileName	varchar(20)	YES		NULL	
fileBody	text	YES		NULL	

```
3 rows in set (0.00 sec)
```

```
mysql> select * from datafiles;
```

id	fileName	fileBody
10	file1	this is file1. hello world. This is the last line.
20	file2	import java.util.*; import java.io.*; import java.sql.*; public class TestMySQL { public static Connection getConnection() throws Exception { String driver = "org.gjt.mm.mysql.Driver"; String url = "jdbc:mysql://localhost/octopus"; String username = "root"; String password = "root"; Class.forName(driver); // load MySQL driver return DriverManager.getConnection(url, username, password); } }

```
2 rows in set (0.00 sec)
```

```
mysql>
```

Oracle Solution: InsertTextFileToOracle.java

The solution using the Oracle database, the `InsertTextFileToOracle` class, is identical to the MySQL database solution with the exception of the `getConnection()` method, shown next. You can download the complete solution from the book's Web site.

```
public static Connection getConnection() throws Exception {
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@localhost:1521:caspiian";
    String username = "scott";
    String password = "tiger";
```

```

        Class.forName(driver); // load Oracle driver
        return DriverManager.getConnection(url, username, password);
    }

```

Testing InsertTextFileToOracle.java

Testing the solution involves two steps.

Step 1: Compile and Run the Program

Step 1 is to prepare the input text files:

```

$ cat c:/temp/data/file1.txt
this is file1.
hello world.
This is the last line.

```

```

$ cat c:/temp/data/file2.txt
import java.util.*;
import java.io.*;
import java.sql.*;

```

```

public class TestMySQL {
    public static Connection getConnection() throws Exception {
        String driver = "org.gjt.mm.mysql.Driver";
        String url = "jdbc:mysql://localhost/octopus";
        String username = "root";
        String password = "root";
        Class.forName(driver); // load MySQL driver
        return DriverManager.getConnection(url, username, password);
    }
}
$

```

```

$ javac InsertTextFileToOracle.java
$ java InsertTextFileToOracle 100 file1 c:/temp/data/file1.txt
$ java InsertTextFileToOracle 200 file2 c:/temp/data/file2.txt

```

Step 2: Check the Database Content, and View CLOB Data

Step 2 is to check the database content. I have formatted the output so it is easier to read.

```

$ sqlplus scott/tiger
SQL*Plus: Release 10.1.0.2.0 - Production on Thu Jul 22 14:31:43 2004
SQL> desc datafiles;

```

Name	Null?	Type
ID	NOT NULL	NUMBER(38)
FILENAME		VARCHAR2(20)
FILEBODY		CLOB

```

SQL> select * from datafiles;

```

ID	FILENAME	FILEBODY
100	file1	this is file1. hello world. This is the last line.

```

200 file2      import java.util.*;
                import java.io.*;
                import java.sql.*;

                public class TestMySQL...

```

As you can see, using the CLOB data type with the MySQL and Oracle 10g databases is straightforward, but that is not the case with the Oracle 8 and Oracle 9 databases. In Oracle 8 and Oracle 9, before you can insert a real CLOB, you need to insert an empty CLOB (called `empty_clob()` in Oracle). `empty_clob()` is an Oracle function call that creates an empty Clob object. Therefore, in Oracle, you cannot just insert a Clob object into a column. First, you create a column with `empty_clob()`. Second, you update that column with the real Clob object.

8-7. How Do You Select and Display a CLOB in a JFrame?

The following example demonstrates how to retrieve a CLOB data type from the database. In this case, you retrieve the `fileBody` (content of the text file) identified by an ID (`id` is the primary key for the `DataFiles` table), displaying it in its own `JFrame`. Given that the code is lengthy to accomplish this job, the example is split over several pages. First, you perform a query to select the CLOB of interest (by providing the ID) and pull it back to the client (also known as materializing the CLOB). The rest of the code simply creates a `JFrame` to hold the retrieved text.

I have developed two classes: `ClobSelectMySQL` and `ClobSelectOracle`. You can invoke these classes by passing the ID of the file:

```

java ClobSelectMySQL <id>
java ClobSelectOracle <id>

```

The `ClobSelectMySQL` and `ClobSelectOracle` classes accept an ID (the primary key to the `DataFiles` table) and extract and display the desired CLOB in a `JFrame`.

Extracting a CLOB from MySQL

Figure 8-1 shows the CLOB from MySQL by invoking this:

```
java ClobSelectMySQL 10
```

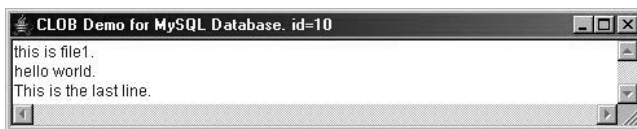
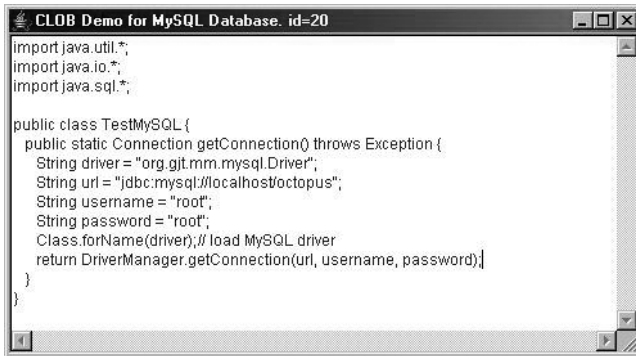


Figure 8-1. Viewing MySQL CLOB data using `JFrame`

Figure 8-2 shows the CLOB from MySQL by invoking this:

```
java ClobSelectMySQL 20
```



```

import java.util.*;
import java.io.*;
import java.sql.*;

public class TestMySQL {
    public static Connection getConnection() throws Exception {
        String driver = "org.gjt.mm.mysql.Driver";
        String url = "jdbc:mysql://localhost:octopus";
        String username = "root";
        String password = "root";
        Class.forName(driver); // load MySQL driver
        return DriverManager.getConnection(url, username, password);
    }
}

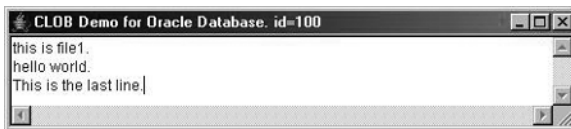
```

Figure 8-2. Viewing MySQL CLOB data using JFrame

Extracting a CLOB from Oracle

Figure 8-3 shows the CLOB from MySQL by invoking this:

```
java ClobSelectMySQL 100
```



```

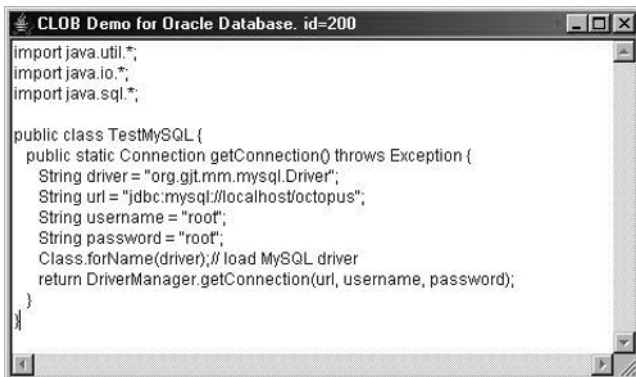
this is file1.
hello world.
This is the last line|

```

Figure 8-3. Viewing Oracle CLOB data using JFrame

Figure 8-4 shows the CLOB from MySQL by invoking this:

```
java ClobSelectMySQL 200
```



```

import java.util.*;
import java.io.*;
import java.sql.*;

public class TestMySQL {
    public static Connection getConnection() throws Exception {
        String driver = "org.gjt.mm.mysql.Driver";
        String url = "jdbc:mysql://localhost:octopus";
        String username = "root";
        String password = "root";
        Class.forName(driver); // load MySQL driver
        return DriverManager.getConnection(url, username, password);
    }
}

```

Figure 8-4. Viewing Oracle CLOB data using JFrame

Solution: ClobSelectMySQL.java

The following shows the ClobSelectMySQL.java solution:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

import jcb.db.*;

/**
 * This class displays a Clob object in a JFrame
 */
public class ClobSelectMySQL extends JPanel {

    // look and feel constants
    public static final String MOTIF_LOOK_AND_FEEL =
        "com.sun.java.swing.plaf.motif.MotifLookAndFeel";

    public static final String WINDOWS_LOOK_AND_FEEL =
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";

    public static final String METAL_LOOK_AND_FEEL =
        "javax.swing.plaf.metal.MetalLookAndFeel";

    /**
     * Get a connection object.
     */
    public static Connection getConnection() ...

    /**
     * Extract and return the CLOB object.
     * @param id the primary key to the CLOB object.
     */
    public static String getCLOB(int id) ...

    /**
     * Constructor to display CLOB object.
     * @param id the primary key to the DataFiles table
     */
    public ClobSelectMySQL(int id) ...

    public static void main(String args[]) ...
}
```

getConnection()

The following shows getConnection():

```
/**
 * Get a connection object.
 */
public static Connection getConnection() throws Exception {
    String driver = "org.gjt.mm.mysql.Driver";
    String url = "jdbc:mysql://localhost/octopus";
    String username = "root";
    String password = "root";
```



```

        Class.forName(driver); // load MySQL driver
        return DriverManager.getConnection(url, username, password);
    }

```

getCLOB()

The following shows getCLOB():

```

/**
 * Extract and return the CLOB object as String.
 * @param id the primary key to the CLOB object.
 */
public static String getCLOB(int id) throws Exception {
    Connection conn = null ;
    ResultSet rs = null;
    PreparedStatement pstmt = null;
    String query = "SELECT fileBody FROM DataFiles WHERE id = ?" ;
    try {
        conn = getConnection();
        pstmt = conn.prepareStatement(query) ;
        pstmt.setInt(1, id);
        rs = pstmt.executeQuery();
        rs.next();
        Clob clob = rs.getClob(1);
        // materialize CLOB onto client
        String wholeClob = clob.getSubString(1, (int) clob.length());
        return wholeClob;
    }
    finally {
        DatabaseUtil.close(rs);
        DatabaseUtil.close(pstmt);
        DatabaseUtil.close(conn);
    }
}

```

constructor ClobSelectMySQL()

The following shows ClobSelectMySQL():

```

/**
 * Constructor to display CLOB object.
 * @param id the primary key to the DataFiles table
 */
public ClobSelectMySql(int id) throws Exception {
    setLayout(new GridLayout(1, 1));
    add(new TextArea(getCLOB(id), 3, 10));
}

```

main()

The following shows main():

```

public static void main(String args[]) throws Exception {
    int id = Integer.parseInt(args[0]);
    UIManager.setLookAndFeel(METAL_LOOK_AND_FEEL) ;
    JFrame frame = new JFrame("CLOB Demo for MySQL Database. id="+id);
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {

```

```

        System.exit(0);
    }
});
frame.setContentPane(new ClobSelectMySQL(id));
frame.pack();
frame.setVisible(true);
}

```

Solution: ClobSelectOracle.java

The Oracle solution is identical to the MySQL solution with the exception of the `getConnection()` method, which returns an Oracle Connection object. You can download the complete Oracle solution from the book's Web site.

This is the Java method `getConnection()` for the Oracle database:

```

/**
 * Get an Oracle connection object.
 */
public static Connection getConnection() throws Exception {
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@scorpien:1521:caspien";
    String username = "scott";
    String password = "tiger";
    Class.forName(driver); // load Oracle driver
    return DriverManager.getConnection(url, username, password);
}

```

8-8. How Do You Select and Display an Oracle CLOB Using a Servlet?

I will use the data set up in the previous recipe to show how to select and display an Oracle CLOB using a servlet.

Viewing Oracle CLOB Data

I have developed a servlet, `DisplayOracleClobServlet`, that accepts the ID of a file and displays the associated file. (As you can see from the previous pages, the output has not been formatted, and the CLOB data has not been altered.) Run the servlet with an ID of 100, as shown in Figure 8-5, and then run the servlet with an ID of 200, as shown in Figure 8-6. Next, run the servlet with an ID of 300, which is not in the database. If the file's ID does not exist in the database, then you will get an error, as shown in Figure 8-7.



Figure 8-5. Viewing Oracle CLOB data using a servlet (id=100)

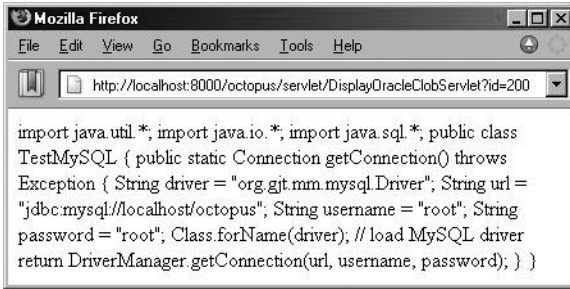


Figure 8-6. Viewing Oracle CLOB data using a servlet (id=200)



Figure 8-7. Viewing nonexistent Oracle CLOB data using a servlet

If the database connection information is not valid (a wrong username/password or wrong database URL) or if the database is not available, then you will get the error shown in Figure 8-8.

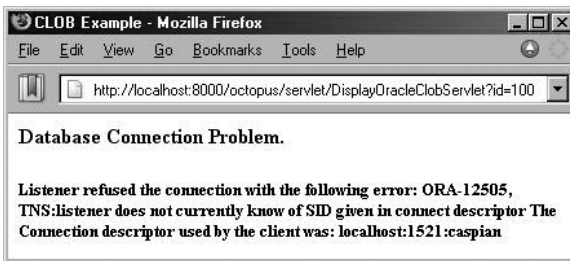


Figure 8-8. Displaying error message using a servlet

Displaying a CLOB Using a Servlet: DisplayOracleClobServlet

The following shows the DisplayOracleClobServlet solution:

```
import java.io.*;
import java.sql.*;

import javax.servlet.*;
import javax.servlet.http.*;

import jcb.db.DatabaseUtil;

public class DisplayOracleClobServlet extends HttpServlet {
```

```

public static Connection getConnection() throws Exception {
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@localhost:1521:caspiant";
    String username = "scott";
    String password = "tiger";
    Class.forName(driver); // load Oracle driver
    return DriverManager.getConnection(url, username, password);
}

public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
    throws IOException, ServletException {

    System.out.println("-- DisplayOracleClobServlet begin --");

    Clob fileAsCLOB = null;
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    String id = request.getParameter("id").trim();
    String query = "select fileBody from DataFiles where id = "+id;
    ServletOutputStream out = response.getOutputStream();

    // all responses will be in text/html format
    response.setContentType("text/html");

    try {
        conn = getConnection();
    }
    catch(Exception e) {
        out.println("<html><head><title>CLOB Example</title></head>");
        out.println("<body><h4>Database Connection Problem.</h4>");
        out.println("<h5>"+e.getMessage()+"</h5></body></html>");
        return;
    }

    try {
        stmt = conn.createStatement();
        rs = stmt.executeQuery(query);
        if (rs.next()) {
            fileAsCLOB = rs.getClob(1);
        }
        else {
            out.println("<html><head><title>CLOB Example</title></head>");
            out.println("<body><h3>No file found for id="+
                id+"</h3></body></html>");
            return;
        }
    }

    // Materialize the CLOB as a String object (get the whole clob).
    long length = fileAsCLOB.length();
    // note that the first character is at position 1
    String fileAsString = fileAsCLOB.getSubString(1, (int) length);

```

```

        // write it for display
        out.println(fileAsString);
        System.out.println("CLOB writing done.");
    }
    catch (SQLException e) {
        out.println("<html><head><title>Error: CLOB Example</title></head>");
        out.println("<body><h3>Error="+e.getMessage()+"</h3></body></html>");
        return;
    }
    finally {
        DatabaseUtil.close(rs);
        DatabaseUtil.close(stmt);
        DatabaseUtil.close(conn);
    }
    System.out.println("-- DisplayOracleClobServlet end --");
}

public void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
    doGet(request, response);
}
}

```

8-9. How Do You Select and Display a MySQL CLOB Using a Servlet?

MySQL supports CLOB and offers four data types for using it:

Type	Format
TINYTEXT	A string with a maximum length of 255 characters
TEXT	A string with a maximum length of 65,535 characters
MEDIUMTEXT	A string with a maximum length of 16,777,215 characters
LONGTEXT	A string with a maximum length of 4,294,967,295 characters

Setting Up the MySQL Database

This shows how to set up the MySQL database for this example:

```
mysql> use octopus;
Database changed
```

```
mysql> desc DataFiles;
```

Field	Type	Null	Key	Default	Extra
id	int(11)		PRI	0	
fileName	varchar(20)	YES		NULL	
fileBody	text	YES		NULL	

```
3 rows in set (0.00 sec)
```

```
mysql> select * from datafiles;
```

```

+-----+-----+-----+
| id | fileName | fileBody
+-----+-----+-----+
| 10 | file1    | this is file1.
|   |          | hello world.
|   |          | This is the last line.
+-----+-----+-----+
| 20 | file2    | import java.util.*;
|   |          | import java.io.*;
|   |          | import java.sql.*;
|   |          |
|   |          | public class TestMySQL {
|   |          |     public static Connection getConnection() throws Exception {
|   |          |         String driver = "org.gjt.mm.mysql.Driver";
|   |          |         String url = "jdbc:mysql://localhost/octopus";
|   |          |         String username = "root";
|   |          |         String password = "root";
|   |          |         Class.forName(driver); // load MySQL driver
|   |          |         return DriverManager.getConnection(url, username, password);
|   |          |     }
|   |          | }
+-----+-----+-----+
2 rows in set (0.00 sec)

```

mysql>

Viewing MySQL CLOB Data

I have developed a servlet, `DisplayMySQLClobServlet`, that accepts the ID of a file and displays the associated file. (As you can see, the output has not been formatted, and the CLOB data has not been altered.) Run the servlet with an ID of 10, as shown in Figure 8-9, and then run the servlet with an ID of 20, as shown in Figure 8-10. Next, run the servlet with an ID of 30 (which is not in the database); if the data is not in the database, you will see the error shown in Figure 8-11.



Figure 8-9. Viewing MySQL CLOB data using a servlet (id=10)

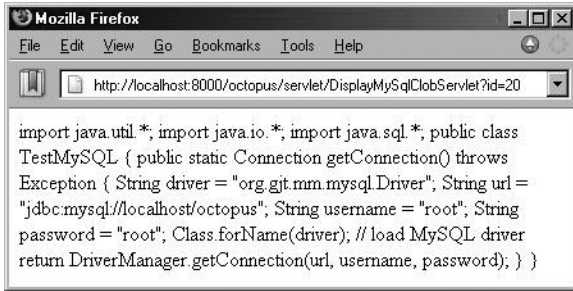


Figure 8-10. Viewing MySQL CLOB data using a servlet (id=20)



Figure 8-11. Viewing nonexistent Oracle CLOB data using a servlet

If the database connection information is not valid (a wrong username/password or wrong database URL) or if database is not available, then you will get the error shown in Figure 8-12.

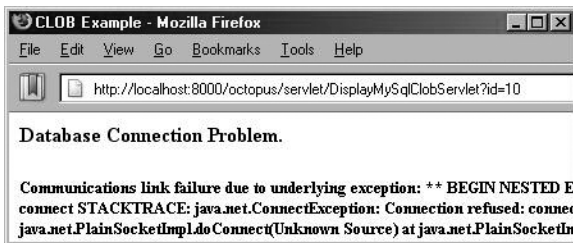


Figure 8-12. Displaying error message using a servlet

Displaying a CLOB Using a Servlet: DisplayMySqlClobServlet

The MySQL solution is identical to the Oracle solution with the exception of the `getConnection()` method, which returns a MySQL Connection object. You can download the complete MySQL solution from the book's Web site.

This is `getConnection()` for the MySQL database:

```
public static Connection getConnection() throws Exception {
    String driver = "org.gjt.mm.mysql.Driver";
    String url = "jdbc:mysql://localhost/octopus";
    String username = "root";
```

```

String password = "root";
Class.forName(driver); // load MySQL driver
return DriverManager.getConnection(url, username, password);
}

```

8-10. How Do You Select and Display an Oracle CLOB (As a URL) Using a Servlet?

If CLOB data (such as an RSS feed, a text résumé, or an HTML blog) is too big (more than a couple of megabytes) and is shared among many users, it is a good idea to retrieve the CLOB from the database, then create a copy of it on the server side, and finally make it URL addressable (to avoid the performance cost).

Here I will provide a solution that displays a CLOB as a URL on the browser; when you click the URL (or open the URL in the browser), then you will view the CLOB. Therefore, a servlet will accept an ID of a CLOB, and then it will store the CLOB on the server as a text file and send an associated URL (of the CLOB) to the browser. To solve this problem effectively, you need to create a directory (on the Web server side) and make it URL addressable; therefore, you need to define the following two parameters (defined inside the servlet):

- CLOB_DIRECTORY, the directory where CLOB data will be placed as files
- CLOB_URL, the CLOB_DIRECTORY as a URL

Setting Up the Database

For this solution, I will use the CLOB data prepared in earlier sections.

Solution

I have developed a servlet, `DisplayOracleClobAsURLServlet`, that accepts the ID of a file and displays the associated file. For example, if `id=200`, then you will get the screen shown in Figure 8-13.



Figure 8-13. *Displaying a generated GUID for a CLOB using a servlet*

By opening the CLOB as a URL, you will get the screen shown in Figure 8-14.

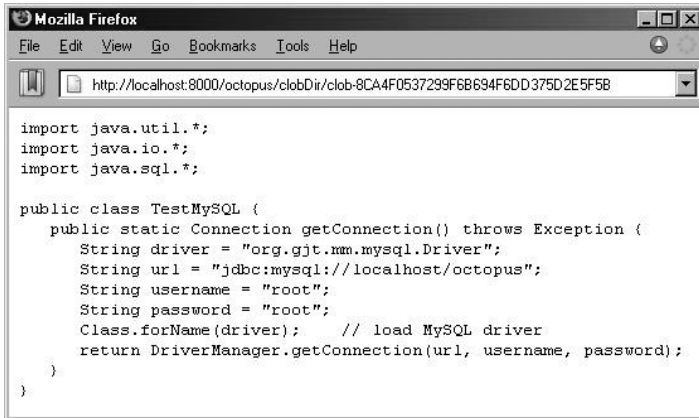


Figure 8-14. *Displaying a CLOB using a generated GUID*

If the file's ID does not exist in the database, then you will get the screen shown in Figure 8-15.



Figure 8-15. *Displaying an error for a nonexistent CLOB using a servlet*

If the database connection information is not valid (wrong username/password or wrong database URL), then the servlet will display an error message.

Displaying a CLOB Using a Servlet: DisplayOracleClobAsURLServlet

The following shows the DisplayOracleClobAsURLServlet solution:

```

import java.io.*;
import java.sql.*;

import javax.servlet.*;
import javax.servlet.http.*;

import jcb.db.DatabaseUtil;
import jcb.util.IOUtil;
import jcb.util.RandomGUID;

public class DisplayOracleClobAsURLServlet extends HttpServlet {

    // directory where clob data will be placed as files.
    private static final String CLOB_DIRECTORY =
        "c:/tomcat/webapps/octopus/clobDir";

```

```

// CLOB_DIRECTORY as a URL
private static final String CLOB_URL =
    "http://localhost:8000/octopus/clobDir";

private static final String CLOB_FILE_PREFIX = "/clob-";

public static Connection getConnection() ...
private static String getClobAsURL(Clob Clob) ...
public void doGet(...) ...

public void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
    doGet(request, response);
}
}

```

getConnection()

The following shows `getConnection()`:

```

public static Connection getConnection() throws Exception {
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@localhost:1521:caspien";
    String username = "scott";
    String password = "tiger222";
    Class.forName(driver); // load Oracle driver
    return DriverManager.getConnection(url, username, password);
}

```

getClobAsURL()

The following shows `getClobAsURL()`:

```

private static String getClobAsURL(Clob Clob) throws Exception {
    InputStream in = null;
    FileOutputStream out = null;
    try {
        if (Clob == null) {
            return null;
        }

        // get a random GUID for Clob filename
        String guid = RandomGUID.getGUID();
        String ClobFile = CLOB_DIRECTORY + CLOB_FILE_PREFIX + guid;
        in = Clob.getAsciiStream();
        if (in == null) {
            return null;
        }

        out = new FileOutputStream(ClobFile);
        int length = (int) Clob.length();
        int bufferSize = 1024;
        byte[] buffer = new byte[bufferSize];
        while ((length = in.read(buffer)) != -1) {
            out.write(buffer, 0, length);
        }
    }
}

```

```

        out.flush();
        return CLOB_URL + CLOB_FILE_PREFIX + guid;
    }
    finally {
        IOUtil.close(in);
        IOUtil.close(out);
    }
}

```

doGet()

The following shows doGet():

```

public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
    throws IOException, ServletException {
    Clob clob = null;
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    String id = request.getParameter("id").trim();
    String query = "select fileBody from DataFiles where id = "+id;
    ServletOutputStream out = response.getOutputStream();
    response.setContentType("text/html");
    out.println("<html><head><title>DisplayOracleClobAsURLServlet"+
        "</title></head>");
    try {
        conn = getConnection();
    }
    catch(Exception e) {
        out.println("<body><h4>Database Connection Problem.</h4>");
        out.println("<h5>"+e.getMessage()+"</h5></body></html>");
        return;
    }

    try {
        stmt = conn.createStatement();
        rs = stmt.executeQuery(query);
        if (rs.next()) {
            clob = rs.getClob(1);
            out.println("<body><h3>file id="+id+"</h3>"+
                getClobAsURL(clob)+"</body></html>");
        }
        else {
            out.println("<body><h1>No File found for id="+id+"</h1></body></html>");
            return;
        }
    }
    catch (Exception e) {
        out.println("<body><h1>Error="+e.getMessage()+"</h1></body></html>");
        return;
    }
    finally {
        DatabaseUtil.close(rs);
        DatabaseUtil.close(stmt);
        DatabaseUtil.close(conn);
    }
}

```

8-11. How Do You Select and Display a MySQL CLOB (As a URL) Using a Servlet?

If CLOB data (such as an RSS feed, a text résumé, or an HTML blog) is too big (more than a couple of megabytes) and is shared among many users, it is a good idea to retrieve the CLOB from the database, then create a copy of it on the server side, and finally make it URL addressable (to avoid the performance cost).

Here I provide a solution that displays a CLOB as a URL on the browser; when you click the URL (or open the URL in the browser), then you will view the CLOB. Therefore, a servlet will accept an ID of a CLOB, and then it will store the CLOB on the server as a text file and send an associated URL (of the CLOB) to the browser. To solve this problem effectively, you need to create a directory (on the Web server side) and make it URL addressable; therefore, you need to define the following two parameters (defined inside the servlet):

- CLOB_DIRECTORY, the directory where CLOB data will be placed as files
- CLOB_URL, the CLOB_DIRECTORY as a URL

Solution

I have developed a servlet, `DisplayMySQLClobAsURLServlet`, that accepts the ID of a file and displays the associated file. For example, if `id=10`, then you will get the screen shown in Figure 8-16.



Figure 8-16. Displaying a generated GUID for a CLOB using a servlet

By opening the CLOB as a URL, you will get the screen shown in Figure 8-17.



Figure 8-17. Displaying a CLOB using a generated GUID

If the file's ID does not exist in the database, then you will get the screen shown in Figure 8-18.



Figure 8-18. *Displaying an error for a nonexistent CLOB using a servlet*

If the database connection information is not valid (a wrong username/password or wrong database URL), then you will get an error message.

Setting Up the MySQL Database

To show this example in action, I will use the `DataFiles` table defined and populated in earlier sections.

Displaying a CLOB Using a Servlet: `DisplayMySqlClobAsURLServlet`

The MySQL solution is identical to the Oracle solution with the exception of the `getConnection()` method, which returns a `MySQLConnection` object. You can download the complete MySQL solution from the book's Web site.

This is `getConnection()` for the MySQL database:

```
public static Connection getConnection() throws Exception {
    String driver = "org.gjt.mm.mysql.Driver";
    String url = "jdbc:mysql://localhost/octopus";
    String username = "root";
    String password = "root";
    Class.forName(driver); // load MySQL driver
    return DriverManager.getConnection(url, username, password);
}
```

8-12. How Do You Insert a CLOB into an Oracle Database Using a Servlet?

The SQL `INSERT` statement inserts new rows/records into a table. The general syntax is as follows:

```
INSERT INTO table_name (column_name_1, column_name_2, ...)
VALUES (value_for_column_1, value_for_column_2, ...)
```

To insert a CLOB into an Oracle database using a servlet, you will represent the intended CLOB as a URL (the URL will be pointing to CLOB data, such as a text file). The reason for this is that servlets cannot access the local file system (a client's/browser's local machine). You can also pass the CLOB to the database as a `String` object. Therefore, you will represent the CLOB as a URL. The following sections will use the `DataFiles` table defined in the earlier sections.

Setting Up the Oracle Database

Oracle 10 has simplified CLOBs in JDBC. There is no need to use Oracle's proprietary SQL functions, such as `empty_clob`. Inserting CLOBs is simple; in fact, CLOBs are just long `String` objects. This shows how to set up the database:

```
$ sqlplus scott/tiger
SQL*Plus: Release 10.1.0.2.0 - Production on Fri Feb 18 16:42:51 2005
Connected to: Oracle Database 10g Enterprise Edition Release 10.1.0.2.0
```

```
SQL> desc datafiles;
Name                               Null?    Type
-----
ID                                  NOT NULL NUMBER(38)
FILENAME                            VARCHAR2(20)
FILEBODY                             CLOB
```

```
SQL> select id, filename from datafiles;
```

```

      ID  FILENAME
-----  -
      1000  file1
      2000  file2
```

```
SQL> insert into datafiles(id, filename, filebody)
  2 values(4000, 'file4000', 'This is the content of file4000.');
```

```
1 row created.
```

```
SQL> select id, filename from datafiles;
```

```

      ID  FILENAME
-----  -
      4000  file4000
      1000  file1
      2000  file2
```

Creating the Oracle Servlet Interface

The servlet interface is as follows:

```
http://localhost:8000/octopus/servlet/InsertClobToOracleServlet?
id=<id>&name=<name>&file=<file-as-URL>
```

Therefore, InsertClobToMySQLServlet has three parameters:

- id (the ID of file)
- name (the name of file)
- file (the URL of file, representing the CLOB; the servlet will open the URL, construct a CLOB, and insert it into the CLOB column of the DataFiles table)

Now insert a new record with the following data in the DataFiles table:

- id (500)
- name (file500)
- file (the URL of file: <http://www.geocities.com/mparsian/data/file500.txt>)

Figure 8-19 shows the content of the URL (<http://www.geocities.com/mparsian/data/file500.txt>).

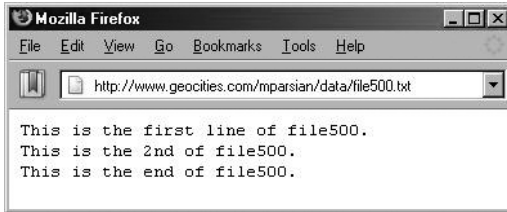


Figure 8-19. *Displaying a file using a servlet*

Therefore, the servlet call is as follows:

```
http://localhost:8000/octopus/servlet/InsertClobToOracleServlet?  
id=500&name=file500&file=http://www.geocities.com/mparsian/data/file500.txt
```

Inserting a New CLOB Record

Figure 8-20 shows the result of inserting a new CLOB record using a Java servlet.



Figure 8-20. *Inserting a new CLOB record using a servlet*

Reinserting the Same CLOB Record

Figure 8-21 shows the result of reinserting a new CLOB record using a servlet.



Figure 8-21. *Reinserting the same CLOB record using a servlet*

Viewing the Database Content After Insertion

Using `DisplayOracleClobServlet`, you can view the CLOB in a Web browser, as shown in Figure 8-22.

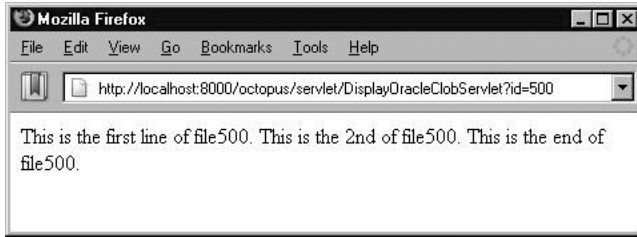


Figure 8-22. Viewing CLOB using a servlet

Solution: InsertClobToOracleServlet

The following shows the InsertClobToOracleServlet solution:

```
import java.io.*;
import java.sql.*;

import javax.servlet.*;
import javax.servlet.http.*;

import jcb.db.DatabaseUtil;
import jcb.util.IOUtil;

public class InsertClobToOracleServlet extends HttpServlet {
    static final String INSERT_CLOB =
        "insert into datafiles(id, filename, filebody) values (?, ?, ?)";

    public static Connection getConnection() throws Exception { ... }
    public void doGet(...) { ... }
    public void doPost(...) { ... }
    public void insertCLOB(...) { ... }
    public static String getClobContentAsString(...) { ... }
    private static String trimParameter(...) { ... }
}
```

getConnection()

The following shows getConnection():

```
public static Connection getConnection() throws Exception {
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@localhost:1521:caspiant";
    String username = "scott";
    String password = "tiger";
    Class.forName(driver); // load Oracle driver
    return DriverManager.getConnection(url, username, password);
}
```

doGet()

The following shows doGet():

```
public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
    throws IOException, ServletException {
```



```

String fileContent = null;
Connection conn = null;

String id = trimParameter(request.getParameter("id"));
String name = trimParameter(request.getParameter("name"));
String fileAsURL = trimParameter(request.getParameter("file"));
ServletOutputStream out = response.getOutputStream();

response.setContentType("text/html");
out.println("<html><head><title>InsertClobToOracleServlet</title></head>");

try {
    conn = getConnection();
    fileContent = getClobsContentAsString(fileAsURL);
    insertCLOB(conn, id, name, fileContent);
    out.println("<body><h4>OK: inserted a new record with id="
        +id+"</h4></body></html>");
}
catch(Exception e) {
    e.printStackTrace();
    out.println("<body><h4>Error: "+e.getMessage()+"</h4></body></html>");
}
}

```

insertCLOB()

The following shows insertCLOB():

```

public void insertCLOB(Connection conn,
                      String id,
                      String name,
                      String fileContent)
    throws Exception {
    PreparedStatement pstmt = null;
    try {
        pstmt = conn.prepareStatement(INSERT_CLOB);
        pstmt.setString(1, id);
        pstmt.setString(2, name);
        pstmt.setString(3, fileContent);
        pstmt.executeUpdate();
    }
    finally {
        DatabaseUtil.close(pstmt);
    }
}

```

getClobsContentAsString()

The following shows getClobsContentAsString():

```

public static String getClobsContentAsString(String urlAsString)
    throws Exception {
    InputStream content = null;
    try {
        java.net.URL url = new java.net.URL(urlAsString);
        java.net.URLConnection urlConn = url.openConnection();
        urlConn.connect();
        content = urlConn.getInputStream();
    }
}

```

```

        return IOUtil.inputStreamToString(content);
    }
    finally {
        IOUtil.close(content);
    }
}

```

trimParameter()

The following shows trimParameter():

```

private static String trimParameter(String s) {
    if ((s == null) || (s.length() == 0)) {
        return s;
    }
    else {
        return s.trim();
    }
}
}

```

8-13. How Do You Insert a CLOB into a MySQL Database Using a Servlet?

To insert a CLOB into a MySQL database using a servlet, you will represent the intended CLOB as a URL (the URL will be pointing to a CLOB data, such as a text file). The reason for this is that servlets cannot access the local file system (a client's/browser's local machine). You can also pass the CLOB to the database as a String object. Therefore, you will represent the CLOB as a URL. The following sections use the Datafiles table defined in the earlier questions.

Setting Up the MySQL Database

Inserting CLOBs in MySQL is simple; in fact, CLOBs are just long String objects. This shows how to set up the database:

```
mysql> desc datafiles;
```

Field	Type	Null	Key	Default	Extra
id	int(11)		PRI	0	
fileName	varchar(6)	YES		NULL	
fileBody	text	YES		NULL	

```
3 rows in set (0.00 sec)
```

```
mysql> select id, filename from datafiles;
```

id	filename
20	file2

```
1 row in set (0.00 sec)
```

```
mysql> insert into datafiles(id, filename, filebody)
  2 values(4000, 'file4000', 'This is the content of file4000.');
```

Query OK, 1 row affected, 1 warning (0.00 sec)

```
mysql> select id, filename from datafiles;
+-----+-----+
| id   | filename |
+-----+-----+
| 20   | file2    |
+-----+-----+
| 4000 | file4000 |
+-----+-----+
2 rows in set (0.00 sec)
```

Creating the Oracle Servlet Interface

The servlet interface is as follows:

```
http://localhost:8000/octopus/servlet/InsertClobToMySQLServlet?
id=<id>&name=<name>&file=<file-as-URL>
```

Therefore, `InsertClobToMySQLServlet` has three parameters:

- `id` (the ID of file)
- `name` (the name of file)
- `file` (the URL of file representing the CLOB; the servlet will open the URL, construct a CLOB, and insert it into the CLOB column of the `DataFiles` table)

Let's insert a new record with the following data in the `DataFiles` table:

- `id` (500)
- `name` (file500)
- `file` (the URL of file: `http://www.geocities.com/mparsian/data/file500.txt`)

Figure 8-23 shows the content of the URL (`http://www.geocities.com/mparsian/data/file500.txt`).

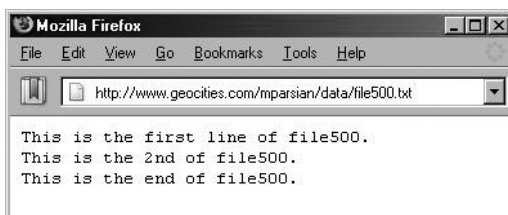


Figure 8-23. Viewing CLOB using a servlet

Therefore, the servlet call is as follows:

```
http://localhost:8000/octopus/servlet/InsertClobToMySQLServlet?
id=500&name=file500&file=http://www.geocities.com/mparsian/data/file500.txt
```

Inserting a New CLOB Record

Figure 8-24 shows the result of inserting a new CLOB record.



Figure 8-24. Inserting a new CLOB record using a servlet

Viewing the Database Content After Insertion

Using MySQL's command prompt, you can view the inserted record (with id=500):

```
mysql> select id, filename from datafiles;
+-----+-----+
| id   | filename |
+-----+-----+
| 20   | file2    |
+-----+-----+
| 500  | file500  |
+-----+-----+
| 4000 | file4000 |
+-----+-----+
3 rows in set (0.00 sec)

mysql> select id, filename, filebody from datafiles where id=500;
+-----+-----+-----+
| id   | filename | filebody |
+-----+-----+-----+
| 500  | file50   | This is the first line of file500. |
|      |          | This is the 2nd of file500.       |
|      |          | This is the end of file500.       |
+-----+-----+-----+

1 row in set (0.00 sec)
```

Using `DisplayOracleClobServlet`, you can view the CLOB in a Web browser, as shown in Figure 8-25.

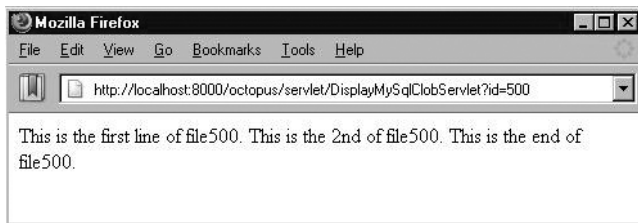


Figure 8-25. Viewing an inserted CLOB record using a servlet

Solution: InsertClobToMySQLServlet

The MySQL solution is identical to the Oracle solution with the exception of the `getConnection()` method, which returns a MySQL Connection object. You can download the complete MySQL solution from the book's Web site.

The following is `getConnection()` for the MySQL database:

```
public static Connection getConnection() throws Exception {
    String driver = "org.gjt.mm.mysql.Driver";
    String url = "jdbc:mysql://localhost/octopus";
    String username = "root";
    String password = "root";
    Class.forName(driver); // load MySQL driver
    return DriverManager.getConnection(url, username, password);
}
```

8-14. How Do You Update an Existing CLOB of an Oracle Database Using a Servlet?

The SQL `UPDATE` statement modifies the existing column's data in a table. The simplified syntax is as follows (which updates a single column):

```
UPDATE table_name
    SET column_name_1 = new_value_1,
        column_name_2 = new_value_2, ...
    WHERE column_name_x = some_value_1 and
          column_name_y = some_value_2 and ...
```

You can update any number of columns using the SQL `UPDATE` statement. For details, please refer to the following Web site: http://www.w3schools.com/sql/sql_update.asp.

To update a CLOB in an Oracle database using a servlet, you will represent the new value of a CLOB as a URL (the URL will be pointing to a CLOB data, such as a text file). The reason for this is that servlets cannot access the local file system (a client's/browser's local machine). You can also pass the new CLOB value to the database as a `String` object. Therefore, you will represent the new value of a CLOB as a URL. The following sections use the `DataFiles` table defined in the earlier sections.

Setting Up the Oracle Database

Oracle 10 has simplified CLOBs in JDBC. There is no need to use Oracle's proprietary SQL functions, such as `empty_clob`. Inserting/updating CLOBs is simple; in fact, CLOBs are just long `String` objects. This shows how to set up the database:

```
$ sqlplus scott/tiger
SQL*Plus: Release 10.1.0.2.0 - Production on Sat Feb 19 22:38:38 2005
Copyright (c) 1982, 2004, Oracle. All rights reserved.
```

```
SQL> desc datafiles;
Name                                Null?    Type
-----
ID                                    NOT NULL NUMBER(38)
FILENAME                             VARCHAR2(20)
FILEBODY                              CLOB
```

```
SQL> select id, filename, filebody from datafiles where id=1000;
```

```
ID      FILENAME  FILEBODY
-----
1000    file1     this is file1. hello world.
```

```
SQL> update datafiles
  2   set filebody='this is a long ... string. aha.'
  3   where id=1000;

1 row updated.

SQL> commit;
Commit complete.

SQL> select id, filename, filebody from datafiles where id=1000;
ID      FILENAME  FILEBODY
-----
1000   file1     this is a long ... string. aha.
```

Creating the Oracle Servlet Interface

The servlet interface is as follows:

```
http://localhost:8000/octopus/servlet/UpdateOracleClobServlet?
id=<id>&file=<file-as-URL>
```

Therefore, UpdateOracleClobServlet has two parameters:

- id (the ID of file, which uniquely identifies record)
- file (the URL of file representing the CLOB; the servlet will open the URL, construct a CLOB, and update it into the CLOB column of the DataFiles table)

Let's update an existing record with the following data in the DataFiles table:

- id (1000)
- file (the URL of file: <http://www.geocities.com/mparsian/data/file500.txt>)

Figure 8-26 shows the content of the URL (<http://www.geocities.com/mparsian/data/file500.txt>).

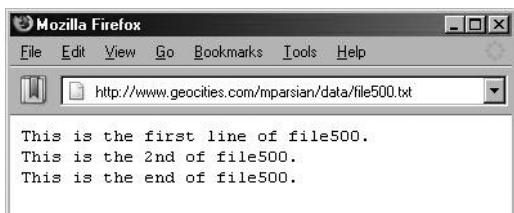


Figure 8-26. *Displaying a file using a servlet*

Therefore, the servlet call is as follows (but all in one line):

```
http://localhost:8000/octopus/servlet/UpdateOracleClobServlet?
id=1000&file=http://www.geocities.com/mparsian/data/file500.txt
```

Updating an Existing CLOB Record

Figure 8-27 shows the result of updating an existing CLOB record.

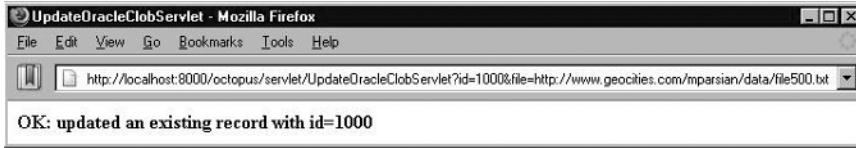


Figure 8-27. Updating a CLOB using a servlet

Database Content After Insertion

The following shows the database after the insertion:

```
SQL> select id, filename, filebody from datafiles where id=1000;
```

ID	FILENAME	FILEBODY
1000	file1	This is the first line of file500. This is the 2nd of file500. This is the end

Using `DisplayOracleClobServlet`, you can view the CLOB in a Web browser, as shown in Figure 8-28.



Figure 8-28. Viewing an updated CLOB using a servlet

The Solution: UpdateOracleClobServlet

The following shows the `UpdateOracleClobServlet` solution:

```
import java.io.*;
import java.sql.*;

import javax.servlet.*;
import javax.servlet.http.*;

import jcb.db.DatabaseUtil;
import jcb.util.IOUtil;

public class UpdateOracleClobServlet extends HttpServlet {
    static final String UPDATE_CLOB =
        "update datafiles set filebody=? where id=?";

    public static Connection getConnection() throws Exception {... }
    public void doGet(...) {... }
    public void doPost(...) {...}
```

```

    public void updateCLOB(...) {...}
    public static String getClobContentAsString(...) {...}
    private static String trimParameter(...) {...}
}

```

getConnection()

The following shows getConnection():

```

public static Connection getConnection() throws Exception {
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@localhost:1521:caspian";
    String username = "scott";
    String password = "tiger";
    Class.forName(driver); // load Oracle driver
    return DriverManager.getConnection(url, username, password);
}

```

doGet()

The following shows doGet():

```

public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
    throws IOException, ServletException {
    String fileContent = null;
    Connection conn = null;
    String id = trimParameter(request.getParameter("id"));
    String fileAsURL = trimParameter(request.getParameter("file"));
    ServletOutputStream out = response.getOutputStream();

    response.setContentType("text/html");
    out.println("<html><head><title>UpdateOracleClobServlet</title></head>");

    try {
        conn = getConnection();
        fileContent = getClobContentAsString(fileAsURL);
        updateCLOB(conn, id, fileContent);
        out.println("<body><h4>OK: updated an existing "+
            record with id="+id+"</h4></body></html>");
    }
    catch(Exception e) {
        e.printStackTrace();
        out.println("<body><h4>Error: "+e.getMessage()+"</h4></body></html>");
    }
}

```

updateCLOB()

The following shows updateCLOB():

```

public void updateCLOB(Connection conn, String id, String fileContent)
    throws Exception {
    PreparedStatement pstmt = null;
    try {
        pstmt = conn.prepareStatement(UPDATE_CLOB);
        pstmt.setString(1, fileContent);
        pstmt.setString(2, id);
    }
}

```



```

        pstmt.executeUpdate();
    }
    finally {
        DatabaseUtil.close(pstmt);
    }
}

```

getClobsContentAsString()

The following shows `getClobsContentAsString()`:

```

public static String getClobsContentAsString(String urlAsString)
    throws Exception {
    InputStream content = null;
    try {
        java.net.URL url = new java.net.URL(urlAsString);
        java.net.URLConnection urlConn = url.openConnection();
        urlConn.connect();
        content = urlConn.getInputStream();
        return IOUtil.inputStreamToString(content);
    }
    finally {
        IOUtil.close(content);
    }
}

```

trimParameter()

The following shows `trimParameter()`:

```

private static String trimParameter(String s) {
    if ((s == null) || (s.length() == 0)) {
        return s;
    }
    else {
        return s.trim();
    }
}

```

8-15. How Do You Update an Existing CLOB of a MySQL Database Using a Servlet?

The SQL `UPDATE` statement modifies the existing column's data in a table. The simplified syntax is as follows (which updates a single column):

```

UPDATE table_name
    SET column_name_1 = new_value_1,
        column_name_2 = new_value_2, ...
    WHERE column_name_x = some_value_1 and
          column_name_y = some_value_2 and ...

```

You can update any number of columns using the SQL `UPDATE` statement. For details, please refer to the following Web site: http://www.w3schools.com/sql/sql_update.asp.

To update a CLOB in a MySQL database using a servlet, you will represent the new value of a CLOB as a URL (the URL will be pointing to a CLOB data, such as a text file). The reason for this is that servlets cannot access the local file system (a client's/browser's local machine). You can also pass the new CLOB value to the database as a `String` object. Therefore, you will represent the new value of a CLOB as a URL. The following sections use the `DataFiles` table defined in the earlier questions.

Each database vendor handles BLOB/CLOB updates differently. According to the MySQL documentation (<http://dev.mysql.com/doc/connector/j/en/cj-implementation-notes.html>), “the Clob implementation does not allow in-place modification (they are *copies*, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, you should use the `PreparedStatement.setClob()` method to save changes back to the database. The JDBC API does not have a `ResultSet.updateClob()` method.” On the other hand, in Oracle, “to write LOB (large objects such as CLOB and BLOB) data, the application must acquire a write lock on the LOB object. One way to accomplish this is through a `SELECT FOR UPDATE`. Also, disable autocommit mode.”

Setting Up the MySQL Database

In MySQL, updating existing CLOBs is simple; in fact, CLOBs are just long String objects. This shows how to set up the database:

```
mysql> use octopus;
Database changed
mysql> desc datafiles;
+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id         | int(11)       |      | PRI | 0        |      |
| fileName  | varchar(6)    | YES  |     | NULL     |      |
| fileBody  | text          | YES  |     | NULL     |      |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select id, filename, filebody
-> from datafiles where id=4000;
+-----+-----+-----+
| id | filename | filebody
+-----+-----+-----+
| 4000 | file40 | This is the content of file4000. |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> update datafiles
-> set filebody='My New CLOB Value...!!!'
-> where id=4000;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select id, filename, filebody
-> from datafiles where id=4000;
+-----+-----+-----+
| id | filename | filebody
+-----+-----+-----+
| 4000 | file40 | My New CLOB Value...!!! |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Creating the MySQL Servlet Interface

The servlet interface is as follows:

```
http://localhost:8000/octopus/servlet/UpdateMySQLClobServlet?
id=<id>&file=<file-as-URL>
```

Therefore, InsertClobToMySQLServlet has two parameters:

- id (the ID of file, which uniquely identifies record)
- file (the URL of file representing the CLOB; the servlet will open the URL, construct a CLOB, and update it into the CLOB column of the DataFiles table)

Let's update an existing record with the following data in the DataFiles table:

- id (4000)
- file (the URL of file: <http://www.geocities.com/mparsian/data/file500.txt>)

Figure 8-29 shows the content of the URL (<http://www.geocities.com/mparsian/data/file500.txt>).

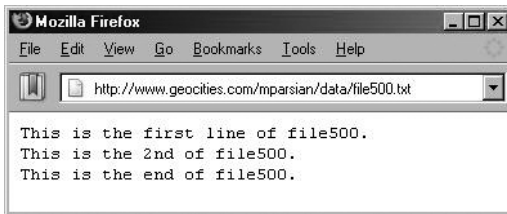


Figure 8-29. *Displaying a file using a servlet*

Therefore, the servlet call is as follows (but all in one line):

```
http://localhost:8000/octopus/servlet/UpdateMySQLClobServlet?
id=4000&file=http://www.geocities.com/mparsian/data/file500.txt
```

Updating an Existing CLOB Record

Figure 8-30 shows the results of updating an existing CLOB record.

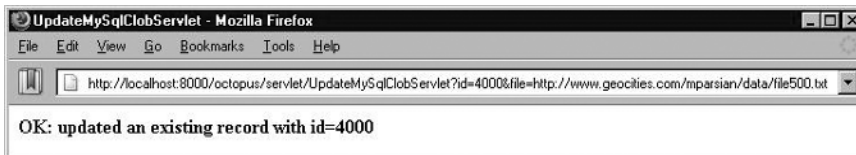


Figure 8-30. *Updating a CLOB using a servlet*

Viewing the Database Content After Insertion

This shows the database after the insertion:

```
mysql> select id, filename, filebody from datafiles where id=4000;
+-----+-----+-----+
| id   | filename | filebody |
+-----+-----+-----+
| 4000 | file40   | This is the first line of file500. |
|      |          | This is the 2nd of file500.       |
|      |          | This is the end of file500.       |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Using `DisplayMySQLClobServlet`, you can view the CLOB in a Web browser, as shown in Figure 8-31.



Figure 8-31. Viewing an updated CLOB using a servlet

Solution: UpdateMySQLClobServlet

The MySQL solution is identical to the Oracle solution with the exception of the `getConnection()` method, which returns a MySQL Connection object. You can download the complete MySQL solution from the book's Web site.

The following is `getConnection()` for the MySQL database:

```
public static Connection getConnection() throws Exception {
    String driver = "org.gjt.mm.mysql.Driver";
    String url = "jdbc:mysql://localhost/octopus";
    String username = "root";
    String password = "root";
    Class.forName(driver); // load MySQL driver
    return DriverManager.getConnection(url, username, password);
}
```

8-16. How Do You Delete an Existing CLOB of an Oracle Database Using a Servlet?

The SQL `DELETE` statement deletes rows in a table. The simple syntax is as follows:

```
DELETE FROM table_name
WHERE column_name_1 = some_value_1 and
      Column_name_2 = some_value_2 and ...
```

The goal is to delete an existing database record that has a CLOB column. You can do this by providing the primary key for the desired record (to be deleted). You may also delete the CLOB record using SQL's `LIKE` statement against the content of the CLOB (the body of the file), but this is not

recommended. (Most databases will not allow you to index the CLOBs; for example, MySQL allows you to index portions of CLOBs.) For solving this problem, you will use the `DataFiles` table (the `id` column is the primary key, and `fileBody` is the CLOB column). The servlet interface is as follows:

```
http://localhost:8000/octopus/servlet/DeleteClobFromOracleServlet?id=<id>
```

Therefore, `DeleteClobFromMySQLServlet` has only one parameter:

- `id` (the ID of CLOB, which is the primary key that identifies the record)

To delete an existing record with the ID of 500, issue this:

```
http://localhost:8000/octopus/servlet/DeleteClobFromOracleServlet?id=500
```

Viewing the Database Content Before Deletion

This is the database before the deletion:

```
SQL> desc DataFiles;
Name                               Null?    Type
-----
ID                                  NOT NULL NUMBER(38)
FILENAME                            VARCHAR2(20)
FILEBODY                             CLOB
```

```
SQL> select id, filename from dataFiles;
```

```

   ID  FILENAME
-----
 4000  file4000
 1000  file1
 2000  file2
 500   file500
```

Using a Servlet to Delete a Record (with CLOB)

Therefore, the servlet call is as follows:

```
http://localhost:8000/octopus/servlet/DeleteClobFromOracleServlet?id=500
```

Viewing the Actual Servlet Call for Deleting a Record (with CLOB)

Figure 8-32 shows the result of deleting a record.



Figure 8-32. *Deleting a CLOB using a servlet*

Viewing the Database Content After Deletion

This shows the database after the deletion:

```
SQL> select id, filename from dataFiles;
```

```

      ID  FILENAME
-----  -
      4000 file4000
      1000 file1
      2000 file2

```

Solution: DeleteClobFromOracleServlet

This shows the DeleteClobFromOracleServlet solution:

```

import java.io.*;
import java.sql.*;

import javax.servlet.*;
import javax.servlet.http.*;

import jcb.db.DatabaseUtil;
import jcb.util.IOUtil;

public class DeleteClobFromOracleServlet extends HttpServlet {

    private static final String DELETE_CLOB_RECORD =
        "delete from DataFiles where id = ?";

    public static Connection getConnection() throws Exception {...}
    public void doGet(...) {...}
    public void doPost(...) {...}
}

```

getConnection()

This shows getConnection():

```

public static Connection getConnection() throws Exception {
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@matrix:1521:caspiant";
    String username = "mp";
    String password = "mp2";
    Class.forName(driver); // load Oracle driver
    return DriverManager.getConnection(url, username, password);
}

```

doGet()

This shows doGet():

```

public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
    throws IOException, ServletException {

    Connection conn = null;
    PreparedStatement pstmt = null;
    String id = request.getParameter("id").trim();
    ServletOutputStream out = response.getOutputStream();
    response.setContentType("text/html");
    out.println("<html><head><title>Delete CLOB Record</title></head>");
}

```

```

try {
    conn = getConnection();
    pstmt = conn.prepareStatement(DELETE_CLOB_RECORD);
    pstmt.setString(1, id);
    pstmt.executeUpdate();
    pstmt.executeUpdate();
    out.println("<body><h4>deleted CLOB record with id="
        +id+"</h4></body></html>");
}
catch (Exception e) {
    out.println("<body><h4>Error="+e.getMessage()+"</h4></body></html>");
}
finally {
    DatabaseUtil.close(pstmt);
    DatabaseUtil.close(conn);
}
}

```

doPost()

This shows doPost():

```

public void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
    doGet(request, response);
}

```

8-17. How Do You Delete an Existing CLOB of an MySQL Database Using a Servlet?

The goal is to delete an existing database record, which has a CLOB column. Basically, you can do this by providing the primary key for the desired record (to be deleted). For solving this problem, you will use the DataFiles table (the id column is the primary key, and fileBody is the CLOB column). The servlet interface is as follows:

```
http://localhost:8000/octopus/servlet/DeleteClobFromMySQLServlet?id=<id>
```

Therefore, DeleteClobFromMySQLServlet has only one parameter:

- id (the ID of CLOB, which is the primary key that identifies the record)

To delete an existing record with the ID of 500, you will issue the following servlet call:

```
http://localhost:8000/octopus/servlet/DeleteClobFromMySQLServlet?id=500
```

Viewing the Database Content Before Deletion

This is the database before the deletion:

```

mysql> use octopus;
Database changed
mysql> desc datafiles;

```

```

+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)       |      | PRI | 0        |       |
| fileName  | varchar(6)    | YES  |     | NULL     |       |
| fileBody  | text          | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

```

mysql> select id, filename from datafiles;
+-----+-----+
| id | filename |
+-----+-----+
| 10 | file1    |
| 20 | file2    |
+-----+-----+
2 rows in set (0.05 sec)

```

Using a Servlet to Delete a Record (with CLOB)

Therefore, the servlet call is as follows (deleting an existing record with the ID of 10):

```
http://localhost:8000/octopus/servlet/DeleteClobFromMySQLServlet?id=10
```

Viewing the Actual Servlet Call for Deleting a Record (with CLOB)

Figure 8-33 shows the result of deleting a record.



Figure 8-33. *Deleting a CLOB using a servlet*

Viewing the Database Content After Deletion

This is the database after the deletion:

```

mysql> select id, filename from datafiles;
+-----+-----+
| id | filename |
+-----+-----+
| 20 | file2    |
+-----+-----+
1 row in set (0.00 sec)

```

Solution: DeleteClobFromMySQLServlet

The MySQL solution is identical to the Oracle solution with the exception of `getConnection()`, which returns a `MySQLConnection` object. You can download the complete MySQL solution from the book's Web site.

The following is `getConnection()` for the MySQL database:

```
public static Connection getConnection() throws Exception {
    String driver = "org.gjt.mm.mysql.Driver";
    String url = "jdbc:mysql://localhost/octopus";
    String username = "root";
    String password = "root";
    Class.forName(driver); // load MySQL driver
    return DriverManager.getConnection(url, username, password);
}
```

8-18. Should You Use `java.lang.String` or `java.sql.Clob`? Which Has the Best Performance?

If you have the choice of manipulating large text data (a large text column of a record such as the CLOB data type), should you use `java.lang.String` or `java.sql.Clob`? Which has the best performance?

For better performance, you should use `java.sql.Clob`, since it does not extract any data from the database until you explicitly ask it to (by invoking the `getAsciiStream()` or `getCharacterStream()` method). The JDBC data type `java.sql.Clob` wraps a database locator (which is essentially a pointer to char). That pointer is a rather large number (between 32 and 256 bits in size), but the effort to extract it from the database is insignificant next to extracting the full CLOB content. For insertion into the database, you should use `java.lang.String` since data has not been uploaded to the database yet. Therefore, use the `java.sql.Clob` object only for data extraction (whenever possible).