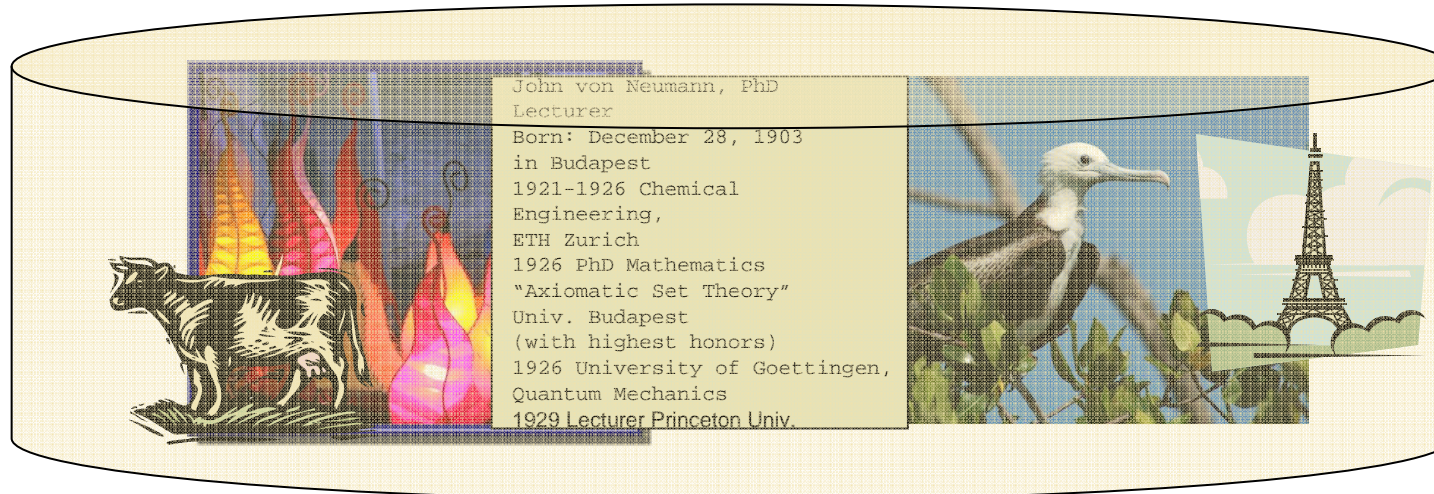


Large Objects in Databases



Dr. Michael Emmerich
(acknowledgement: Dr. W. Obermaier)

Large Objects

- Large Objects are Data-entries with variable size (up to Gigabytes!)
- Modern Applications demand beside the storage of conventional data also the storage of large objects
- Examples are
 - Multimedia Authoring Systems (Scripts, Sketches, Photos, Audio and Videosequences)
 - CAD/CAM Systems: Drawings, Animations
 - Scientific Image/Sound Databases
 - Software Engineering Repositories: Diagrams, Source Code, Binaries



Why large objects in databases

- Integrated Data management
 - Data that belongs together is stored together
 - Unified and controlled data access
- Functionality of database can be utilized
 - Multiple-user control, three-tier architectures, recovery control, buffer management
- Specific multi-media database functions can be utilized
 - Retrieval options (search in contents of large object)
 - Indexing of content for multimedia object
 - Content specific functions provided by DBMS



Large objects (LOB) in Oracle

Oracle supports the following LOB types:

- BLOB: binary LOB; Binary file, up to 4GB
- CLOB: Characted LOB ASCII file, up to 4GB
- NCLOB: national character LOB
 - specific for font set for a language
- BFILE: binary file; reference on file outside of document;only read access



Characteristics of LOBs

- An BLOB in Oracle consist of two parts:
 - LOB data
 - LOB locator (pointer to LOB data)
- The LOB data is only stored up to a size of 3964 bytes directly in a table
- Tables that are stored in cluster format are not allowed to include any BLOB information
- Queries with GROUP BY do not allow for any LOBs, nor are LOBs allowed as search keys for indexes or join attributes. *Why?*



Example: Table with LOBs

- In a table of employees a picture and a curriculum vitae is added for each entry

```
CREATE TABLE Staff;  
name VARCHAR2(40) PRIMARY KEY;  
room CHAR(6);  
tel CHAR(4);  
eMail VARCHAR2(40) UNIQUE NOT NULL  
position CHAR(10) NOT NULL;  
image BLOB;  
curVitae CLOB;  
CHECK (position IN  
( 'head' , 'professor' , 'lecturer' , 'technician' , 'secretary' ) );
```



John von Neumann, PhD
Lecturer
Born: December 28, 1903
in Budapest
1921-1926 Chemical
Engineering,
ETH Zurich
1926 PhD Mathematics
"Axiomatic Set Theory"
Univ. Budapest
(with highest honors)
1926 University of Goettingen,
Quantum Mechanics
1929 Lecturer Princeton Univ.

Initialization of LOBs

- LOBs need to be initialized
- In Oracle:
 - `EMPTY_BLOB()`: Instantiate an empty BLOB
 - `EMPTY_BLOB()`: Instantiate an empty CLOB or NCLOB
 - `BFILENAME(<dir>, <file>)`; for initialisation of a reference to file in Oracle directory <dir>
- Example

```
INSERT INTO Staff VALUES ('Michael Emmerich', NULL,  
NULL, 'michael.emmerich', lecturer, EMPTY_BLOB(),  
EMPTY_BLOB());
```



Manipulation of LOBs

- The following operations are possible with LOBs:
 - Load files into BLOBS, CLOBS, NCLOBS
 - Read data from BLOBS, NCLOBS and BFILES
 - partial update of CLOBS, BLOBS
- These functions are usually only available via APIs like JDBC, not via interactive SQL



LOBs in JDBC

- The functionality for the handling of LOBs is not part of standard JDBC
- The Oracle drivers support LOBs in the packages: `oracle.sql` and `oracle.jdbc`
- Important class from `oracle.sql`:
 - BLOB (implements `oracle.jdbc2.blob`): BLOB Locator
 - CLOB (implements `oracle.jdbc.clob`): CLOB Locator
 - BFILE: BFILE Locator
- Important classes from `oracle.jdbc.driver`: `OracleResultSet` (implements `java.sql.ResultSet` and supports handling of LOBs in the result of a query)



LOBs selection

- SELECT statements select **locators** instead of LOBs
- The user or the application program must not distinguish if the data contained in the LOB is stored in the table or in an extra memory space
- The LOB locators allows to open Java Streams on the LOB data
- Read/Write of LOB data is done via Java streams



Example: LOBs selection

```
BLOB image;
CLOB curVitae;

< establish connection 'con' >

Statement = con.createStatement();
String cmd = "SELECT image, curVitae " +
             "FROM Staff " +
             "WHERE name = 'Michael.Emmerich'";
OracleResultSet rs =
(OracleResultSet(stmt.executeQuery(cmd)));
if (rs.next()) {
    image = rs.getBLOB("image");
    curVitae = rs.getCLOB("curVitae");
    < processing >
}
```



Read and Write of BLOBs

- To write a BLOB, the method `getBinaryOutputStream()` can open an `OutputStream` on the BLOB
 - The Data is written with `write()` Methods
 - To read a BLOB, an `InputStream` can be opened with `getBinaryStream()`
 - Diverse `read()` methods can be used to read objects
- >> examples follow ...



Read BLOB from DB and write it to file

```
conn = getConnection();
Statement st = conn.createStatement();
ResultSet rs =
    st.executeQuery("SELECT IMAGE FROM IMAGES");
while (rs.next()) {
    Blob blob = rs.getBlob(1);
    InputStream is = blob.getBinaryStream();
    FileOutputStream fos = null;
    fos = new FileOutputStream("c:/TEMP/" + fileName);
    byte[] data = new byte[1024];
    int i = 0;
    while ((i = is.read(data)) != -1)
    {
        fos.write(data, 0, i);
    }
}
```



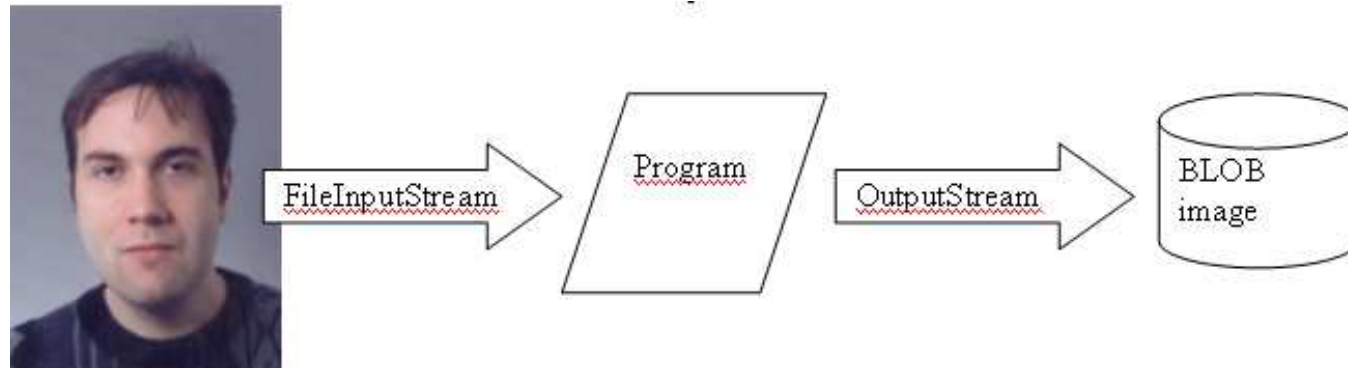
Write BLOB from file to DB

```
Statement st = conn.createStatement();
ResultSet rs =
st.executeQuery("SELECT IMAGE FROM IMAGES FOR UPDATE");

while (rs.next()) {
    Blob blob = rs.getBlob(1);
    System.out.println(blob);
    OutputStream os = blob.setBinaryStream(1);
    FileInputStream fis = null;
    fis = new FileInputStream("c:/TEMP/" + fileName);
    byte[] data = new byte[1];
    int i;
    while ((i = fis.read(data)) != -1) {
        os.write(data, 0, i);
    }
    os.close();
    break;
}
```

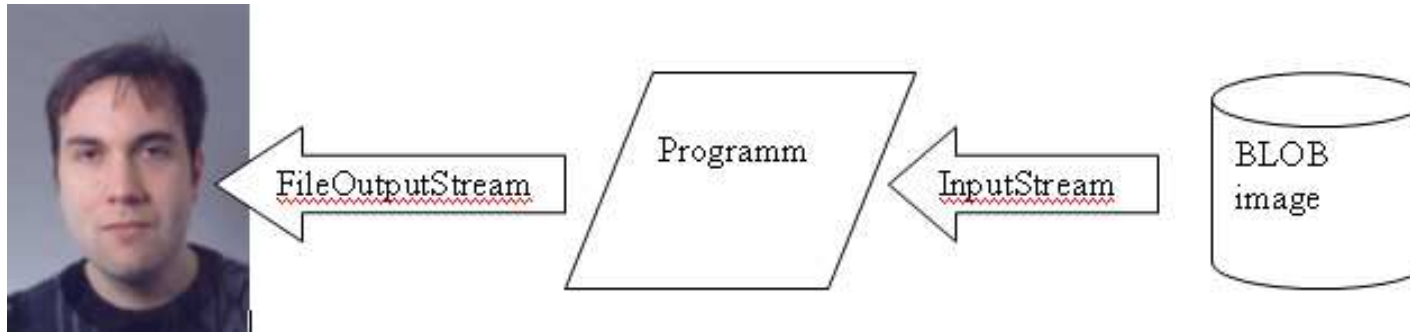


Example: Write a BLOB



```
< select a BLOB-Locator in 'image'>
...
FileInputStream in = new FileInputStream("Emmerich.jpg");
OutputStream out = image.getBinaryOutputStream(),
int chunk = image.getChunkSize(),
byte[] buffer = new byte(chunk),
while ((length = in.read(buffer)) != -1)
    out.write(buffer, 0, length)
in.close();
out.close();
```

Example: Read a BLOB



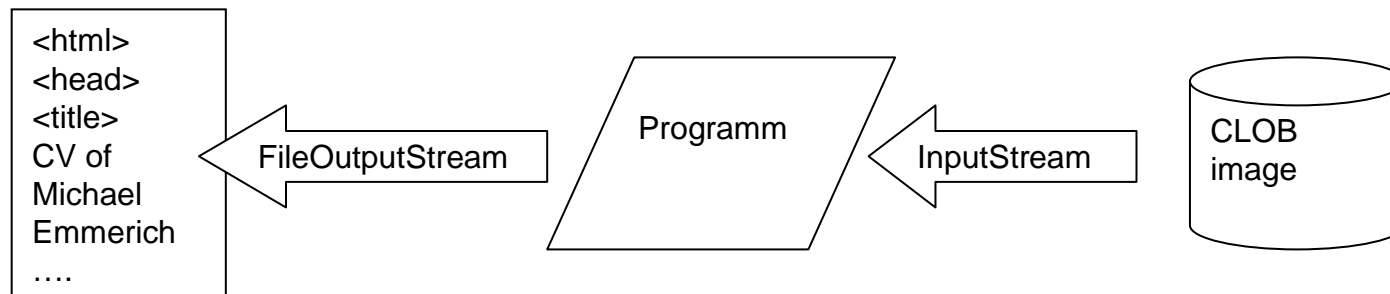
```
< select a BLOB-Locator in 'image'>
...
FileInputStream out = new FileOutputStream("Emmerich.jpg");
InputStream in = image.getBinaryInputStream();
int chunk = image.getChunkSize();
byte[] buffer = new byte(chunk);
while ((length = in.read(buffer)) != -1)
    out.write(buffer, 0, length)
in.close();
out.close();
```

Read and Write of CLOBS

- To read a CLOB the method `getCharacterOutputStream()` and a `Writer` on the CLOB can be opened
- The data is written with the `write` method
- To read a CLOB the method `getCharacterStream()` opens a `Reader` on the CLOB
- The various objects are read with the `read` method

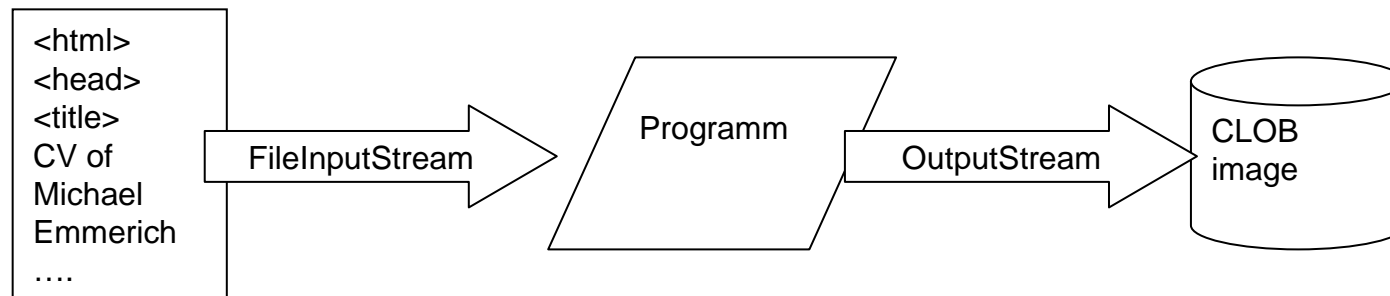


Example: Write a CLOB



```
< select a BLOB-Locator in 'text'>
...
FileInputStream out = new FileOutputStream("Emmerich.jpg");
InputStream in = text.getCharacterInputStream();
int chunk = image.getChunkSize();
byte[] buffer = new byte(chunk);
while ((length = in.read(buffer)) != -1)
    out.write(buffer, 0, length);
in.close();
out.close(),
```

Write a CLOB



```
< select a BLOB-Locator in 'text'>
...
FileInputStream out = new
FileOutputStream("Emmerich.txt");
InputStream in = image.getCharacterStream();
int chunk = text.getChunkSize();
byte[] buffer = new byte(chunk);
while ((length = in.read(buffer)) != -1)
    out.write(buffer, 0, length);}
in.close();
out.close(),
```

Read BFILEs

- **Read BFILEs**
- BFILEs can only be read. Data files can neither be created, nor can data be added.
- The corresponding JDBC data type is BFILE. With `getBFILE()` a BFILE Locator can be extracted
- For producing a BFILE, this file has to be created by the user

http://www.zope.org/Members/peterb/oracle_lobs



Summary

- CLOBs and BLOBs are LOBs are used to store large objects (LOBs) in data bases
- A restricted set of operations can be performed with LOBs
- BFILES provide alternative (point to files), but user needs to make sure that referential integrity is preserved (files must exist)
- Using LOBs has the advantage that all memorized data is within the database
 - Advantages of databases can be used: concurrency, backups, etc.
 - Data-integrity can be preserved as LOBS are related to the objects they belong to
- Access to LOBS is typically via application (API)
- Typical operation: Load content of file into LOB, or retrieve file from LOB (see examples)

