

Working with Large Objects (BLOB And CLOB)

=====

Sometimes as the part of programming requirement, we have to insert and retrieve large files like images, video files, audio files, resume etc wrt database.

Eg: upload image in matrimonial web sites
upload resume in job related web sites

To store and retrieve large information we should go for Large Objects (LOBs).

There are 2 types of Large Objects.

1. Binary Large Object (BLOB)
2. Character Large Object (CLOB)

1) Binary Large Object (BLOB)

A BLOB is a collection of binary data stored as a single entity in the database.

BLOB type objects can be images, video files, audio files etc..

BLOB datatype can store maximum of "4GB" binary data.

2) CLOB (Character Large Objects):

A CLOB is a collection of Character data stored as a single entity in the database.

CLOB can be used to store large text documents (may plain text or xml documents)

CLOB Type can store maximum of 4GB data.

Eg: resume.txt

Steps to insert BLOB type into database:

1. create a table in the database which can accept BLOB type data.
create table persons(name varchar2(10), image BLOB);
2. Represent image file in the form of Java File object.
File f = new File("sachin.jpg");
3. Create FileInputStream to read binary data represented by image file
FileInputStream fis = new FileInputStream(f)
4. Create PreparedStatement with insert query.
PreparedStatement pst = con.prepareStatement("insert into persons values(?,?)");
5. Set values to positional parameters.
pst.setString(1, "sachin");

To set values to BLOB datatype, we can use the following method: setBinaryStream()

public void setBinaryStream(int index, InputStream is)

public void setBinaryStream(int index, InputStream is, int length)

public void setBinaryStream(int index, InputStream is, long length)

6. execute sql query
pst.executeUpdate();

Steps to Retrieve BLOB type from Database

=====

1. Prepare ResultSet object with BLOB type
ResultSet rs = st.executeQuery("select * from persons");
2. Read Normal data from ResultSet
String name = rs.getString(1);
3. Get InputStream to read binary data from ResultSet
InputStream is = rs.getBinaryStream(2);

4. Prepare target resource to hold BLOB data by using FileOutputStream
FileOutputStream fos = new FOS("katrina_new.jpg");

5. Read Binary Data from InputStream and write that Binary data to output Stream.

```
int i=is.read();  
while(i!=-1)  
{  
    fos.write(i);  
    is.read();  
}
```

```
or  
byte[] b= new byte[2048];  
while(is.read(b) > 0){  
    fos.write(b);  
}
```

The diagram illustrates the process of inserting a BLOB into a database. It shows a file 'sachin.jpg' being read by a FileInputStream, then processed by a Driver and DBE to insert into a table 'auto persons'.

File Input: A file named 'sachin.jpg' is read by a `FileInputStream (fis)`.

PreparedStatement: The data is prepared using a `PreparedStatement` with the following SQL statement:

```
PreparedStatement
pstmt.setInt(1,name);
pstmt.setBinaryStream(2,fis);
```

Driver and DBE: The prepared statement is executed by a `Driver` and a `DBE` (Database Environment).

Database Table: The data is inserted into a table named 'auto persons' with the following structure:

id	name	image
		sachin.jpg

The table schema is defined as:

```
int varchar BLOB
```

The table structure is summarized in the following table:

Field	Type
id	int NOT NULL
name	varchar(20) NULL
image	longblob NULL

The diagram illustrates the process of reading a BLOB from a MySQL database and saving it as a file in Java.

MySQL Database: A database named 'octbatch' contains a table 'persons'. The table has columns 'id', 'name', and 'image' (BLOB). The 'image' column contains a BLOB value represented by a female symbol (♀).

ResultSet: The data is retrieved into a 'ResultSet' object. The 'image' column is accessed via an 'InputStream' (indicated by a red arrow).


Java Code: The following code snippet is shown:

```
int id      = resultSet.getInt(1);
String name = resultSet.getString(2);
InputStream is = resultSet.getBinaryStream(3);
```

The 'InputStream' object 'is' is used to read the BLOB data. The data is then copied from the 'is' stream to a 'fos' (FileOutputStream) stream, which saves the data as a file named 'copied_image.jpg' in the directory 'D:\\images'.

```
int i = is.read();
while (i != -1) {
    fos.write(i);
    i = is.read();
}

byte[] b = new byte[1024];
while (is.read(b) > 0)
{
    fos.write(b);
}
```

 apache
commons-io.jar
IOUtils.copy(is, fos);

```

field      Type
1          int NOT NULL
name       varchar(20) NULL
history    longtext NULL

stmt.setCharacterStream(2, new FileReader(new File(pdfLoc)));
void setCharacterStream(int parameterIndex,Reader reader) throws SQLException;
JDBCConnection
JRE System Library [JavaSE-1.8]
src

```

```
Reader reader = resultSet.getCharacterStream(3);
File file = new File("history_copied.txt");
FileWriter writer = new FileWriter(file);
IOUtils.copy(reader,writer);
```

Comments in SQL are of 2 types

- a. -- single line comment
- b. /* multiline comment */

eg: select count(*) from users where name='sachin' and password='tendulkar'; => no problem with inputs

```

Statement Object
select count(*) from users where name='sachin' and password='tendulkar';
    ↑
    |
    | 1. Query compiled('--' treated as comments)
    | 2. Query executed('--' wont be in the execution phase)
    |
    ↓
SQLInjection → end user manipulated the query with special syntax.
                (Statement)

Prepared Statement Object
select count(*) from users where name = ? and password = ?
    ↑
    |
    | 1. Query compiled('--' wont come into picture)
    | 2. Query executed select count(*)
    |
    ↓
SQLInjection → end user manipulated the query with special syntax.
                X
            resolved through Prepared Statement
  
```

The diagram illustrates two scenarios for SQL queries with comments. In the first scenario, a query like `select count(*) from users where name='sachin' and password='tendulkar';` is shown. A red box highlights the closing quote of the password string, and an arrow points to it with the label "comment". Below this, a list indicates: "1. Query compiled('--' treated as comments)" and "2. Query executed('--' wont be in the execution phase)". This leads to a "Statement Object". In the second scenario, a similar query is shown but with placeholders: `select count(*) from users where name = ? and password = ?`. Here, a red box highlights the closing quote of the password string, and an arrow points to it with the label "comment". Below this, a list indicates: "1. Query compiled('--' wont come into picture)" and "2. Query executed select count(*)". This leads to a "Prepared Statement Object". The diagram also shows a flowchart where a user input "sachin" and "tendulkar" is processed by a database engine, resulting in a "values" object containing the strings "sachin" and "tendulkar".

