

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 box labeled 'D://images' contains a file named 'sachin.jpg' represented by a red female symbol. Below it, a 'FileInputStream (fis)' is shown.

PreparedStatement: A box contains the following code:

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

Driver and DBE: The flow continues through a 'Driver' box and a 'DBE' box.

Database Table: A cylinder represents the database. Inside, a table named 'auto persons' is shown with columns 'id', 'name', and 'image'. The 'image' column contains the file 'sachin.jpg' and a red female symbol. Below the table, the column types are listed: 'int varchar BLOB'.

Field Type Table: A table below the database cylinder shows the field types for the 'auto persons' 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. On the left, a MySQL database contains a table named 'persons' with columns 'id' and 'name'. A row with 'id' 3 and 'name' 'foso' is highlighted. The 'foso' value is associated with a BLOB. An arrow points from this BLOB to a 'ResultSet' object. From the 'ResultSet', an 'InputStream' is obtained, represented by a red arrow labeled '1010101001' and 'is'. This stream is then used in a Java application to 'copy from is to fos' (FileOutputStream), resulting in a file named 'copied_image.jpg' in the 'D:\\images' directory. Below the diagram, the corresponding Java code is shown:

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

```
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);

```

    void setCharacterStream(int parameterIndex,Reader reader) throws SQLException;

    stmt.setCharacterStream(2, new FileReader(new File(pdfLoc)));

```

```
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
    */
```

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)

comment

SQLInjection → end user manipulated the query with special syntax.

PreparedStatement Object

```
select count(*) from users where name = ? and password = ?
```

1. Query compiled ('-- wont come into picture)
2. Query executed select count(*)

from users where

name = 'sachin'--
and
password = 'tendulkar'

values

resolved through PreparedStatement

