FuzzyPhoto

AHRC AH/J004367/1

Work package report 3: Data Ingestion and Warehouse

Date of delivery: 29/10/13

Abstract

FuzzyPhoto is a two year AHRC funded research project that is developing and testing computer-based "finding aids" that can recommend potential matches between historical photographic exhibition catalogue records and images of photographs that appear in online collections, even where there is not a precise match. Work package 3 covers the work required to import the meta-data from the project partners into a unified data model from which the links can be harvested.

Distribution Type: Internal

Author: Dr Alexander von Lünen

Keywords: FuzzyPhoto, metadata, historic photographic collections, museums, libraries, catalogue data, database technology, data import procedures, LIDO, data warehouse, data ingestion

Contents

Summary

This document details the process of importing the project partners' data into the (temporary) tables of the FuzzyPhoto database. Data was usually delivered as CSV or XML files, and each of these

formats required different processes. Each dataset is quite unique, so it has not been possible to adopt a unified approach to importing the data. In the future, it would be advisable to ask the partners for a standardized data format.

The individual datasets were first imported into MySQL as specific tables, after which some data cleaning was carried out and then the data was transferred into the chosen metadata schema (LIDO), where some more cleaning was conducted, (see below).

Status

Outputs of this workpackage comprise:

1. This report
2. MySQL temporary tables
3. MySQL LIDO data warehouse comprising:

Partner	Records before cleaning	Records after cleaning
Birmingham City Library	5,513	5,455
British Library	28,974	28,925
CultureGrid	172,148	171,840
ERPS	34,197	34,197
Metropolitan Museum	9,526	9,526
PEIB	20,453	20,453
Musee d'Orsay	46,229	46,228
National Media Museum	8,380	8,380
National Museums Scotland	14,915	14,883
St Andrews University Library	18,620	18,604
Library of Congress	875,267	875,267
Brooklyn Museum	2,352	2,352
National Archives	73,187	71,958
Victoria and Albert Museum	101,538	98,598
		1,406,666

The elapsed time for this work package was 9 months. The resources required to complete this work package were 70 person-days (partner liaison, etc. not included).

Objectives

To import the catalogue data from the project partners – which was delivered in diverse formats – into first a temporary relational database and then into a unified relational database based on the LIDO schema.[1]

The data from the project partners turned out to be quite heterogeneous and required a good deal of work to unify the various data sets into one schema, necessary to generate the links between the different data. The heterogeneity consists of very different structures of the partners' data, usually introduced by the respective record management software (RMS). Each project partner uses a different RMS with varying export capabilities, limiting the extent to which exporting the data can be customized and therefore made compliant to a specific data model. The decision to run a specific RMS is often driven by legacy issues, i.e. availability of a certain RMS package at a certain point which was then populated with the record data. Switching to a different RMS would be too costly for partners and is thus not an option. This meant that few of the data sets could be tailored to comply to a given schema. Therefore, rather than loading the data directly into the chosen metadata (LIDO), temporary tables were created to collate the data and run “clean up” scripts on it.

It was originally considered to develop a batch loader to achieve this, but delays in the data submission by some partners and the very diverse nature of the data made this enterprise not very feasible. The process of importing and cleaning up the data, on the other hand, yielded a very good impression of what would be involved in creating such a tool.

1 http://www.lido-schema.org

The metadata schema

It was first considered to use CIDOC-CRM as an ontology-based data model for the database. However this turned out to be more a hindrance to the project rather than a benefit. CRM is very complex and hard to keep minimal, i.e. there are a lot of mandatory data fields whereas there are very few with LIDO. Given that the meta-data delivered by the partners was so diverse, it would have been hard (if not impossible) to comply with CRM, and would have been necessary to edit the partner's data to a greater extent (see below for a discussion LIDO vs CIDOC CRM).

The complexity of CRM can also be a bit overwhelming, with some object types almost getting into philosophical realms. Since the fuzzy logic software developed by the project does not take the extra features offered by an ontology into account (such as a dedicated reasoner), it wouldn't have been very feasible to use CRM. It was therefore decided that LIDO offers the best compromise in terms of expressivity, flexibility and implementability.

LIDO was devised in 2010 purely as an exchange format to transfer data independently from the RMS being used. Several major players were involved in its design, such as CDWA Lite, Athena, CIDOC-CRM, Spectrum (CollectionsTrust, UK), MuseumDat (Germany). It has established itself as a standard in the GLAM-community since its inception. One of its major benefits is its versatility, i.e. that it allows flexibility without sacrificing expressivity.

LIDO is an event-based data model, i.e. its main concept rests on the “things” that can be “done” to an object, such as creation, acquisition, restoration, exhibition, etc. This allows for very detailed metadata to be stored and the life cycles of an object meticulously recorded.

“Harvesting Formats” vs Ontologies

A line must be drawn between so-called “harvesting formats” (HF) and full-fledged ontologies. HF in general are formats for publishing and interchanging data, independent from a specific database software etc. Ontologies, on the other hand, also describe the data so being published, i.e. ontologies operate at the conceptual level. Simply put, HF and ontologies differ in regard to the amount and level of metadata they incorporate. HF, as mentioned, is about publishing collection records, so metadata usually concerns things like data of creation and creator of an object, for example. Since ontologies were devised in the context of Knowledge Representation Systems, they must provide enough metadata for automated reasoning. For example, the LIDO format, defines a field for the gender of an actor (such as creator or owner of an artefact). For an HF it is good enough to have different identifiers (such as male, female, unknown) to operate, whereas in an ontology there must also be rules about the “nature” of these different genders. “Nature” here referring to a set of logical constraints; for example, that “male” and “female” are mutually exclusive, i.e. when an actor is labelled as being “male”, the “reasoner” (a piece of software that evaluates the ontology) can infer that the actor is not “female”. This may seem trivial to humans, but in an ontology rules such as these need to be defined to allow the reasoner to work efficiently; the point here being that the strength of ontologies is to allow transitive queries. If, for example, I would query for female artists in the database, but not all of the entries have their gender field set, the only fix would be to update the gender field before I could run the query. In an ontology, however, I could define a set of rules that would let the reasoner infer that the record being looked at must be a female person, e.g. by having occupation “actress” and there being a rule in the ontology that an occupation of type “actress” refers to a female person.

So, to summarize, GLAMs might quite likely have data already that could be described as “harvesting format”, usually the collection records. But it is perhaps no so likely that they have a full-blown ontology for their collection. Luckily, standards exist for both (HF and ontologies), and the the following sections will discuss the best candidates and how they could be utilized in the FuzyPhoto project.

Harvesting Formats

There exists a number of HF; the main reason being that metadata about collection items have been recorded for a long time now, starting with paper cards in filing cabinets, and XML has been around for some time now acting as lingua franca of structured data exchange format.

Many different HF have been designed in the past years, usually along national and topical interests (i.e. depending on the type of collection). There seems to be a clear trend, however, to find an internationally agreeable standard. This has resulted in the LIDO (Lightweight Information Describing Objects, `http://www.lido-schema.org`) XML schema. Other, more national, schemas such as Museumdat by the German Museums Association (`http://www.museumdat.org`) have voiced their support for LIDO. It thus seems to be a promising candidate for getting an exchange format for GLAM data.

LIDO only defines a minimum set of mandatory data fields to be populated, making it rather flexible. Given that is has been developed by several museum organizations, it should be quite close to the kind of data that is frequently encountered in the GLAM sector.

Ontologies

The selection of ontologies in the GLAM sector is much more limited. For years Dublin Core had been the least common denominator, with various national bodies (LOC, DNB, etc) rolling their own standard. As of late, CIDOC CRM strives to be the gold standard in the cultural heritage sector. While quite complex, it has been closely modelled after the collection management philosophies in the GLAM area and is experiencing ever more support from it. CIDOC CRM offers a multitude of classes to describe collections and related concepts. It thus makes a good candidate for storing the project data on the search engine side.

Strategy for the project

That being said, it should be obvious that many of the project partners in the GLAM sector may very well have data in their respective proprietary collection managements systems that could be easily mapped (i.e. exported/converted) to LIDO, while it is rather unlikely that they have sufficient metadata for CIDOC CRM. There are, however, tools and manuals on the CIDOC website to transfer LIDO (or general XML data) to CIDOC CRM, so the door is not closed on that one.

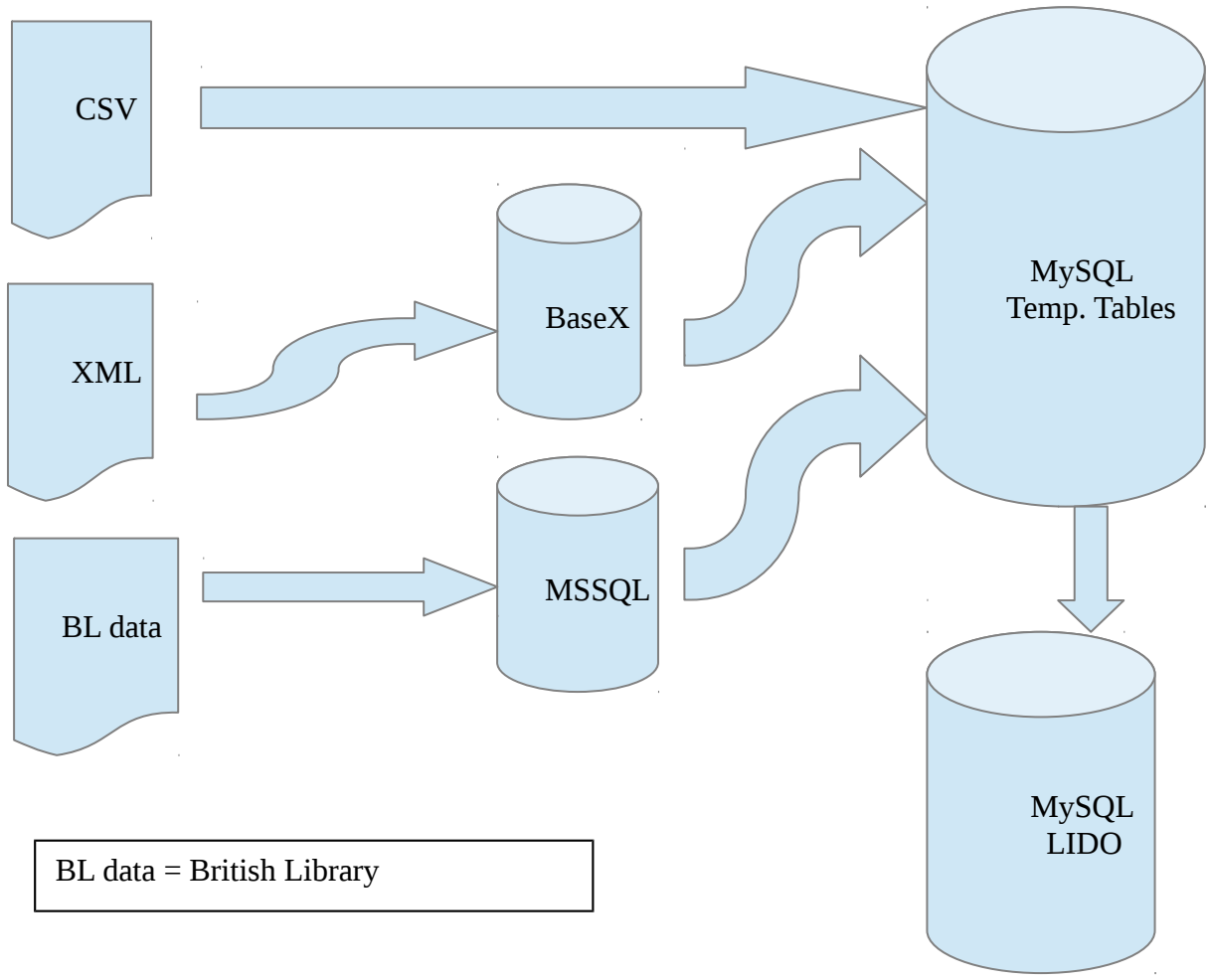
It thus makes sense to agree that the LIDO format is used as an interchange format within the FuzzyPhoto team, i.e. the project partners will deliver their data in some kind of structured data (possibly LIDO, but other formats are permissible) to the team, which will convert it to the LIDO format in order to unify the different data sets.

Requirements

As the data from the partners' is very diverse, a range of software tools were required. For XML data, an XML data store and XQuery queries were readily utilized (see below). For tabular data (CSV), the import facilities of MySQL usually sufficed to handle it. After the data had been imported, “cleaning up” the data – such as eliminating duplicates or spelling variants – was done chiefly with SQL queries with some user-defined functions (UDF) found on the Internet. Some operations, however, were too complex to be handled in SQL, so Java and Python scripts were used externally.

The software dependencies were thus:

- BaseX: as XML data store
- MySQL: as database, both for temporary tables and the LIDO schema
- LibreOffice Base: to aid in importing some of the data
- LibreOffice Calc: dto.
- MS SQL Server Express 2008: for the BL data
- MS SQL Server Express 2008 client library: for transferring BL data to MySQL
- MySQL Workbench for Windows: dto.
- Python
- Java



General Issues

The schema name “fuzzyphoto” was used to hold the partners’ data in the MySQL database. This schema served as temporary data store from which data could be transferred into the LIDO schema, which is held in the schema named “mydb” (this name was assigned by the ERM modeller that comes with the MySQL workbench and that was used to model the LIDO XML schema into a relational model).

The data from the partners had some issues that made it hard to clearly distinguish events and actors etc. For instance, in many data sets it isn't always obvious whether the date specified refers to the date of the shoot or the date of the print (and maybe this is difficult to establish from the source). Since “Shoot” and “Print” are two events in the LIDO model, it would be relevant to know the difference, while it should be ok to neglect this for the fuzzy logic algorithms.

Another obstacle was that field names in the partner's data were not consistent, both across the data sets and at times even within the same data set. For example, the sample data the V&A provided had different field names than the final data set we received. It would be advisable to prescribe for the partners the names of the fields they have to use in their data set so that a batch loader can just pick them up, rather than having to write a configuration file that labels fields correctly.

Importing CSV files

Importing CSV into MySQL is very straight-forward, using the load data command within the `mysql` shell. However, the table the data is imported into needs to be created first. For example, to load the data from the National Museum of Scotland one would create the table in `mysql` first:

```
create table nms (id varchar(100), description text, type
varchar(100));
```

Then, the command to load the data would be issued (make sure to get the field delimiters right, these are just the ones I have been using):

```
load data local infile 'nms.csv' into table nms fields terminated
by '\t' enclosed by '”' lines terminated by '\n' (id, description,
type);
```

The order of the columns need to be specified in the order they appear in the CSV file. Also, types need to be specified correctly in the create statement, or the data won't be imported. It is advisable to create all columns as string types (i.e. either varchar or text), and convert them them afterwards, as typos etc. might prevent import. (Note: in many cases I used the vertical bar character “|” as field separator, as this is far less ambiguous).

NB there is also a command line tool named `mysqlimport`, which does pretty much the same job, but you don't have to log in to the `mysql` shell. The syntax for that would be:

```
mysqlimport --fields-terminated-by=”\t” --lines-terminated-by=”\n”
-p -u {username} -v -L {database_name} {filename}
```

One would still need to create the tables first, though, so using the mysql shell might be handier. On the other hand, mysqlimport is much faster than the load data command, so for large data files this will be much more efficient. Note that there is no option for mysqlimport to specify the name of the

table in the database the data is supposed to be loaded into. The tool assumes the file name is the same as the table name, and your only option would be to rename either the file name or the table's name within the database.

Importing XML files

Importing XML proved to be a lot more complicated, as the schemas were quite diverse and each data set required a different approach. After trying XSLT scripts, it was decided to use a XML store, BaseX (`http://www.basex.org`), to do the transfer. The main reason is that BaseX provides a GUI and visualizations, making rapid prototyping much easier.

As a first step the XML files are loaded into BaseX. BaseX has the option to do bulk imports, so that when the data comes spliced up in several files they can be easily merged within BaseX. The software also offers different visualizations, such as a tree view of the XML data, which helps analyzing it.

BaseX has an XQuery shell integrated. After some experimenting, it was decided to create individual XQuery queries for every data set that would generate SQL Insert statements, which then can be run inside MySQL (there are XQuery software libraries that allow JDBC connections, so that the data could be exported to the MySQL database directly; however, these libraries were only available as commercial versions).

In terms of modelling the XML data as relational database tables, every nested element (i.e. sub-branch of the higher-up nodes) in the XML document tree were modelled as their own entity/table, linked by the object Id of the top node, if the analysis showed that they could have more than one entry.

For example, the V&A data has – among others – an object for dimensions:

```
<dimensions>
   <part name="(image only)">
      <dimension field="Height" unit="cm">40.9</dimension>
      <dimension field="Width" unit="cm">27</dimension>
   </part>
   <part name="(mount)">
      <dimension field="Height" unit="cm">47.5</dimension>
      <dimension field="Width" unit="cm">35</dimension>
   </part>
</dimensions>
```

The SQL create statements would look like this (this, obviously needs to be done prior to the import):

```
CREATE TABLE vanda_dimensions (
  id varchar(50) NOT NULL,
  name varchar(50) DEFAULT NULL,
  width varchar(255) DEFAULT NULL,
```

```
  height varchar(255) DEFAULT NULL
);
```

"Id" refers to the object id, "name" is the type of dimensions (i.e. mount/image/etc.) and "width" and "height" are the actual dimensions. The XQuery script would thus produce an `insert` statement like this:

```
INSERT IGNORE INTO vanda_dimensions (id, name, width, height)
VALUES( 'O10003' ,  '(image only)' ,  '27cm' ,  '40.9cm' );
```

```
INSERT IGNORE INTO vanda_dimensions (id, name, width, height)
VALUES( 'O10003' ,  '(mount)' ,  '35cm' ,  '47.5cm' );
```

NB the brackets around "mount" etc. are removed by the SQL clean up scripts.

Cleaning the data

Cleaning the data is done within SQL once the data has been imported. In some cases it made more sense to do this once all the individual datasets had been transferred into the LIDO schema, as some issues occurred in more than one dataset (such as date formats). The cleaning operations for the individual data sets will be described in the next sections, the one for the general LIDO schema is detailed further below in the section on the LIDO implementation.

Individual Datasets

Birmingham City Library

The data from BCL was received as Excel sheet. Exporting from LibreOffice Calc to CSV and importing it into MySQL was no challenge.

The data needed some clean up (see `clean_birmingham.sql`), chiefly to correct typos, get spelling consistent and to unify date formats.

British Library

The dataset from the BL proved to be the most difficult to import, simply because we received it as MS SQL Server 2000 dump file. There was no MS SQL Server instance running on the computers the team had access to. Since installing it on one of the other Windows machines required Admin rights, AvL used his private laptop for this. The process was very messy and required a lot of trial and error, as the migration wizard would make wrong assumptions and data types etc. had to be adjusted manually before it succeeded. It is highly recommended to ask the BL for a different format in the future. As pointed out below, MS SQL Server is used by the BL as XML data store, so exporting the data as XML file (as done by AvL, see below) should be feasible.

The process ran as follows (all on the MS Windows machine):

1. Download and install SQL Server Express 2008 (from `http://msdn.microsoft.com`). One needs to search a bit for this version, as the latest version (SQL Server Express 2012) is not downward compatible enough for a SQL Server 2000 dump file.
2. Download and install the SQL Server 2008 Client library (for ODBC).
3. Install MySQL Workbench v5.2.4+ (for the migration wizard).
4. Configure MySQL server on the Linux box which holds the FuzzyPhoto db to allow connections on port 3306:
    - `iptables -A INPUT -i eth0 -p tcp --destination-port 3306 -j ACCEPT`
    - in `/etc/mysql/my.cnf` change/add this line: `bind=0.0.0.0 !`
    - `reboot`
5. Restore the dump file received from the BL (in this case it was named `JerwoodDiscovery.bak`) into SQL Server Express 2008 (can be done with the GUI tools that come with the installation package).
6. Configure an ODBC DSN in your system settings (here it was given the name “mssql”). Important: Use path for server, i.e. `AVL-PC\MSSMLBIZ`
7. Use the data migration wizard in the MySQL workbench to transfer the data from the SQL Server 2008 Express database (via ODBC) to the MySQL database running on the Linux box.
    - There was an error during this process coming from the schema DDL when choosing automatic migration: “Wrong syntax for column timestamp in table TcRecords”. The reason being that the SQL Server 2000 data type timestamp is different from the SQL-92 standard timestamp data type.

- To fix this, do the following:
    a) click “Back” twice from this point to get to “Manual Editing”
    b) choose “All Objects”
    c) pick table *TcRecords*, go to columns
    d) change the DDL from `timestamp TIMESTAMP(0) NULL` to `timestamp BIGINT DEFAULT 0`
    e) in “Manual Editing” delete the index `UNIQUE INDEX IX_Key …` from the DDL; this may raise an error, which you can ignore, just re-create the index in the MySQL database after migration manually

The data should now be in the MySQL database on the Linux box. There are quite a number of tables, most of which seem to be redundant or otherwise unnecessary. It turned out that some of the tables held straight tabular data, while others were used to hold much of the same data in XML format (i.e. SQL Server 2000 is used as XML store by the BL). After some analysis and a query to the BL, it was assessed that most of the XML data was the more recent and complete version. Thus, it was decided that it is easier to dump the XML data as a text file from the db and then re-import it as SQL data in the way described above (“Importing XML data”), rather than trying to do this within MySQL. Out of the 28 tables that were imported from SQL Server dump file, only one table (*TcRecords*) had all the relevant XML data in it. The column in that table was simply dumped into a text/XML file through the command line tool:

```
mysql -u avl -p -v JerwoodDiscovery < bl_xml.sql > bl.xml
```

“JerwoodDiscovery” is the database name the BL dump file was migrated into with the MySQL Workbench data migration wizard (see above), “bl_xml.sql” is the name of the file holding the SQL statement selecting the correct column in *TcRecords*: `select ItemXob from TcRecords;` and “bl.xml” is the name of the file that column data is redirected into. After issuing that command, you should have a file bl.xml which holds the relevant XML data from the table *TcRecords* from the BL SQL Server dump file. This XML needed some cleaning before being imported into BaseX, just to make the XQuery queries run smoother (and to make it easier to re-cycle the scripts created previously for other data sets):

1. Delete the first line (these are the column names)
2. Delete all `<?xml ...>` tags from every line but the first (use an editor with a global replace function); every row in *TcRecords* represents an individual XML file, hence the XML declaration in every row
3. Add “`<bl>`” as second line and “`</bl>`” as last line of the file, so that all rows are aggregated into one global entity

For this XML file a number of XQuery scripts were created to create a series of tables to translate the XML schema into a relational data model. The XQuery scripts export these into CSV files rather than SQL `insert` statements, simply because there were a lot of columns/tables that would have needed to be created manually and it was quicker to let the CSV import assign names and data types (and then clean up those if necessary).

It was noted that there were some duplicate lines in the data, but these could be easily identified within LibreOffice Calc (the equivalent to Microsoft Excel) and deleted. The CSV file was then imported into MySQL as described above.

While *TcRecords* held the relevant data in terms of catalogue records, some of the tables in the SQL

Server dump file had some necessary “auxiliary” data. The relevant tables were thus copied into the temporary schema within MySQL:

```
create table fuzzyphoto.bl_textbooks as select * from
JerwoodDiscovery.JerwoodTextbooks;

create table fuzzyphoto.bl_publishers as select * from
JerwoodDiscovery.JerwoodPublishers;

create table fuzzyphoto.bl_pibooks as select * from
JerwoodDiscovery.JerwoodPIBooks;

create table fuzzyphoto.bl_itemtypes as select * from
JerwoodDiscovery.TcItemTypes;

create table fuzzyphoto.bl_events as select * from
JerwoodDiscovery.JerwoodCVEvents;

create table fuzzyphoto.bl_departments as select * from
JerwoodDiscovery.JerwoodCVDepartments;

create table fuzzyphoto.bl_genres as select * from
JerwoodDiscovery.JerwoodCVGenres;
```

The create statement for the *TcRecords* table (as *bl_records*) is in a separate file `create_bl_records.sql`, as it was decided to slightly alter the field names to make it easier to use in follow up queries.

The cleaning was quite straight-forward, as it basically only required some “trimming” of text columns (i.e. using the `trim()` function in SQL to remove leading and trailing whitespace characters) and updating of id's between tables (in `clean_bl.sql`).

CollectionsOnline

A Java application was created to download and parse the records from the Collections Online (CO) website. However, once the records were put into the database and the NMeM data was received a comparison showed that the two were identical. There are more collections in CO, but they're usually about newer photographic collections that have little relation to the partner's data sets. Therefore, the CO data was removed from the *fuzzyphoto* db.

CultureGrid

A Java application was written to download data from the CultureGrid (CG; `http://www.culturegrid.org.uk/`) website. To query the CG website the photographer names from the ERPS and PEIB databases were used. The CG website provides the data as XML, so the single records were loaded into BaseX and transformed into SQL insert statements with XQuery queries (see above).

The data in the main table then needed some clean up, chiefly on typos etc. in names (see `clean_culturegrid.sql`). The auxiliary tables also needed attention, as most had different data stored in one field when it should have been several fields. For this, the original auxiliary table was copied (with “`_tmp`” at the end) and the separated fields inserted into it. Then the original data was dropped and the “`*_tmp`” table was renamed to the original table's name (i.e. without “`_tmp`”). The scripts to do this all is named `upd_..._tmp.sql`

Library of Congress & Brooklyn Museum

AvL took a copy of David's downloaded data from the Library of Congress and Brooklyn Museum. The data was a bit messy in some places, as David had stored it in a MySQL schema with Latin1 encoding. However, there were names in the data (chiefly Japanese and Arabic photographers/sitters) that don't fit into Latin1, but need UTF8. AvL tried to re-convert it, but the functions within MySQL failed to do that (there seems to be no way to explicitly tell a string function in MySQL's SQL what encoding a specific column is in, other than at `CREATE` time). Thus, a list of of messed up characters was compiled and the conversion was done manually, i.e. calls to the `replace()` SQL function (see `clean_loc.sql`). Other than that, the insertion was rather straight-forward (see `insert_loc.sql`).

Metropolitan Museum NYC

The data from the Met arrived as an Excel sheet. Converting it to CSV and importing it into MySQL turned out to be without any issues. The data was very clean and needed only one update statement to set the "*alpha_sort*" column which holds the photographers names in surname, forename order (whereas the "*attribution*" column holds the name in forename surname order):

```
update fuzzyphoto.met set alpha_sort = 'N/A' where alpha_sort =
'';
```

This helps in querying the data when transferring it into LIDO. The only odd issue with the Met data is that it seems to have used the dates of the prints rather than the date of the shoot, so the event type "Print" should be used for the LIDO event table for these rows.

National Archives

The data from the National Archives was slightly messy. It turned out that the CSV file they gave us uses commas as field separator in most rows, but in some rows it uses the TAB character. Thus, importing it with the MySQL import tool as well as with LibreOffice failed at the first tries. Furthermore, in two rows misplaced TABS were found, i.e. instead of using one TAB there were three (without them being empty columns, but indeed superfluous characters). There were also misplaced 'NULL' words (i.e. the literal word, rather than the column value), which had to be deleted.

Also, extra commas were encountered. These seem to come from "empty" rows and are most likely a result of "gaps" in the National Archive's database, as much as the other issues described are a problem on their side (maybe the export function of their database client doesn't work properly).

Generally, most rows used quotation marks to indicate string values (as opposed to numerical values), but for some rows this rule was violated, i.e. there were quotation marks missing (usually only the starting quotation mark was missing). The only fix for this was to search for character strings (in a **good** text editor, in which one can search via regular expressions) that don't have a comma and a quotation mark as first characters. NB that's why the "NULL"s from above should be deleted first, otherwise they trigger that regular expression. The "uncouth" strings encountered were (just the first word of the string given here, to make it clearer; the "Original" value in the following table should be the term being searched for in your editor, the "Convert into" the replace term in the editor):

Original	Convert into
,Photograph …	,”Photograph ...
“”	[null]
,Imperial …	,”Imperial ...
,Cabinet …	,”Cabinet …
,A bust …	,”A bust …
,(1) ...	,”(1) …
,Instantaneous …	,”Instantaneous …
,CDV …	,”CDV …
,Cart …	,”Cart …
, (1) …	,”(1) …
, 'Photograph …	,”'Photograph ...

The following two occurred at the end of strings:

[blank],19	“,19
[blank],18	“,18

There were some rows that were essentially empty rows. “Essentially” because although they had some value set, this was one of the following: “Item number not used”, “Number not used”, “... not registered”, “... not used” or “... Not Used”; i.e. it is not possible to link them into the National Archives collections. These rows should be (and have been) deleted.

Unfortunately, since both MySQL and LibreOffice Base/Calc won't load the data in its original state, one has to do the cleaning “manually”, i.e. in an editor. As the errors in the original CSV might be random (i.e. there might be different issues with a new export), there is little in terms of a strategy that could be offered at this point. One should try to import the data and check the messages and log files. Sometimes the MySQL importer gives only summary statements, such as “x many rows in data, y many rows imported, z many warnings”. One then needs to check whether the number of rows in the MySQL table is correct, and check the log.

The import command used was:

```
mysqlimport --fields-terminated-by=',' --lines-terminated-by='\n'
–fields-optionally-enclosed-by='”' -p -u <username> -v -L
fuzzyphoto national_archives.csv
```

Once the data was in there, the data field (there are just two usable columns, the id – which is split across two columns, see below – and the description field, which holds all information), needs to be parsed for the authors. The description field more or less only holds the photographer's name and a description of the image (sometimes the description is also the title, but seems to be rather random). Therefore, I created an additional column for the photographer and extracted that name from the description column. Luckily, other than the NMS data, the National Archives used uniform structures and tokens in their description field, so the extraction was not overtly complicated (all in `clean_narch.sql`).

National Museums Scotland

The data of the NMS arrived in a relatively unstructured state. The Excel sheet consisted of only two columns: an id and a description field where all meta-data such as creator, date of creation, title, etc. were listed together. Moreover, the structure of this description field was not consistent. The data had been entered by volunteers, and every data entry volunteer apparently used a different structure. This made organising the data rather hard. Some data has been extracted when keywords such as “created in” were provided next to the dates and so on, but a good deal remains unstructured as part of the `lido_photography.object_description` field.

National Media Museum

We received the data from the NMeM as an Excel sheet. As a first try, it was exported into a CSV file, but the import into MySQL (in the way described above) failed. Apparently, there were newline characters in the description column that confused the importer. It was consequently decided to try it with LibreOffice's Base program, as this is usually more tolerant towards irregular characters (and can be configured to handle special characters).

To do this, do the following:

1. Create a new db in Base, use “Use existing connection” and “Spreadsheet” as type
2. Import the NMeM spreadsheet (i.e. the Excel file); NB this will only provide an editing interface to the file, and not create an actual copy of it within Base
3. Create another db in Base, use “Use existing connection” and “MySQL” as type and connect to the db “fuzzyphoto”
4. In the first db (the NMeM data) in the list of tables, do a right-click and “Copy” on the table; in the second db (the MySQL connection) go to the table list and do a right-click and “Paste Special”, pick “Data source table” as source, go through the wizard and pick the correct column data types. If you would just do the normal “Paste”, data types would be guessed (and wrongly in most cases) by the wizard and cause another error

The cleaning up process was more or less solely about the names of the creators. In the beginning there was only one column for creators and multiple creators were concatenated into that one column. A quick check revealed that there were some entries with more than four names in that one column, so three more columns for extra names were added to the table. Then, a number of update statements were run to split the one original creator name column into these added columns. Furthermore, some updates were run to get some consistency into the spelling of the names (see `update_maker.sql`).

Victoria & Albert Museum

The data from the V&A was received as XML data and was well structured. The format was discussed with the V&A prior to the delivery, upon a sample set sent earlier. There were five objects in the XML schema that needed to be separated out into individual tables in the MySQL db (see the `create_*.sql` files). There were some empty nodes in the XML file, but they were quickly spotted and deleted in XQuery, so that they posed no problem when creating the SQL insert statements from within BaseX.

Cleaning up the data involved straightening out some inconsistencies in the creator names, such as

replacing HTML entities (e.g. "`&`") or checking for multiple creators in one data field. Furthermore, fields were set to null when they held a string of length zero in various tables.

Sorting out multiple titles in the one title field was somewhat more elaborate. Some items had multiple titles, often the photo and the album title. In these cases it made more sense to use the brief description field as title. To achieve this all titles from the *vanda_titles* table that are multiple titles to one id were dropped, then the unique titles were inserted into the main table, and then the main table was updated with the brief descriptions for those rows that had been dropped (i.e. the ones with multiple titles).

There are multiple dimensions for the image and the mounted image. Under the LIDO schema, this would result in different events, i.e. several rows. However, there were no dates etc. given for the mount, so insufficient data was available to create an extra event. (Although events in LIDO do not need to have a date, the fuzzy logic algorithms would need more than just different dimensions). The mount dimensions were therefore dropped from the data set.

St Andrews University Library

The data from St Andrews came as a bunch of CSV files (13 CSV files, and one `ini` file with meta-data on the CSV file, and an Excel sheet with a diagram showing the relations between the different entities). The tables were created with the help of the `ini` file, i.e. the SQL `create` statements were informed by the `ini` file (see `create_standrews_tables.sql`). The CSV files were then imported by the method described in the General Issues sections.

The data was pretty clean (St Andrews spent quite some time to clean it up themselves), so it was essentially only a couple of typos that needed corrections.

Musee d'Orsay

The MO sample data was delivered as an XML file. A number of XQuery queries turned this into SQL insert statements, with the table create statements being authored manually (see `create_*.sql` files). The only peculiarity with the data was that there were several entities with "*historique*" in their names. The difference in name was the only variation, the structure being the same across all these entities. Consequently, only one table in SQL was created as *orsay_historique* and the different entities were imported into this one table, with a column "*hist_type*" that would specify from which of the different entities the rows came. Similarly, there were two entities with "*actuelle*" in their names, for which one table *orsay_situation_actuelle* was created in MySQL.

The data as such was quite clean. The only update necessary was deleting "empty" rows, i.e. rows which for some reasons had no actual data in it (NB this became quite an issue with some of the tables in the final dataset we got from them, making some of the SQL files quite large (~60MB) with a lot of empty rows that get imported, just to be dropped afterwards, wasting time; as an improvement, the XQuery script should filter out those rows immediately before the SQL export; on the other hand, not all of the tables/files were actually used for the LIDO schema – see below – so some of the SQL files can be ignored, thus saving time).

Transferring the partners' data into the LIDO schema

Entities and relationships

LIDO is defined as an XML Schema Definition (XSD; see `http://www.lido-schema.org/schema/v1.0/lido-v1.0.xsd`) and is thus not immediately usable with MySQL (unless one wants to use the XML features of MySQL). The XML schema was therefore

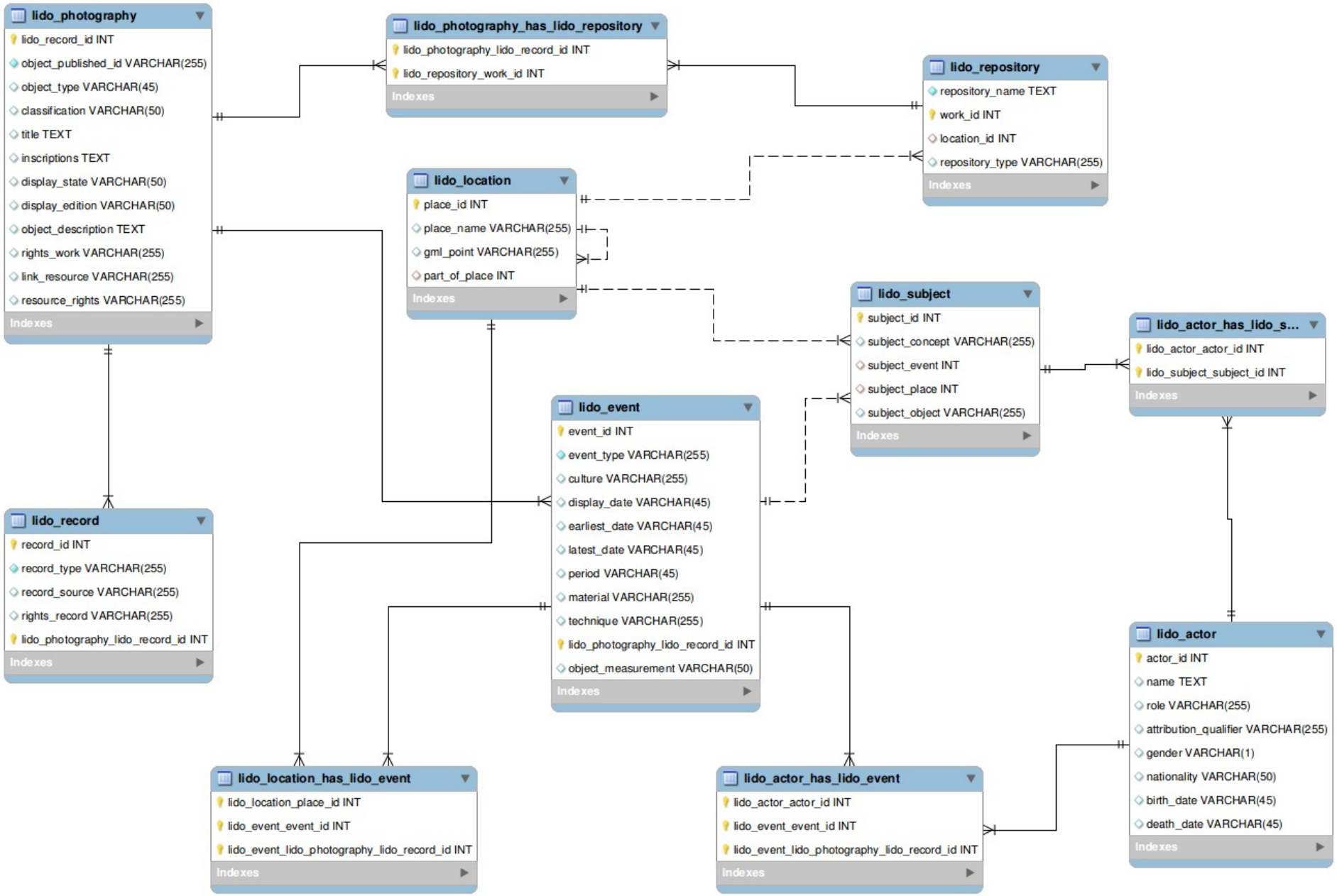
manually translated into an Entity-Relationship-Model (ERM), based on the schema visualizations on the LIDO webpage. The only deviation from the XSD is that the field "*Object Measurement*" was put in the event entity rather than the main record entity. This is because of different sizes of photographic prints made from the same photograph, with prints being regarded as events to one original photo and not new objects (as a LIDO introductory paper itself suggests). Consequently, measurements change with every event, a.k.a. print, and the events table needs a column for that. Furthermore, a column named "*internal_remark*" was added to the main table (*lido_photography*) to mark the origin of the data (i.e. strings like "nmem" etc. were used to differentiate between the sources of data). While the different partners could be distinguished by using the *lido_repository* table, this would involve several Inner Joins. For the cleaning operations, using *internal_remark* was much quicker.

The SQL scripts to create and clean the LIDO schema in MySQL are all in the lido/ subdirectory (see below for a ERM diagram of the LIDO schema implementation). The SQL scripts to migrate the individual data sets from the partners reside in the respective partner's subdirectory (as `insert_lido_*.sql`). The analysis of what columns to transfer across between an individual data set was entirely manual, i.e. the respective data was visually inspected and upon that a decision was made about the corresponding columns in the partner's data and LIDO. When transferring creator names into the table/field *lido_actor.name*, the field *lido_actor.role* should usually be "Photographer", unless the data clearly states/shows that it is a different role (such as "Printer"). The nomenclature for the names should be *<surname>, <first name> <middle name>*.[2] Some LIDO data has to be "made up" as it is not in the data from the partners as such, e.g. the data for the *lido_repository* table, which is data about the respective partner institution etc (see the SQL scripts).

To implement the LIDO schema, the XSD diagrams were used as a blue-print and the ERM was modelled in the modeller in MySQL Workbench. This made it possible to draw an ERM graphically, define data types and relationships, and then have the software generate the database in SQL. The default name for that database assigned by the modeller is "*mydb*" and it was hard to figure out how to change that, so it was left at that.

The ERM diagram for the LIDO schema used for the FuzzyPhoto project is as follows:

2 It should go without saying that the transfer into the LIDO schema for the names should be done with `INSERT IGNORE INTO … SELECT DISTINCT …` to avoid duplicates.


lido_photography
lido_record_id INT
object_published_id VARCHAR(255)
object_type VARCHAR(45)
classification VARCHAR(50)
title TEXT
inscriptions TEXT
display_state VARCHAR(50)
display_edition VARCHAR(50)
object_description TEXT
rights_work VARCHAR(255)
link_resource VARCHAR(255)
resource_rights VARCHAR(255)
Indexes
lido_photography_has_lido_repository
lido_photography_lido_record_id INT
lido_repository_work_id INT
Indexes
lido_repository
repository_name TEXT
work_id INT
location_id INT
repository_type VARCHAR(255)
Indexes
lido_location
place_id INT
place_name VARCHAR(255)
gml_point VARCHAR(255)
part_of_place INT
Indexes
lido_subject
subject_id INT
subject_concept VARCHAR(255)
subject_event INT
subject_place INT
subject_object VARCHAR(255)
Indexes
lido_actor_has_lido_s...
lido_actor_actor_id INT
lido_subject_subject_id INT
Indexes
lido_event
event_id INT
event_type VARCHAR(255)
culture VARCHAR(255)
display_date VARCHAR(45)
earliest_date VARCHAR(45)
latest_date VARCHAR(45)
period VARCHAR(45)
material VARCHAR(255)
technique VARCHAR(255)
lido_photography_lido_record_id INT
object_measurement VARCHAR(50)
Indexes
lido_record
record_id INT
record_type VARCHAR(255)
record_source VARCHAR(255)
rights_record VARCHAR(255)
lido_photography_lido_record_id INT
Indexes
lido_actor
actor_id INT
name TEXT
role VARCHAR(255)
attribution_qualifier VARCHAR(255)
gender VARCHAR(1)
nationality VARCHAR(50)
birth_date VARCHAR(45)
death_date VARCHAR(45)
Indexes
lido_location_has_lido_event
lido_location_place_id INT
lido_event_event_id INT
lido_event_lido_photography_lido_record_id INT
Indexes
lido_actor_has_lido_event
lido_actor_actor_id INT
lido_event_event_id INT
lido_event_lido_photography_lido_record_id INT
Indexes

The transfer schemas were as follows:

Birmingham City Library

Only one temporary table for BCL, which needs to be split up to fill various LIDO tables.

Temporary Table	LIDO
date_range	lido_event.earliest_date/latest_date (split)
Format	lido_event.material
Dimensions	lido_event.object_measurement
Title	lido_photography.title
Description	lido_photography.object_description
Reference	lido_photography.object_published_id
Creator	lido_actor.name
	lido_event.event_type = 'Shoot'

British Library

The data from the BL is spread across 15 tables, but a lot of this isn't used (on average its one column per table). All tables are linked through the “id” column.

Temporary Tables	LIDO
bl_records.RecordId	lido_photography.object_published_id
bl_records.Caption	lido_photography.title
bl_records.StartDateRange	lido_event.earliest_date
bl_records.EndDateRange	lido_event.latest_date
bl_records.Height/Width	lido_event.object_measurement
bl_records.Process	lido_event.technique
bl_records.DescriptiveNotes	lido_photography.object_description
bl_records.CopyrightStatus	lido_photography.resource_rights
bl_records.Subject	lido_subject.subject_object (not used, since only one other data sets has this information)
bl_records.Location	lido_location.place_name (not transferred yet, as it requires more work)
bl_records.Inscription	lido_photography.inscription
bl_secondarysupport.support	lido_event.material
bl_events.Event	lido_subject.subject_object (see above)
bl_genres.Genrer	lido_subject.subject_object (see above)
bl_photographers.name	lido_actor.name (role = 'Photographer')
bl_portraitsubject.name	lido_actor.name (role = 'Portrait Subject')
	lido_event.event_type = 'Shoot'

CultureGrid

The data downloaded from the CultureGrid web service is spread across eight tables, although little data from each table is actually used (all tables are linked through the “aggregator_internal_id” column).

Temporary Tables	LIDO
culturegrid.authority_name	lido_photography.resource_rights
culturegrid.dc_description	lido_photography.object_description
culturegrid.dc_identifier	lido_photography.link_resource
culturegrid.dc_rights	lido_photography.rights_work
culturegrid.dc_title	lido_photography.title
culturegrid.temporal_from	lido_event.earliest_date
culturegrid.temporal_to	lido_event.latest_date
dc_creator.dc_creator	lido_actor.name
dc_subject.dc_subject	lido_subject_object (not transferred, since no other data set has this kind of information)
dcterms_spatial.dcterms_spatial	lido_location.place_name (not transferred yet, as it requires some extra work to get it in the right format)

ERPS & PEIB

The Photographic Exhibition In Britain (PEIB) and the Exhibitions of the Royal Photographic Society (ERPS) databases, created at DMU, were transferred into the LIDO schema. The transfer was quite straight-forward; however, given that these databases have digitized catalogues from exhibitions in them, rather than actual records of actual collections of historic photos, the structure is slightly different. For the LIDO schema this simply means a different event type and less data (as there is no repository for these datasets, for example).

The only real issue were the exhibitor names; names were given in full with life dates in brackets as part of the name field. The dates had to be stripped to make it LIDO compliant.

PEIB	LIDO
exhibits.exhibid	lido_photography.object_published_id
exhibits.exhibtitlenorm	lido_photography.title
concat(exhbndetail.opened, ' ', exhbndetail.year)	lido_event.earliest_date
concat(exhbndetail.closed, ' ', exhbndetail.year)	lido_event.latest_date
exhibits.processnorm	lido_event.technique
exhibits.photographernorm	lido_actor.name
	lido_photography.event_type = 'exhibition'

ERPS	LIDO
exhibitors.name	lido_actor.name

exhibits.etid	lido_photography.object_published_id
exhibits.title	lido_photography.title
exhibitions.datestart	lido_event.earliest_date
exhibitions.dateend	lido_event.latest_date
exhibits.process	lido_event.technique
	lido_event.event_type = 'exhibition'

Library of Congress and Brooklyn Museum

Data was across tables, but David created a view to unify them, which is what was copied across.

Temporary Table	LIDO
uid	lido_photography.object_published_id
title	lido_photography.title
description	lido_photography.object_description
person	lido_actor.name
process	lido_event.technique
date	lido_event.earliest_date/latest_date
source	lido_photography.internal_remark
image	lido_photography.resource
	lido_event.event_type = 'Shoot'

National Archives

The data from the National Archives UK was in one table. After the CSV file was inserted one column name had to be altered, because it had a slash character in it, which might have proven troublesome for queries.

Temporary Table	LIDO
series + '_' + piece_ref + '_' + item_ref	lido_photography.object_published_id
Content [renamed from scope/content]	lido_photography.object_description
date_text	lido_event.display_date
	lido_event.event_type = 'Shoot'

National Media Museum

Only one temporary table for NMeM.

Temporary Table	LIDO
id_number	lido_photography.object_published_id
item_name	lido_event.technique
title	lido_photography.title
maker	lido_actor.name

date_made	lido_event.earliest_date (only one date)
measurements	lido_event.object_measurement
description	lido_photography.object_description
	lido_event.event_type = 'Shoot'

National Museum of Scotland

As mentioned, the NMS data is relatively unstructured, but some data could be extracted. The fields are as follows:

Temporary Table	LIDO
id	lido_photography.object_published_id
description	lido_photography.object_description
type	lido_event.technique
maker	lido_actor.name (role = 'Photographer)
title	lido_photography.title
dimensions	lido_event.object_measurement
	lido_event.event_type = 'Shoot'

Metropolitan Museum

The MET data is one temporary table only. The field "*alpha_sort*" is used for the creators' name rather than the field "*attribution*", as the former is in the right format (i.e. surname first) and identical to the latter otherwise.

Temporary Table	LIDO
object_id	lido_photography.object_published_id
alpha_sort	lido_actor.name
object_name	lido_photography.object_type
Title	lido_photography.title
begin_date	lido_event.earliest_date
Medium	lido_event.technique
Description	lido_photography.object_description
	lido_event.event_type = 'Print'
	lido_photography.link_resource = 'http://www.metmuseum.org/Collections/search-the-collections/1900' + object_id

Museé d'Orsay

The data from the Musee d'Orsay is held in 14 tables, although it turned out that at least three of them can be ignored, as the data in them is not usable for the project (or no other partner has delivered this kind of data, so it is slightly pointless to include it). Since loading of the SQL files can be quite time-consuming, one should probably not even import the SQL files, speeding up the process. The tables not being used were *orsay_iconographies* (subject terms), *orsay_historique*

(items' histories, e.g. date of acquisition etc.) and *orsay_fonds* (sponsors). The SQL scripts all start with `insert_*<table name>.sql`.

The Musee d'Orsay data was very clean and their data model quite sophisticated. There are three areas were care must be taken: 1) The data has often more than one title (up to three) per item, usually a short and a long version, but also sometimes the French and the English title. All of the title variants might be relevant, yet LIDO only allows one title (or at least in our SQL implementation of it). As a solution, and considering that this is the only dataset having this issue, the different titles were concatenated into one line, with the different titles separated by a vertical bar ("|") if applicable. 2) A similar issue arose with the inventory number of the item, where there are up to four in the Orsay data. Maybe this indicates different prints of the same image, which in LIDO should be separated out in four different events (of type 'Print'), but there is no other data to go along with it, such as the date of the individual print. So, again, the different inventory numbers were concatenated into one string, and the different numbers separated by an ampersand ("&"). (NB the opposite, unfortunately, is also true for the data: there are records that have several internal id numbers for the same inventory number, and the only obvious difference are different inscriptions, i.e. title and dimensions are the same. During a team meeting it was decided to delete these duplicates, and only enter one record with one inventory number). 3) For the dates, the Musee d'Orsay introduced different qualifiers to denote whether it is a certified date, a date interval or an open interval (i.e. it is given a start or end date to say whether a photo is believed to have been taken before/after a certain date). To hold this in LIDO, the following conventions were used: a) if both dates (*earliest_date/latest_date* in *lido_event*) are given, these are exact intervals, i.e. the shooting was between these dates (only years are given, so there are a good number of rows with the same start and end date/year, specifying that the shooting was in that year). b) if either the start or end date is NULL, it is an open interval.

It should also be noted, that there are no image descriptions in the Orsay data, but a column with inscriptions that were made on the photo. In other datasets inscriptions are often recorded with the image description. On the other hand, there is a column for inscriptions in LIDO, as well as for descriptions. For the Orsay data, the insciption column is used, while no other dataset has anything in there.

Furthermore, while there are dimensions in the *orsay* table, it is in a different format, so they had to be converted (see `clean_orsay.sql`). As another issue, the Musee d'Orsay seems to have photos in their store that they don't own, i.e. they are conserving/safekeeping photos from other museums. Consequently, for every item in *lido_photography* from the Musee d'Orsay data there are two repositories for it: the 'Current' one (i.e. where the photo is held at the moment) and the 'Owner' (i.e. the legal owner of the photo, regardless where the photo is stored). This may be relevant for the widget, as interested persons may need to contact both repositories to get access to a certain image.

The tables are linked via the "*id*" column, and "*num_fiche*" in the main table (*orsay*), respectively.

Temporary Tables	LIDO
orsay.copyright_id	lido_photography.resource_right
orsay.dimensions	lido_event.dimensions
orsay.designation_redigee	lido_event.technique
orsay_titres.titre (see above for comment)	lido_photography.title
orsay_situation_actuelle.mtext	lido_repository.repository_name
orsay_inventaires.inventair (see above for comment)	lido_photography.object_published_id
orsay_inscriptions.inscription	lido_photography.inscriptions

orsay_date_modele.date_debut	lido_event.earliest_date
orsay_date_modele.date_fin	lido_event.latest_date
orsay_auteurs_tmp.artiste	lido_actor.name

St Andrews University Library

The data from St Andrews is spread across 14 tables, of which only few rows were actually transferred. The tables are all linked by the "*ecatalogue_key*" column.

Temporary Tables	LIDO
st_andrews_col_artis.NamCitedName	lido_actor.name (role = 'Photographer')
st_andrews_col_artis.BioBirthDate	lido_actor.birth_date
st_andrews_col_artis.BioDeathDate	lido_actor.death_date
st_andrews_ecatalog.ColObjectNumber	lido_photography.object_published_id
st_andrews_ecatalog.PhoRecordLevel	lido_photography.object_type
st_andrews_ecatalog.PhoOriginalDateEarliest	lido_event_earliest_date
st_andrews_ecatalog.PhoOriginalDateLatest	lido_event.latest_date
st_andrews_ecatalog.ColMainTitle	lido_photography.title
st_andrews_ecatalog.ImaDescription	lido_photography.object_description
st_andrews_pho_funct.PhoFunctionType	lido_subject.subject_object (not transferred, as only one other data set has this data)
st_andrews_pho_media.PhoMedia	lido_event.technique
st_andrews_sub_subject.SubSubjects	lido_subject.subject_object (see above)
st_andrews_sit_site_r.SummaryDate	lido_location.place_name
st_andrews_pho_sitte.NamCitedName	lido_actor.name (role = 'Portrait Subject')
st_andrews_pho_sitte.BioBirthDate	lido_actor.birth_date
st_andrews_pho_sitte.BioDeathDate	lido_actor.death_date
	lido_event.event_type = 'Shoot'

Victoria and Albert Museum

The data from the V&A is distributed across five tables (all linked through the "*id*" column in each).

Temporary Tables	LIDO
vanda.museum_no	lido_photography.object_published_id
vanda.spec_phys_desc (if empty use spec_brief_desc)	lido_photography.object_description
vanda.mus_mat_note	lido_event.material
vanda.url	lido_photography.link_resource
vanda.spec_obj_prod_date_start	lido_event.earliest_date
vanda.spec_obj_prod_date_end	lido_event.latest_date
vanda_dimensions.name/width/height	lido_event.object_measurement (concatenate

	width and height, use only name = “images”)
vanda_obj_names.title	lido_event.technique
vanda_titles.title	lido_photography.title

Cleaning the data in LIDO

There are a number of SQL and Python scripts to clean/unify the meta-data once all the partners' data has been migrated into the LIDO schema. There are also some checking queries and some general queries in the lido/ subdirectory, which should be self-explanatory.

Duplicates in the actor table were detected and cleaned up in SQL. However, this was only done for straight matches (obvious typos were cleaned before), i.e. spelling variants were ignored. This was chiefly done because it would have involved quite some extra work at this point, and the fuzzy logic algorithms should be able to pick this up at a later stage anyway. Furthermore, the actor table has a column for gender in it. Given that sometimes the only information about the creator of a photo in the partners' meta-data was a society or company, the following codes were used: *M* for male, *F* for female, *C* for company (or society), and *U* for unknown. Sometimes only a surname was present in the meta-data, so that no gender could be determined, hence the code *U*. When several names were used in the creator field in the partners' data (e.g. “Smith & Wesson”) this was regarded as code *C*, although judicially it may not have been a company but just two photographers cooperating. Since most creator's in the partner's meta-data didn't have any information on their gender, the i-gender Web API (http://www.i-gender.com/main/index.html) was used to update the actor table in LIDO (with `upd_gender.py`). Some had to be manually checked (such as the unknown ones or the companies). Sometimes the i-gender engine would fail and produce nonsense (i.e. identifying a name as female when it is clearly male, and vice-versa), these had to be manually altered. i.e. proof-reading the result of that Python script is inevitable.

A big issue were different formats for recording dates and duration. In total, twenty different date/duration formats have been encountered across the partner's data. Care has been taken to convert them into a unified date format (ISO8601) using regular expressions, but some formats escaped such conversion. These were chiefly 'fuzzy' dates, such as “early 19th century”. To handle this in an efficient manner, the following strategy was chosen: if there were proper dates, the columns *lido_event.earliest_date* and *lido_event.latest_date* were used (sometimes only one of the two could be set, as there was only present in the source data). If there was a 'fuzzy' date, this values was copied into the column *lido_event.display_date*. This might not be in the LIDO spirit (i.e. this is not what that column was intended to hold), but the only alternative would have been to add a column to hold 'fuzzy' dates, which would stray from the LIDO standard even more.

The following issues were detected after all the partners' data was transferred into the LIDO schema:

1. 10,000 records in *lido_photography* had no corresponding entry in *lido_event*, which should never happen (there always needs to be at least one event such as “Shoot” or “Print”, even when dates etc are not given). Analysis: All these records were from the St Andrews data set; the problem is that the query to transfer the St Andrews data to LIDO inner-joins data from two temporary tables and the number of rows in both tables are not identical. Solution: use Left Join when doing the query. Now fixed.
2. 1,187 records in *lido_photography* have a duplicate “*object_published_id*”. Analysis 1: 308 rows are in the CultureGrid data that are also in the NMeM data; Solution: delete the

CultureGrid ones. Analysis 2: the other rows seem to be multiple photos in an album all being given the same object id (from St Andrews and V&A data); Solution: this is ok, ignore.

3. 58 rows with `object_published_id = ''` (i.e. not such id). Analysis: they are all from the BCL data set, checked the temporary table and original data file: most of them have no description and most not even a title. Solution: delete.
4. 2,989 rows had neither title nor description and are thus useless. Solution: delete.
5. While care has been taken not to duplicate photographers, it is inevitable that some are duplicated simply through spelling variants, i.e. typos in the partners' data. A lot could be sorted out with SQL and Python scripts, but there are certainly a good number of them present. The fuzzy logic algorithms should be able to cope with those, i.e. to identify them as matches regardless.

General Issues

Data transfer

While trying to copy the LIDO database from AvL's local machine (MySQL 5.5) to KMD2 (MySQL 5.0.26) several issues arose:

1. The dump file from MySQL 5.5, although plain SQL statements, turned out to be incompatible with MySQL 5.0; there is a switch for `mysqldump` to force downward compatibility.
2. The dump file needs to be transferred to KMD2 and imported locally, as AvL's machine and KMD2 are on different subnets and DMU's routers seem to block port 3306 (which MySQL runs on).
3. There is a bug in MySQL before version 5.0.48 that crashes the database (not only when importing data), to do with a flaw in index truncation causing an assertion in MySQL's code to fail (see `http://bugs.mysql.com/bug.php?id=28125`). Ideally, the MySQL version on KMD2/3 should be updated, but the SLES10SP2 version of Linux running on these machines don't seem to have a newer package available in their repository (most likely because SuSE has discontinued support for this version of their Linux distribution). Packages might be available on OpenSuSE or older mirrors, such as `http://ftp5.gwdg.de/pub/linux/suse/opensuse/discontinued/distribution/10.2/repo/`. However, given time predicaments etc., AvL decided not to update the MySQL on KMD2.

The commands to successfully migrate the LIDO database from MySQL 5.5 to MySQL 5.0.26 on KMD2 were thus ('mydb' being the schema the LIDO database is held in):

1. First, the index on the column *lido_photography.title* was dropped entirely, as it is likely to be the trigger for above mentioned bug. It can be re-created after the import, or it can be ignored. Its main use was when the data was cleaned up, to help with queries.
2. On the machine with MySQL 5.5: `mysqldump -u <db_user_name> -p --add-drop-database --add-drop-table --compatible=mysql40 --complete-insert --set-charset --disable-keys mydb >`

`dump.sql`

3. (S)FTP the file `dump.sql` to KMD2 (or whatever machine you are using as database server).
4. On the designated server, import the file `dump.sql` like this: `mysql -u <db_user_name> -p < dump.sql`

NB you may have to edit the dump file and add the following lines to the top of the file:

```
drop database if exists mydb;
create database mydb character set 'utf8';
use mydb;
```

This should have actually been done by `mysqldump`, but maybe the downward compatibility switch suppressed this.

Migrating Temporary Tables to the LIDO schema

Given that some data sets arrived at about the same time, one could contemplate to copy data from the respective temporary tables into the LIDO schema in parallel. Unfortunately, MySQL capability to run queries in parallel is somewhat limited. Even for SELECT statements MySQL locks the whole table for write-access. Since LIDO uses a number of n:m relationships data is pulled in from other tables from the LIDO schema when populating those relationship tables, meaning that inserting into any of the affected tables can't have new data inserted into them while there is a SELECT query running against them (see `http://thushw.blogspot.co.uk/2010/11/mysql-deadlocks-with-concurrent-inserts.html` for an introduction into the issue). As some of those queries to build the n:m relationships can be quite slow, building the LIDO schema can take several days. This should be considered for future builds. NB make sure there are proper indexes in place in both the LIDO tables and the temporary tables to speed up the migration.

Transferring data to the cluster server

One issue arose when trying to copy the LIDO data (applies to the temporary tables as well) to the cluster server.[3] The default database engine type for MySQL databases is “InnoDB”, whereas for clustered servers, this has to be “NDB” (check the MySQL documentation to learn about the differences). However, the NDB database type has a row size limit of 14,000 bytes per column. The only column large enough to trigger this is the *description* field (in *lido_photography.object_description*, or in the respective temporary tables). Since we’re using UTF-8 encoding, which uses 3 bytes per character (i.e. there’s a 4,666 character limit, c. an A4 page), some of the description fields would go beyond this limit. In the present version there 12 rows in the CultureGrid data that would violate the 14,000 byte restriction. Upon inspection of the rows in question were simply chopped down to comply with this limit (the descriptions in those rows looked like full-blown articles, with “Conclusions” and “References” sections in them):

```
Update fuzzyphoto.culturegrid set dc_description =
substr(dc_description, 1, 13999) where length(dc_description) >
14000;
```

3 While the issue arose through the CultureGrid data, future data sets might trigger it as well, so it is discussed here, rather than in the CultureGrid section.

NB This needs to be done for *lido_photography.object_description* as well, if the data has already gone in. Furthermore, the parameters for uploading the data to the cluster need to be adhusted, but this is detailed in the Cluster work package report.