Databases Project – Spring 2017

Prof. Anastasia Ailamaki

Team No: **14**

Names: Perrotta Lucie, Phan Hoang Kim Lan, Nguyen Tim

# Deliverable 2

Design evolved a bit from milestone 1. Tables we added are has\_featured\_character(story\_id, character\_id) as well as has\_editing(story\_id, artist\_id).

## Assumptions

Parsing leads to lots of assumptions. Here the ones we had to assume :

* Names are full of information between parenthesis or brackets (such as (signed), (translator)…). We delete them to be able to rely them (instead of having twice the same author). Also to make them match even more, we construct a comparative string, which is the name of author without spaces, dot and hyphen. These can permit us to avoid duplicata or obtain full names of artists (our actual version uniquely keep first found entry but should be modify so that we keep the longer and most detailed ones).
* Dates are difficult to retrieve from given csv file, because of its non-uniform format. We assume:
  + “1870’s” become “1870”
  + “July 10 1870” become “1870”
  + “1870-07-10” become “1870”
* Stories with null titles are deleted.

## Parsing

We choose to use a local database, using **wamp server** (windows local server) and **phpmyadmin**. Thus we are parsing the csv file using php.

We used the following php commands to do the parsing. This is the “*functions.php*” file, used in every php parsing files.

We had to parse multiple times the csv’s given to us.

For the beginning, we created scripts which create sql commands from csv data. During the process, we get rid of null values such as “Null”, “none”, “[nn]”, “?”, etc. We also had to change some column types such as:

* isbn , rating, number of issue becoming varchar
* synospsis of story becoming text
* dates becoming integer

[functions.php]

**Website** :

Websites come from publisher, indicia\_publisher and brand\_group csv file. We get the url values, parse null values, and add it in a new website table. We never add twice the same website. Once done, we process publisher, indicia\_publisher and brand\_group csv files and write into a new csv file their values plus their url values changed into website ids (which become a foreign key referencing the website table). These csv files are then process into sql commands to import them into our database.

[website\_tocsv]

[website\_to\_id]

[website.php]

**Artist, Characters, Genre** :

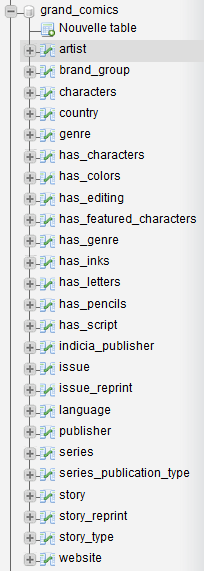
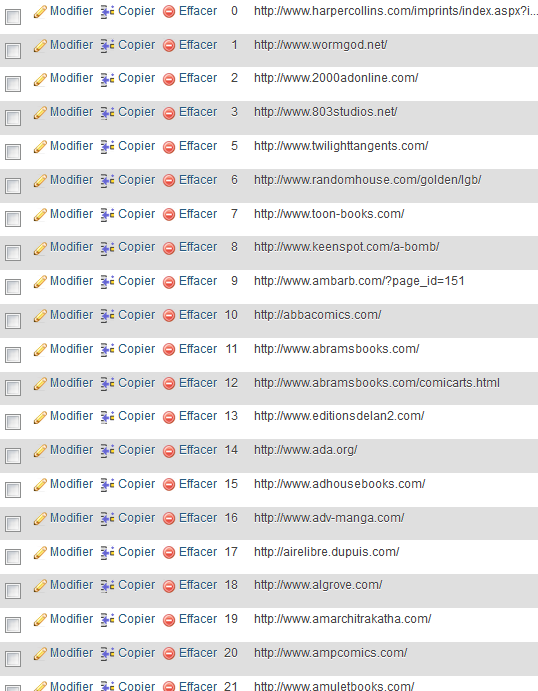
Artists come from story csv file. We get names, parse null values, and add it in a new artist table.

If a story has an artist (for script, letters, inks…) we add in an artist csv file the found artist (id, artist name) and we also add in a has\_ csv file the pairs (story\_id, artist\_id). These csv files are then process into sql commands to import them into our database.

This same idea is also applied to characters (features and characters from story) and genres.

[genre\_tocsv.php]

**Screenshots of the current database**

**** 

All tablesWebsite table

Has\_genre table Language table

## Query Implementation

Here are the 8 queries we were asked to implement in SQL.

For this first query we first select all the brand name possessing at least one indicia publisher from Belgium. We choose to get the name of the brand\_group and the number of Belgian indicia it possesses. Then we simply sort the resulting table by number of indicia and get the name only.

*-- a)*

**SELECT** **T**.**name**

**FROM** (

**SELECT** **B**.**name**,

**COUNT**(**\***) **AS** bid

**FROM** brand\_group B,

indicia\_publisher I,

publisher P,

country C

**WHERE** **C**.**name** **=** 'Belgium' **AND**

**C**.**id** **=** **I**.**country\_id** **AND**

**I**.**publisher\_id** **=** **P**.**id** **AND**

**P**.**id** **=** **B**.**publisher\_id**

**GROUP BY** **B**.**name**

) **AS** T

**ORDER BY** **T**.**bid**

For the query b), we simply use the chain AND rule to get all publishers from Denmark, in a straightforward fashion.

*-- b)*

**SELECT** **P**.**id**, **P**.**name**

**FROM** publisher P,

country C,

series S

**WHERE** **C**.**name** **=** 'Denmark' **AND**

**S**.**country\_id** **=** **C**.**id** **AND**

**S**.**publisher\_id** **=** **P**.**id**

For querry c), the fashion is similar to b), we simply apply the chain rule, to get series from Switzerland and published in a magazine.

*-- c)*

**SELECT** **S**.**name**

**FROM** series S,

country C,

series\_publication\_type T

**WHERE** **T**.**name** **=** 'magazine' **AND**

**T**.**id** **=** **S**.**publication\_type\_id** **AND**

**S**.**country\_id** **=** **C**.**id** **AND**

**C**.**name** **=** "Switzerland"

Here in d) we simply want to get all issues from 1990, sorted by year. Note that for simplicity purposes, all the dates have been converted to years stored as integers, since most of the dates simple consists as a year, often followed by text such as “circa”. It is hence easier to work with int(4) formatted years.

*-- d)*

**SELECT** **COUNT**(**\***)

**FROM** issue I,

**WHERE** **I**.**publication\_date** **>=** 1990

**GROUP BY** **I**.**publication\_date**

We still are not sure about the query e), our result returns the number of series that have been published by some publisher, instead of separating them by indicia\_publisher. The correct query should be quite similar but we had no time yet to figure out what’s wrong.

*-- e) almost OK (now returns all series by publisher, not by indicia)*

**SELECT** **T**.**name**, **COUNT**(**\***)

**FROM** (

**SELECT** **I**.**name**

**FROM** indicia\_publisher I,

publisher P,

series S

**WHERE** **I**.**publisher\_id** **=** **P**.**id** **AND**

**S**.**publisher\_id** **=** **P**.**id** **AND**

**I**.**name** **LIKE** '%DC\_\_omics%'

) **AS** T

**GROUP BY** **T**.**name**

For query f), we first select all stories that have been reprinted at least once, and then regroup them by original story. Finally, we count how many times each original story has been reprinted and sort them according to that. We only print the names of the stories for aesthetical purposes.

*-- f)*

**SELECT** **S**.**title**

**FROM** story S,

story\_reprint R

**WHERE** **S**.**id** **=** **R**.**origin\_id**

**GROUP BY** **R**.**origin\_id**

**ORDER BY** **COUNT**(**R**.**origin\_id**)

This query (g) was interesting since it uses the chain rule in a particular fashion. What we are seeking are artists who contributed to every part of the making of some story. That is, we want all artist who did color, write, draw and ink a story. We simply want an artist who did all 4 on a same story and a story who had all 4 done by a single artist.

*-- g)*

**SELECT distinct** **A**.**name**

**FROM** artist A,

has\_script SC,

has\_pencils P,

has\_color C,

has\_inked I,

story S

**WHERE** **A**.**id** **=** **SC**.**artist\_id** **AND**

**A**.**id** **=** **P**.**artist\_id** **AND**

**A**.**id** **=** **C**.**artist\_id** **AND**

**A**.**id** **=** **I**.**artist\_id** **AND**

**S**.**id** **=** **SC**.**story\_id** **AND**

**S**.**id** **=** **P**.**story\_id** **AND**

**S**.**id** **=** **C**.**story\_id** **AND**

**S**.**id** **=** **I**.**story\_id**

For the last query h), we wanted Batman to be a non-featured character of a non-reprinted story. For that purpose, we seek all stories which were not reprinted, that is, which are not featured in the reprint table. We hence look for the story in the reprint table and expect the number of its occurrences to be 0.

Then, we simply say we want Batman to be in the non-featured characters. Note the utilization of the command LIKE in order to seek for all strings containing “Batman”.

*-- h)*

**SELECT** **S**.**title**

**FROM** story S,

character C,

has\_characters HS

**WHERE** 0 **=** ( **SELECT** **COUNT**(distinct **R**.**origin\_id**)

**FROM** story\_reprint R

**WHERE** **S**.**id** **=** **R**.**origin\_id**

) **AND**

**HS**.**characters\_id** **=** **C**.**id** **AND**

**HS**.**story\_id** **=** **S**.**id** **AND**

**C**.**name** **LIKE** '%Batman%'

## General Comments