Databases Project – Spring 2017

## Prof. Anastasia Ailamaki

# Team No: 18

Romain Gehrig

Dominique Roduit

Thierry Treyer

Contents

Contents 1

Deliverable 1 3

Assumptions 3

Entity Relationship Schema 4

Schema 5

Description 6

Relational Schema 9

ER schema to Relational schema 9

DDL 10

General Comments 10

Deliverable 2 19

Assumptions 19

Data Loading 19

Query Implementation 20

Query a: 20

Description of logic: 20

SQL statement 20

Interface 23

Design logic Description 23

Screenshots 23

General Comments 27

Deliverable 3 28

Assumptions 28

Query Implementation 28

Query a: 28

Description of logic: 28

SQL statement 28

Query Analysis 33

Selected Queries (and why) 33

Query 1 33

Query 2 33

Query 3 33

Interface 33

General Comments 33

# Deliverable 1

## Assumptions

<In this section write down the assumptions you made about the data. Write a sentence for each assumption you made>

**Story**

We exploded the attributes of this table in many tables, especially for all the fields containing multiples comma separated values. We describe above the replacement of attributes by new tables.

**script, pencils, inks, colors, letters**

These 5 attributes were replaced by 2 new tables :

1. **person**

this table will contain all the persons (artists or editors) who were contained in the 5 attributes above (and in the deleted attribute “editing”, explanation below). It is thus made of the attributes “alias\_id” who refers to a person, to reproduce the fact that sometimes some artists have an alias name. “firstname” and “lastname”.

1. **Participate**

This is the link between person and story to attribute artists to the stories. This single table fully replace the 5 attributes above, each different attribute can be identified by the “role” field whose the type is an enum which express which of the 5 attribute is concerned by the record.

**characters, feature**

2 attributes replaces by 3 tables. On table “hero” which will store the list of all heros, and one intermediate table by attribute which allow to split each comma separated values in many single record linked to the story.

**genre**

This attribute was replaced by two tables : one table “genre” to contain all the genre and avoid duplicata, and an intermediate table “story\_genre” with two identifying relationships, in order to explode the separated values in multiples records. Each genre associated to a story is a record in this table.

**editing**

this attribute was deleted because we can retrieve it via the “editing\_id” attribute linked to its indicia\_publisher, itself link to the issue containing the story. The editor is now in a separated table “edit” containing only editors and linking a person (an editor in this case) to the issue containing the story published by the indicia\_publisher linked to the concerned story. The credits are stored in the “note” attribute of this table.

**Issue**

**valid\_isbn**

this attribute was removed, we only kept “isbn”.

**price**

attribute replaced by a new table “price” containing the issue concerned by the price, the amount (price itself) and the currency in which this price is exprimed (currency is simply an enum, since the values of currency are a finite set of same values).

## Entity Relationship Schema

<In this section you should have figure of the ER schema as well as descriptions about entities and relations>

### Schema



### Description <Describe all the choices you made for Entities and Relationships>

**Story (relations)**

**story.id - story\_reprint.origin\_id**

An origin story (old printed story) is 0 or one story.

A story can be the origin story 0, 1 or many times (if reprinted many times).

Identifying relationship because the origin story printing is itself a story.

**story.id - story\_reprint.target\_id**

A target story (old printed story) is exactly one story.

A story can be the target story 1 or many times (if reprinted many times).

Identifying relationship because the target story printing is itself a story.

**story.type\_id - story\_type.id**

A story can be of zero or one type.

A story type could be attributed to 0 or many stories.

**story.issue\_id – issue.id**

A story is contained by 1 issue

An issue contains 1 or many issues.

**story.id – participate.story\_id**

A person can participate in one or many stories

A story can have one or many artists who participated to its realisation

**story.id – feature.story\_id**

A feature can be part of zero or one story

A story can have none or many feature.

**story.id – character.story\_id**

A character can be attributed to none or many stories

A story have at least one characters, or many.

**story.id – story\_genre.story\_id**

A genre can be attributed to 1 or many stories

A story can be of one or many genre

**Indicia\_publisher.id – issue.indicia\_publisher\_id**

A story can be published by 0 or 1 indicia publisher

An indicia publisher can publish 1 or many stories

**Issue (relations)**

**Issue\_reprint.origin\_id – issue.id**

An origin issue (old printed issue) is exactly one issue.

An issue can be the origin issue 1 or many times (if reprinted many times).

Identifying relationship because the origin issue is itself an issue.

**Issue\_reprint.target\_id – issue.id**

A target (new printed issue) is exactly one issue.

An issue can be the target issue 1 or many times (if reprinted many times).

Identifying relationship because the target issue is itself an issue.

**Indicia\_publisher.id – issue.indicia\_publisher\_id**

An indicia\_publisher can publish 1 or many issue.

An issue can be published by 0 or one indicia publisher

**issue.id – price.issue\_id**

An issue can be sold to 1 or many price

An a price with its currency can be attributed to only one issue.

**publisher (relations)**

**Publisher.id – indicia\_publisher.publisher\_id**

An indicia publisher can be owned by only one publisher.

A publisher can be the owner of 0 or many indicia publishers.

Identifying relationship because an indicia\_publisher depends directly of a publisher, it can't exists without a publisher.

**Publisher.id – brand\_group.publisher\_id**

A brand is owned by exactly one publisher.

A publisher can own 0 or many brands.

Identifying relationship because an brand\_group depends directly of a publisher, it can't exists without a publisher.

**Publisher.id – series.publisher\_id**

A publisher can publish 0, one or many series.

A serie can be published by one publisher.

**Country (relations)**

**Indicia\_publisher.country\_id – country.id**

An indicia publisher comes from one country

A country can be attributed to 0 or many company

**Country.id – publisher.country\_id**

A publisher comes from one country.

A country can be assign to many publisher.

**Series.country\_id – country.id**

A country can be attributed to 0, one or many series.

A serie comes from one country.

**Language (relations)**

**Language.id – series.language\_id**

A language can be attributed to 0, one or many series.

A series can be in one language.

**Series (relations)**

**Series\_publication\_type.id – series.publication\_type\_id**

A publication type can be attributed to 0, one or many series.

A serie can be of one publication type.

**Series.id – issue.series\_id**

An issue belongs to 0 or one serie.

A serie can wrap 1 or many issues.

**Issue.id – series.first\_issue\_id**

It could exists 0 or 1 first issue for a serie

An issue can be the first issue of 0, 1 or many series.

**Issue.id – series.last\_issue\_id**

It could exists 0 or 1 last issue for a serie

An issue can be the last issue of 0, 1 or many series.

## Relational Schema

### ER schema to Relational schema

<Describe the transition from ER schema to Relational schema>



The relations made only of single strokes as above (thin or bold but without arrows), generate the creation of a new table containing two foreign keys who link together the two tables concerned.

C:\Users\Dominique\AppData\Local\Microsoft\Windows\INetCache\Content.Word\notnullrelation.png

The relations who contains one arrow are reduced to a NON NULL foreign key in the table in front of the arrow.

In the example above, the relation could be translated by a NON NULL foreign key in the table B.

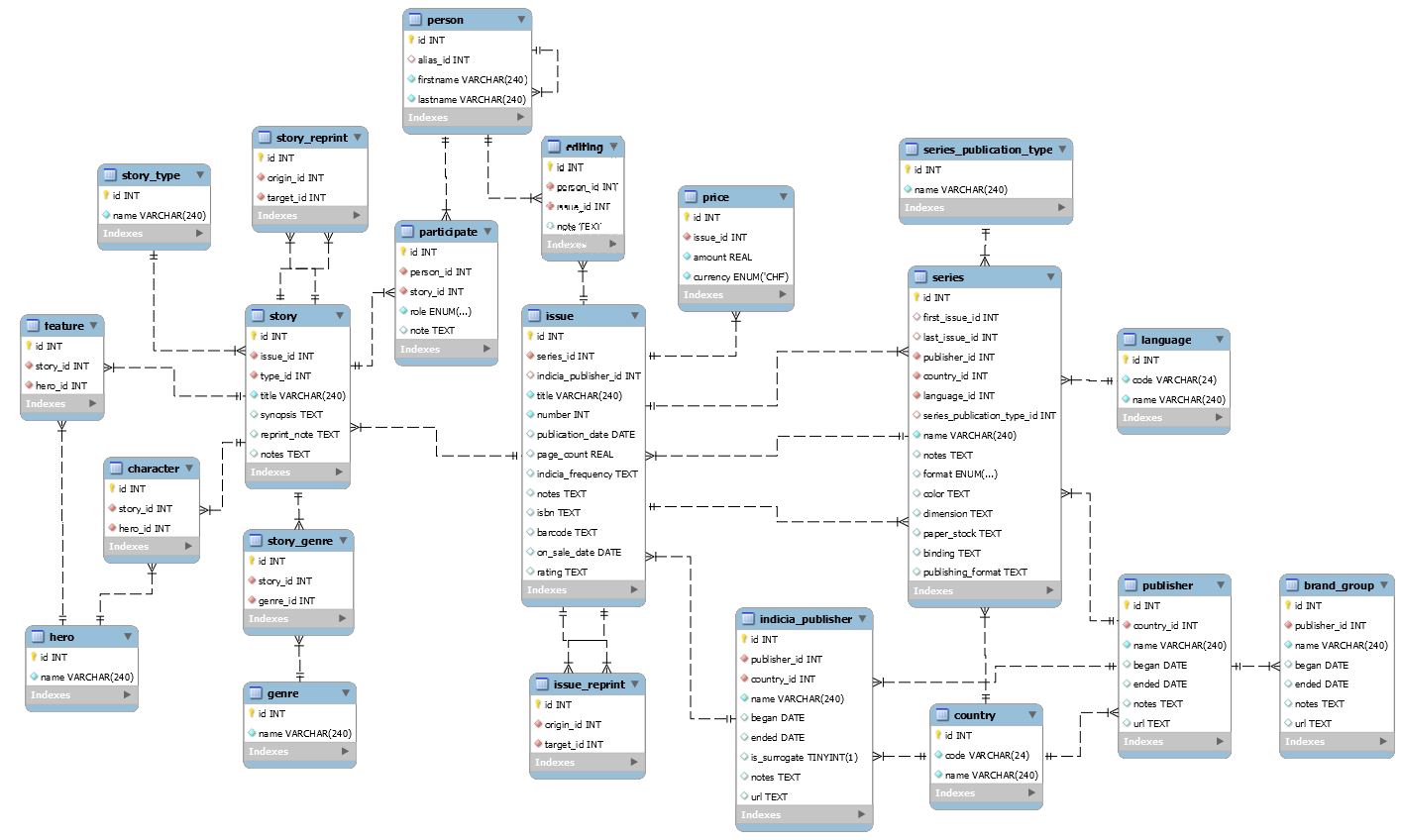


In the relations who are only made of one-one (two bold arrows) and zero-one (one bold and one thin arrows as in this example) constraints, the foreign key must be UNIQUE.

In the example above, the relation could be translated by a UNIQUE foreign key in the table A.

The participation constraints “at least one” (denoted by a bold stroke) can’t be captured in SQL without check constraint or assertion.

### 



### DDL

-- -----------------------------------------------------

-- Table story\_type

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** story\_type **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

name **VARCHAR(**240**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**)**

**);**

-- -----------------------------------------------------

-- Table country

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** country **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

code **VARCHAR(**24**)** **NOT** **NULL,**

name **VARCHAR(**240**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**)**

**);**

-- -----------------------------------------------------

-- Table publisher

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** publisher **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

country\_id **INT** **NOT** **NULL,**

name **VARCHAR(**240**)** **NOT** **NULL,**

began **DATE** **NULL** **COMMENT** ' '**,**

ended **DATE** **NULL,**

notes TEXT **NULL,**

url TEXT **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** country\_country\_id\_idx **(**country\_id **ASC),**

**CONSTRAINT** country\_country\_id

**FOREIGN** **KEY** **(**country\_id**)**

**REFERENCES** country **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

-- -----------------------------------------------------

-- Table indicia\_publisher

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** indicia\_publisher **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

publisher\_id **INT** **NOT** **NULL,**

country\_id **INT** **NOT** **NULL,**

name **VARCHAR(**240**)** **NOT** **NULL,**

began **DATE** **NULL,**

ended **DATE** **NULL,**

is\_surrogate **TINYINT(**1**)** **NULL** **COMMENT** 'A Boolean indicating whether the indicia publisher is a company related to the master publisher or an unrelated company who published on behalf of the master publisher.'**,**

notes TEXT **NULL,**

url TEXT **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** country\_country\_id\_idx **(**country\_id **ASC),**

**INDEX** publisher\_publisher\_id\_idx **(**publisher\_id **ASC),**

**CONSTRAINT** country\_country\_id

**FOREIGN** **KEY** **(**country\_id**)**

**REFERENCES** country **(**id**)**

**ON** **DELETE** **RESTRICT** **,**

**CONSTRAINT** publisher\_publisher\_id

**FOREIGN** **KEY** **(**publisher\_id**)**

**REFERENCES** publisher **(**id**)**

**ON** **DELETE** **RESTRICT**

**);**

-- -----------------------------------------------------

-- Table language

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** **language** **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

code **VARCHAR(**24**)** **NOT** **NULL,**

name **VARCHAR(**240**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**)**

**);**

-- -----------------------------------------------------

-- Table series\_publication\_type

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** series\_publication\_type **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

name **VARCHAR(**240**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**)**

**);**

-- -----------------------------------------------------

-- Table series

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** series **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

first\_issue\_id **INT** **NULL,**

last\_issue\_id **INT** **NULL,**

publisher\_id **INT** **NOT** **NULL,**

country\_id **INT** **NOT** **NULL,**

language\_id **INT** **NOT** **NULL,**

series\_publication\_type\_id **INT** **NULL,**

name **VARCHAR(**240**)** **NOT** **NULL,**

notes TEXT **NULL,**

**format** ENUM**(**'Hardcover'**,** 'Comics'**)** **NULL** **COMMENT** 'Description of physical format (Hardcover, comics, etc.)'**,**

color TEXT **NULL,**

dimension TEXT **NULL,**

paper\_stock TEXT **NULL,**

binding TEXT **NULL,**

publishing\_format TEXT **NULL** **COMMENT** 'UNKNOWN FIELD IN THE WIKI'**,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** issue\_first\_issue\_id\_idx **(**first\_issue\_id **ASC),**

**INDEX** issue\_last\_issue\_id\_idx **(**last\_issue\_id **ASC),**

**INDEX** country\_country\_id\_idx **(**country\_id **ASC),**

**INDEX** language\_language\_id\_idx **(**language\_id **ASC),**

**INDEX** publisher\_publisher\_id\_idx **(**publisher\_id **ASC),**

**INDEX** series\_publication\_type\_publication\_type\_idx **(**series\_publication\_type\_id **ASC),**

**CONSTRAINT** issue\_first\_issue\_id

**FOREIGN** **KEY** **(**first\_issue\_id**)**

**REFERENCES** issue **(**id**)**

**ON** **DELETE** **SET** **NULL,**

**CONSTRAINT** issue\_last\_issue\_id

**FOREIGN** **KEY** **(**last\_issue\_id**)**

**REFERENCES** issue **(**id**)**

**ON** **DELETE** **SET** **NULL,**

**CONSTRAINT** country\_country\_id

**FOREIGN** **KEY** **(**country\_id**)**

**REFERENCES** country **(**id**)**

**ON** **DELETE** **RESTRICT,**

**CONSTRAINT** language\_language\_id

**FOREIGN** **KEY** **(**language\_id**)**

**REFERENCES** **language** **(**id**)**

**ON** **DELETE** **RESTRICT,**

**CONSTRAINT** publisher\_publisher\_id

**FOREIGN** **KEY** **(**publisher\_id**)**

**REFERENCES** publisher **(**id**)**

**ON** **DELETE** **RESTRICT,**

**CONSTRAINT** series\_publication\_type\_publication\_type\_id

**FOREIGN** **KEY** **(**series\_publication\_type\_id**)**

**REFERENCES** series\_publication\_type **(**id**)**

**ON** **DELETE** **SET** **NULL**

**);**

-- -----------------------------------------------------

-- Table issue

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** issue **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

series\_id **INT** **NOT** **NULL,**

indicia\_publisher\_id **INT** **NULL,**

title **VARCHAR(**240**)** **NOT** **NULL,**

**number** **INT** **NOT** **NULL** **COMMENT** 'Issue number'**,**

publication\_date **DATE** **NULL** **COMMENT** 'May be approximate (circa 1870) or a precise date'**,**

page\_count **REAL** **NULL** **COMMENT** 'Is a real number!?!'**,**

indicia\_frequency TEXT **NULL** **COMMENT** 'Publication frequency (monthly, quarterly, etc.)'**,**

notes TEXT **NULL** **COMMENT** 'Freeform notes'**,**

isbn TEXT **NULL** **COMMENT** 'UNKNOWN FIELD IN WIKI'**,**

barcode TEXT **NULL,**

on\_sale\_date **DATE** **NULL** **COMMENT** 'UNKNOWN FIELD IN WIKI'**,**

rating TEXT **NULL** **COMMENT** 'Approval and age rating'**,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** indicia\_publisher\_indica\_publisher\_id\_idx **(**indicia\_publisher\_id **ASC),**

**INDEX** series\_series\_id\_idx **(**series\_id **ASC),**

**CONSTRAINT** indica\_publisher\_id

**FOREIGN** **KEY** **(**indicia\_publisher\_id**)**

**REFERENCES** indicia\_publisher **(**id**)**

**ON** **DELETE** **SET** **NULL,**

**CONSTRAINT** series\_series\_id

**FOREIGN** **KEY** **(**series\_id**)**

**REFERENCES** series **(**id**)**

**ON** **DELETE** **RESTRICT**

**);**

-- -----------------------------------------------------

-- Table story

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** story **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

issue\_id **INT** **NOT** **NULL,**

type\_id **INT** **NOT** **NULL,**

title **VARCHAR(**240**)** **NOT** **NULL** **COMMENT** 'Name of the story'**,**

synopsis TEXT **NULL,**

reprint\_note TEXT **NULL,**

notes TEXT **NULL** **COMMENT** 'Arbitrary notes'**,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** story\_type\_type\_id\_idx **(**type\_id **ASC),**

**INDEX** issue\_issue\_id\_idx **(**issue\_id **ASC),**

**CONSTRAINT** story\_type\_type\_id

**FOREIGN** **KEY** **(**type\_id**)**

**REFERENCES** story\_type **(**id**)**

**ON** **DELETE** **RESTRICT,**

**CONSTRAINT** issue\_issue\_id

**FOREIGN** **KEY** **(**issue\_id**)**

**REFERENCES** issue **(**id**)**

**ON** **DELETE** **RESTRICT**

**);**

-- -----------------------------------------------------

-- Table story\_reprint

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** story\_reprint **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

origin\_id **INT** **NOT** **NULL,**

target\_id **INT** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** story\_origin\_id\_idx **(**origin\_id **ASC),**

**INDEX** story\_target\_id\_idx **(**target\_id **ASC),**

**CONSTRAINT** story\_origin\_id

**FOREIGN** **KEY** **(**origin\_id**)**

**REFERENCES** story **(**id**)**

**ON** **DELETE** **CASCADE,**

**CONSTRAINT** story\_target\_id

**FOREIGN** **KEY** **(**target\_id**)**

**REFERENCES** story **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

-- -----------------------------------------------------

-- Table issue\_reprint

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** issue\_reprint **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

origin\_id **INT** **NOT** **NULL,**

target\_id **INT** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** issue\_origin\_id\_idx **(**origin\_id **ASC),**

**INDEX** issue\_target\_id\_idx **(**target\_id **ASC),**

**CONSTRAINT** issue\_origin\_id

**FOREIGN** **KEY** **(**origin\_id**)**

**REFERENCES** issue **(**id**)**

**ON** **DELETE** **CASCADE,**

**CONSTRAINT** issue\_target\_id

**FOREIGN** **KEY** **(**target\_id**)**

**REFERENCES** issue **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

-- -----------------------------------------------------

-- Table brand\_group

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** brand\_group **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

publisher\_id **INT** **NOT** **NULL,**

name **VARCHAR(**240**)** **NOT** **NULL,**

began **DATE** **NULL,**

ended **DATE** **NULL,**

notes TEXT **NULL,**

url TEXT **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** publisher\_publisher\_id\_idx **(**publisher\_id **ASC),**

**CONSTRAINT** publisher\_publisher\_id

**FOREIGN** **KEY** **(**publisher\_id**)**

**REFERENCES** publisher **(**id**)**

**ON** **DELETE** **RESTRICT**

**);**

-- -----------------------------------------------------

-- Table hero

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** hero **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

name **VARCHAR(**240**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**)**

**);**

-- -----------------------------------------------------

-- Table feature

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** feature **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

story\_id **INT** **NOT** **NULL,**

hero\_id **INT** **NOT** **NULL,**

**INDEX** story\_story\_id\_idx **(**story\_id **ASC),**

**INDEX** hero\_hero\_id\_idx **(**hero\_id **ASC),**

**PRIMARY** **KEY** **(**id**),**

**CONSTRAINT** hero\_hero\_id

**FOREIGN** **KEY** **(**hero\_id**)**

**REFERENCES** hero **(**id**)**

**ON** **DELETE** **CASCADE,**

**CONSTRAINT** story\_story\_id

**FOREIGN** **KEY** **(**story\_id**)**

**REFERENCES** story **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

-- -----------------------------------------------------

-- Table person

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** person **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

alias\_id **INT** **NULL,**

firstname **VARCHAR(**240**)** **NOT** **NULL,**

lastname **VARCHAR(**240**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** person\_alias\_id\_idx **(**alias\_id **ASC),**

**CONSTRAINT** person\_alias\_id

**FOREIGN** **KEY** **(**alias\_id**)**

**REFERENCES** person **(**id**)**

**ON** **DELETE** **SET** **NULL**

**);**

-- -----------------------------------------------------

-- Table character

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** **character** **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

story\_id **INT** **NOT** **NULL,**

hero\_id **INT** **NOT** **NULL,**

**INDEX** story\_story\_id\_idx **(**story\_id **ASC),**

**INDEX** hero\_hero\_id\_idx **(**hero\_id **ASC),**

**PRIMARY** **KEY** **(**id**),**

**CONSTRAINT** hero\_hero\_id

**FOREIGN** **KEY** **(**hero\_id**)**

**REFERENCES** hero **(**id**)**

**ON** **DELETE** **CASCADE,**

**CONSTRAINT** story\_story\_id

**FOREIGN** **KEY** **(**story\_id**)**

**REFERENCES** story **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

-- -----------------------------------------------------

-- Table genre

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** genre **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

name **VARCHAR(**240**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**)**

**);**

-- -----------------------------------------------------

-- Table story\_genre

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** story\_genre **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

story\_id **INT** **NOT** **NULL,**

genre\_id **INT** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** story\_story\_id\_idx **(**story\_id **ASC),**

**INDEX** genre\_genre\_id\_idx **(**genre\_id **ASC),**

**CONSTRAINT** story\_story\_id

**FOREIGN** **KEY** **(**story\_id**)**

**REFERENCES** story **(**id**)**

**ON** **DELETE** **CASCADE,**

**CONSTRAINT** genre\_genre\_id

**FOREIGN** **KEY** **(**genre\_id**)**

**REFERENCES** genre **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

-- -----------------------------------------------------

-- Table price

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** price **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

issue\_id **INT** **NOT** **NULL,**

amount **REAL** **NOT** **NULL,**

-- We must put in the ENUM() the list of currencies

currency ENUM**(**'CHF'**)** **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** issue\_issue\_id\_idx **(**issue\_id **ASC),**

**CONSTRAINT** issue\_issue\_id

**FOREIGN** **KEY** **(**issue\_id**)**

**REFERENCES** issue **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

-- -----------------------------------------------------

-- Table participate

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** participate **(**

id **INT** **NOT** **NULL,**

person\_id **INT** **NOT** **NULL,**

story\_id **INT** **NOT** **NULL,**

role ENUM**(**'script'**,** 'pencil'**,** 'ink'**,** 'color'**,** 'letter'**,** 'editing'**)** **NOT** **NULL,**

note TEXT **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** person\_person\_id\_idx **(**person\_id **ASC),**

**INDEX** story\_story\_id\_idx **(**story\_id **ASC),**

**CONSTRAINT** person\_person\_id

**FOREIGN** **KEY** **(**person\_id**)**

**REFERENCES** person **(**id**)**

**ON** **DELETE** **CASCADE,**

**CONSTRAINT** story\_story\_id

**FOREIGN** **KEY** **(**story\_id**)**

**REFERENCES** story **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

-- -----------------------------------------------------

-- Table editing

-- -----------------------------------------------------

**CREATE** **TABLE** **IF** **NOT** **EXISTS** editing **(**

id **INT** **NOT** **NULL** AUTO\_INCREMENT**,**

person\_id **INT** **NOT** **NULL,**

issue\_id **INT** **NOT** **NULL,**

note TEXT **NULL,**

**PRIMARY** **KEY** **(**id**),**

**INDEX** person\_person\_id\_idx **(**person\_id **ASC),**

**INDEX** issue\_issue\_id\_idx **(**issue\_id **ASC),**

**CONSTRAINT** person\_person\_id

**FOREIGN** **KEY** **(**person\_id**)**

**REFERENCES** person **(**id**)**

**ON** **DELETE** **CASCADE,**

**CONSTRAINT** issue\_issue\_id

**FOREIGN** **KEY** **(**issue\_id**)**

**REFERENCES** issue **(**id**)**

**ON** **DELETE** **CASCADE**

**);**

## General Comments

<In this section write general comments about your deliverable (comments and work allocation between team members>

**Thierry** : Relational Schema

**Dominique** : ER scheme and review / modifications / corrections of the relational schema (type of data, split of some tables and cardinality of relations). Filling of this project template.

# Deliverable 2

## Assumptions

<In this section write down the assumptions you made about the data. Write a sentence for each assumption you made>

I corrected a lot of things from part 1 according to the comments done after the feedback, and wrote several php scripts to load the data into the database. The data had to be formatted to fit into our data type for some fields (for example for publication\_date who contains a lot of different date format, sometimes in the form of text). I also had the need to wrap the values of each lines of the csv with quotes, in order to read easily each line, since with comma as a separator, all commas into the quotes didn’t have to be considered as a separator.

## Data Loading

For the small files, it was relatively straightforward.

To treat the big files, however, I mainly wrote 3 scripts :

* **An “error detector”** : who read each line of the csv and if something abnormal is detected in the line, write it in a file for errors, otherwise write it in a file for good lines. (errors can be for example a number of values who doesn’t match with the header, often due to the fact that we read each values with the “,” separator and if the values are not wrapped with quotes, we read the values in a wrong way)
* **A chunker** : who split the big files in multiple files containing 100’000 records each.
* **A “parser”/”inserter”** : who read each chunk separately, construct the SQL queries for insertions (by groups of 500 insertions in once), and execute the queries on the database.

First, after error detector, I corrected the files containing lines with error (with a script who added quotes to the values, to avoid reading a comma in a value like a separator), and then applied this 3 scripts on the files. If an error occurred in the insertion, I displayed the errors, and with some try all happened very well. Once the data loaded in the database I checked that the number of records was the same that the number of lines in the csv files.

Other techniques like manipulating the data with SQL and INSERTION of a SELECT statement were also used but it’s sometimes mainly DIY.

### Import of tables: hero, characters

Python script to generate 2 csv files : One containing a mapping (hero\_id, hero\_name), and the other containing a mapping (story\_id, hero\_id).

stories **=** open**(**"story.csv"**)**

rstories **=** csv**.**DictReader**(**stories**)**

heroes **=** defaultdict**(**list**)**

**for** i**,**line **in** enumerate**(**rstories**):**

**if** **not** line**[**"characters"**]:**

**continue**

**for** c **in** line**[**"characters"**].**split**(**";"**):**

trimed **=** c**.**strip**()**

**if** trimed**:**

heroes**[**trimed**].**append**(**line**[**"id"**])**

sheroes **=** sorted**(**heroes**.**keys**())**

hero\_file **=** open**(**"heros.csv"**,** "w"**)**

hero\_writer **=** csv**.**writer**(**hero\_file**)**

hero\_story\_file **=** open**(**"hero\_story.csv"**,** "w"**)**

hero\_story\_writer **=** csv**.**writer**(**hero\_story\_file**)**

hero\_writer**.**writerow**([**"id"**,**"name"**])**

hero\_story\_writer**.**writerow**([**"story\_id"**,** "hero\_id"**])**

**for** i**,**hero **in** enumerate**(**sheroes**,**start**=**1**):**

hero\_writer**.**writerow**([**i**,**hero**])**

**for** sid **in** heroes**[**hero**]:**

hero\_story\_writer**.**writerow**([**sid**,**i**])**

## Query Implementation

### Query a: Print the brand group names with the highest number of Belgian indicia publishers

#### Description of logic:

What I want is to count how many indicia\_publisher from Belgium each publisher has, and then to pick the publisher who has the most indicia publisher in Belgium, and finally to retrieve the associed brang group.

To do so, we proceed from the inner to the outer of the request below. First, I select the country id for Belgium. Then, I select all indicia publishers who has begium for country id. I make a join with the corresponding publishers in order to be able to retrieve the corresponding brand\_group, and I make a group by to have the count of indicia publisher associate to each publisher. I sorted by the number of indicia publisher associate in descending order and kept only the first result to have the max.

The query displays 15 results.

#### SQL statement

**SELECT** name **FROM** brand\_group **WHERE** publisher\_id **=** **(**

**SELECT** p**.**id **FROM** indicia\_publisher **as** ind

**INNER** **JOIN** publisher **as** p **ON** ind**.**publisher\_id **=** p**.**id

**WHERE** ind**.**country\_id **=** **(SELECT** id **FROM** country **WHERE** name **=** 'Belgium'**)**

**GROUP** **BY** p**.**id

**ORDER** **BY** **count(**p**.**id**)** **DESC**

**LIMIT** 1

**)**

*Results:* 15 records – are shown here only the firsts



### Query b: Print the ids and names of publishers of Danish book series.

#### Description of logic:

We first select the country id for Denmark, then select the publisher id of all series from this country, and retain only the id and name of the publishers in this list.

The query displays 112 results.

#### SQL statement

**SELECT** id**,** name **FROM** publisher **WHERE** id **IN** **(**

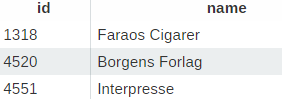
**SELECT** publisher\_id **FROM** series

**WHERE** country\_id **=** **(SELECT** id **FROM** country **WHERE** name **=** 'Denmark'**)** **AND**

publication\_type\_id **=** **(SELECT** id **FROM** series\_publication\_type **WHERE** name**=**'book'**)**

**)**

*Results:* 26 records – are shown here only the firsts



### Query c: Print the names of all Swiss series that have been published in magazines.

#### Description of logic:

I first retrieve the country id for Switzerland and select all series from this country. Then I select the id for the publication type “magazine” and keep only the series with this publication type.

#### SQL statement

**SELECT** name **FROM** series **WHERE**

country\_id **=** **(SELECT** id **FROM** country **WHERE** code **=** 'ch'**)** **AND**

publication\_type\_id **=** **(SELECT** id **FROM** series\_publication\_type **WHERE** name **=** 'magazine'**)**

*Results:* 5 records

### 

### Query d: Starting from 1990, print the number of issues published each year

#### Description of logic:

We first display only issues with publication date greater than 1990, and keep only the year and count of records for each years in doing a group by years. We finally group by publication date.

#### SQL statement

**SELECT** **YEAR(**publication\_date**)** **as** years**,** **count(**id**)** **as** number **FROM** issue

**WHERE** **YEAR(**publication\_date**)** **>=** 1990

**GROUP** **BY** years

**ORDER** **BY** publication\_date **ASC**

*Results:* 31 records – are shown here only the firsts



### Query e: Print the number of series for each indicia publisher whose name resembles ‘DC comics’.

#### Description of logic:

**Solving** : one can firstly search all indicia publishers who contains “DC comics” in their names with the LIKE statement (with % at the beginning who means that something can be written before “DC comics” and the % at the end that something can be written after “DC comics”). We can then make a join to list all series associated to the indicia publishers, and finally group by name to have the number of series for each indicia publisher.

#### SQL statement

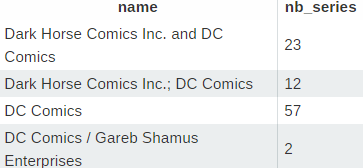
**SELECT** ind**.**name **as** name**,** **count(**ind**.**id**)** **as** nb\_series **FROM** indicia\_publisher ind

**INNER** **JOIN** series s **ON** s**.**publisher\_id **=** ind**.**publisher\_id

**WHERE** ind**.**name **LIKE** '%DC comics%'

**GROUP** **BY** ind**.**name

*Results:* 11 records – are shown here only the firsts



### Query f: Print the titles of the 10 most reprinted stories

#### Description of logic:

**Solving** : We have to list all the reprint for all stories, and to group by stories to count the number of reprint for each story. Afterward, we can order by the number of reprint in descending order and keep only the 10 first result (the 10 higher number of reprint).

**Results** : there is a story with NULL title (in fact there is a lot of stories with NULL title), and 3 stories with the title Spider Man written in 3 different ways, but it’s normal, I checked and it’s really 3 different stories and all reprint for these 3 stories have no link.

#### SQL statement

**SELECT** s**.**title **FROM** story\_reprint sr

**INNER** **JOIN** story s **ON** sr**.**origin\_id **=** s**.**id

**GROUP** **BY** s**.**id

**ORDER** **BY** **COUNT(**s**.**id**)** **DESC**

**LIMIT** 10

*Results:* 10 records – are shown here only the firsts



### Query g: Print the artists that have scripted, drawn, and colored at least one of the stories they were involved in.

#### Description of logic:

I firstly select (inner select statement) all instances of participation who has for roles (script, pencil, ink, color), and group them by id of the person, and by id of story to have the number of roles associated to each persons for each stories, and then keep only the persons who have 4 associated results for a story (that is, for a story, we have 4 results associated to the 4 roles enumerated above). Finally, we select the ids of the persons corresponding to the results of the inner select statement, and distinct by the name to have only one time the same person (because an artist might have scripted, drawn and colored more than one story).

#### SQL statement

**SELECT** **distinct** name **FROM** person **WHERE** id **IN** **(**

**SELECT** person\_id **FROM** participate

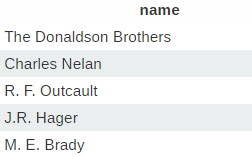
**WHERE** role **IN** **(**'script'**,** 'pencil'**,** 'ink'**,** 'color'**)**

**GROUP** **BY** person\_id**,** story\_id

**HAVING** **count(**person\_id**)** **=** 4

**)**

*Results:* 6 records – are shown here only the firsts



### Query h: Print all non-reprinted stories involving Batman as a non-featured character.

#### Description of logic:

First, one can write the inner select statement who will retrieve the ids of the stories containing Batman as character and not as a feature. To do so, we select from hero the row for which the name is Batman, and do a join on character to retrieve the id of all stories associated to this character. Then, we do another join on feature, but a RIGHT join to keep only the stories who doesn’t contain this hero as a feature. And we group by stories to have only one id of each stories who match these conditions.

Then, we select the stories who has the ids in the list of the results from the inner select statement, and do a left join on story\_reprint to have the associated reprint or NONE as origin\_id if the story doesn’t contain any reprint. We thus retain only the rows who has a NULL origin\_id.

#### SQL statement

**SELECT** **distinct** **s.title** **FROM** story s

**LEFT** **JOIN** story\_reprint sr **ON** sr**.**origin\_id **=** s**.**id

**WHERE** sr**.**origin\_id **IS** **NULL** **AND** s**.**id **IN** **(**

**SELECT** c**.**story\_id **FROM** hero h

**LEFT** **JOIN** characters c **ON** c**.**hero\_id **=** h**.**id

**RIGHT** **JOIN** feature f **ON** f**.**story\_id **=** c**.**story\_id **AND** f**.**hero\_id **!=** h**.**id

**WHERE** h**.**name **=** 'Batman'

**GROUP** **BY** c**.**story\_id

**)**

*Results:* 0 records

## Interface

### Design logic Description

**Design logic**:

**Insertion / Deletion**

We can choose the table into which we want to insert some data with a custom handmade select component, and according to the choice, an include is done to display the appropriate form for each tables. However, a class Table was written in order to generate the insertion sql statement and do it automatically, without writing the sql statement for every table. This class also allow to display the 5 last records, in order to display our newly entered record once the “Insert” button is clicked.

**Search :**

The search is done using a lot of javascript (jquery), in order to retrieve the results in a parallel way. Firstly, we send to a first php script the search key and the tables the user want to search in (by default all tables at the beginning). This php script will then build a search request for each table (it checks if the search key is just a numerical argument, or a text and search strictly the number entered if it’s a numerical key, or in all varchar and text field if it’s a text). When the queries are built, they are sent to another php script who will manage the execution of the query and yields the results. Each time a table returns its results, I update a progressbar who shows in which table we are currently searching, and the remaining part of work to do. But usually, the progressbar is very quickly filled because the request are rather fast. For a numerical search, the result is instantaneous.

We display a limit of 5 results for each table. If there is more of 5 results in a table, we mentioned that we only display the 5 first results, and we give the possibility to display all the results for the table in a floating window loaded over the main view, in order to keep the search interface clean and user-friendly.

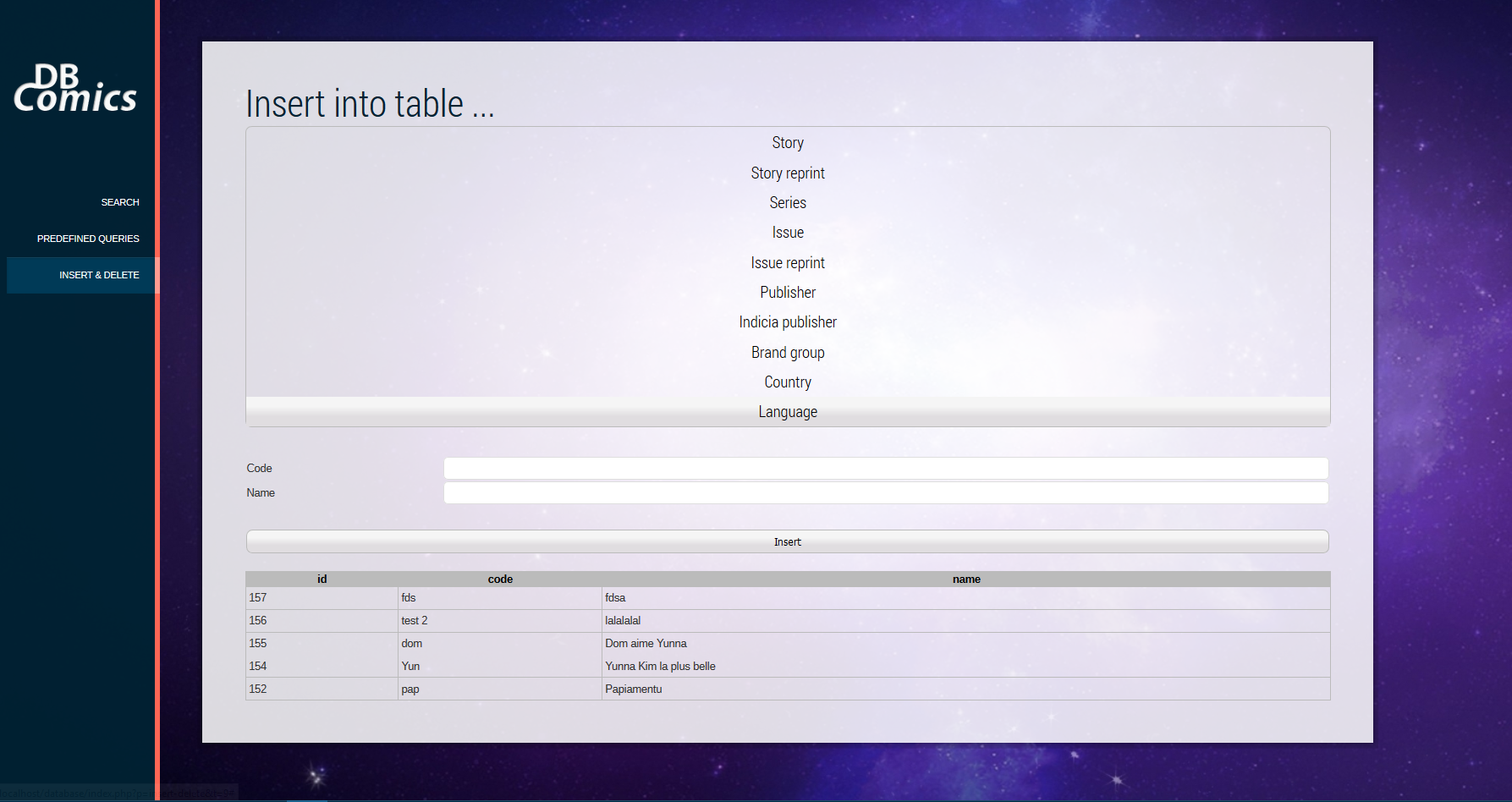
The table in which it doesn’t make sense to search are not displayed in the list of searchable table (all intermediate table who contains only foreign keys). However, all is retrieved automatically from the database. That is, nothing is written by hand, all list of tables and list of fields for each tables are directly given by the database itself. This way, we could add field or table and our interface will always be updated.

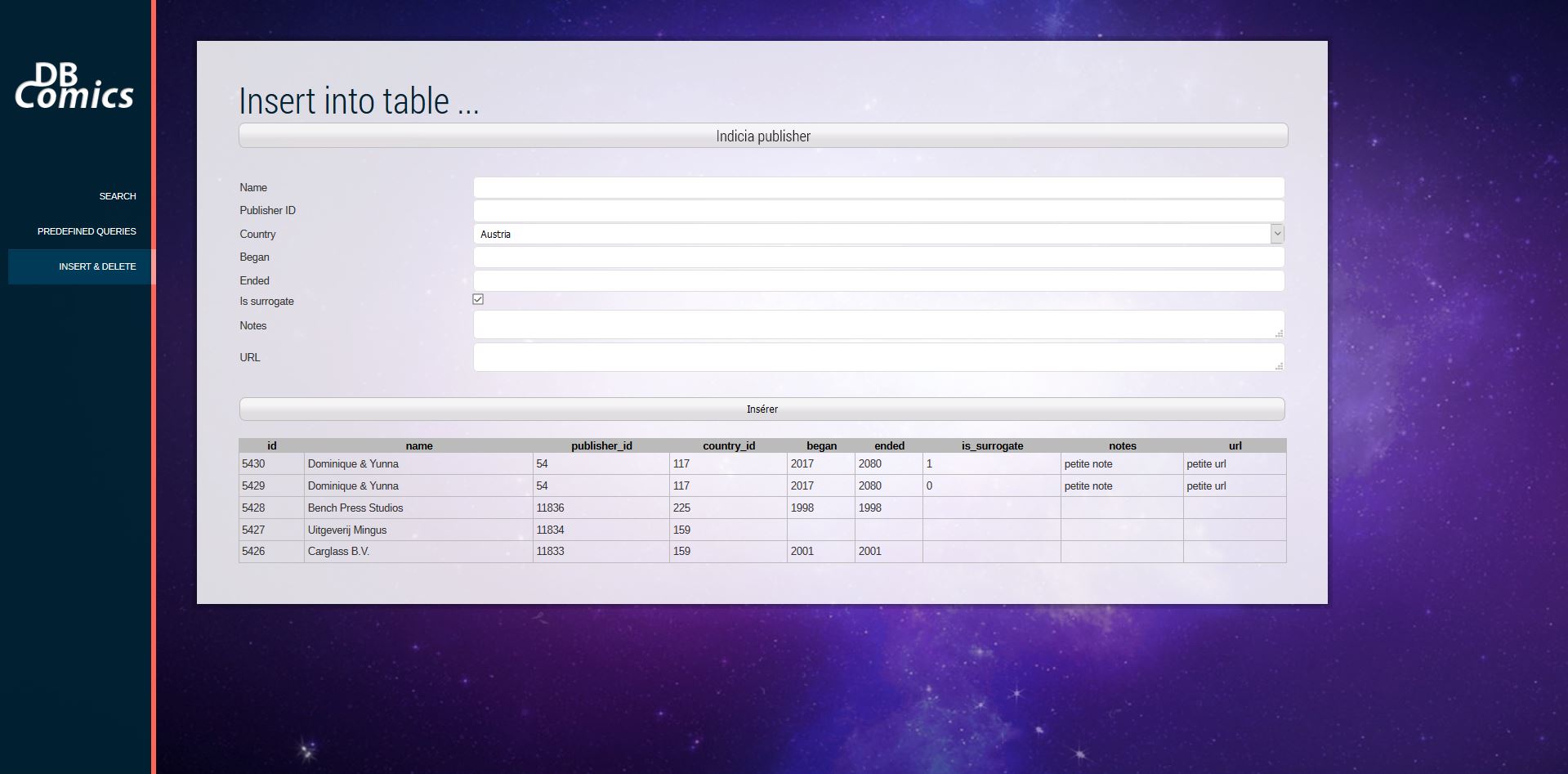
**Technologies used** : MySQL for the DB, PHP, with of course CSS, HTML, and JS for the interface.

### Screenshots

**Insertion / deletion**

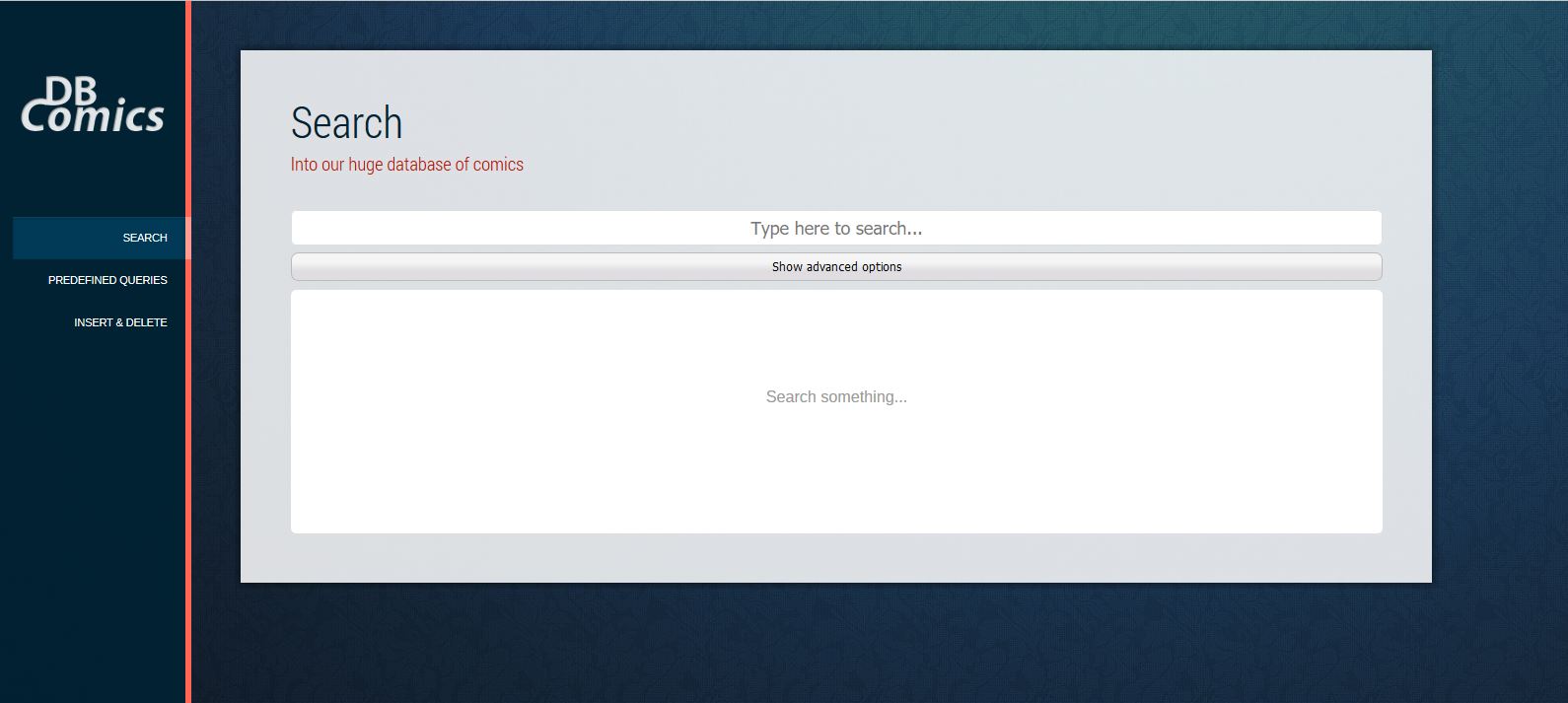
One can select the table we want to insert into …



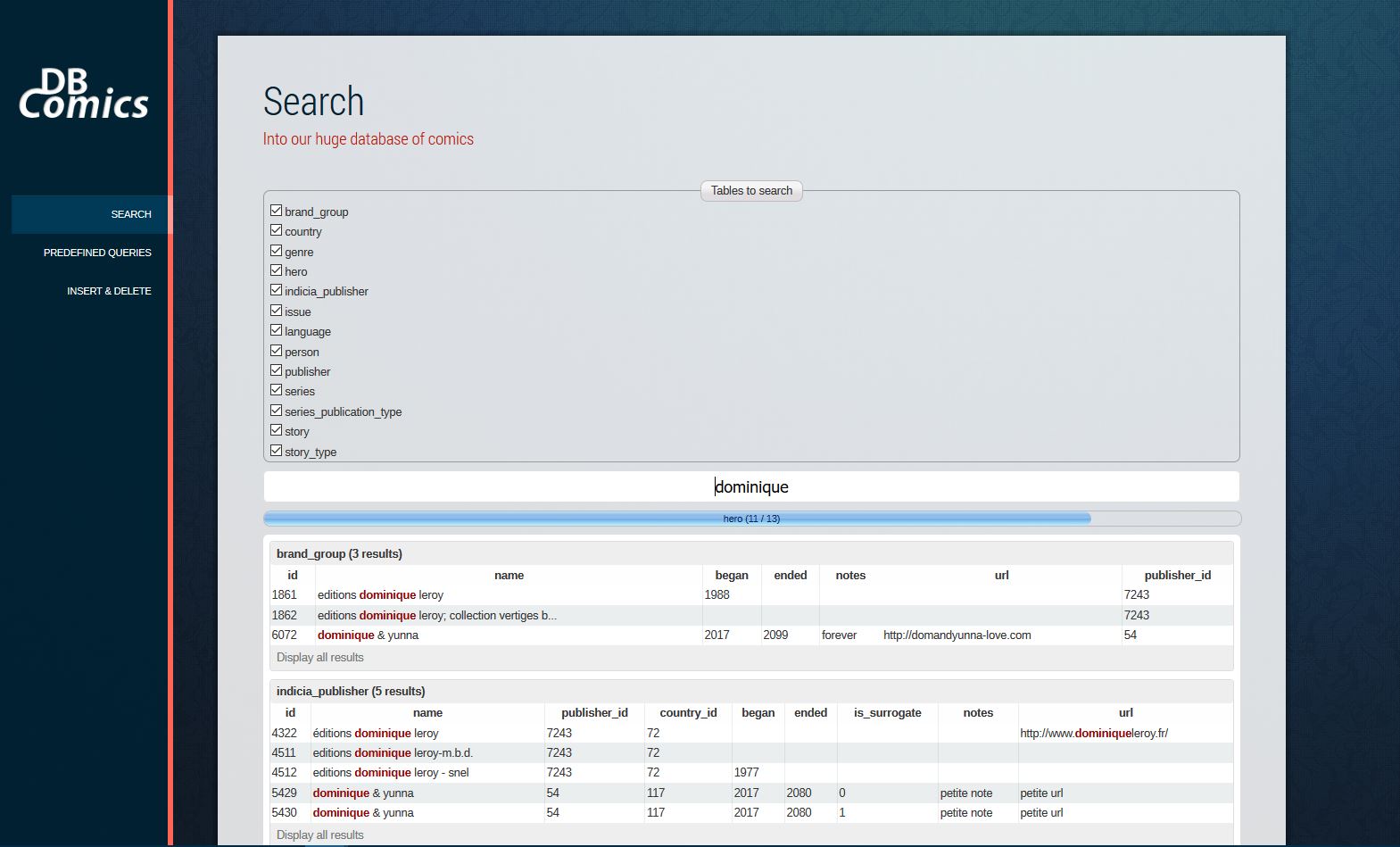


**Search**

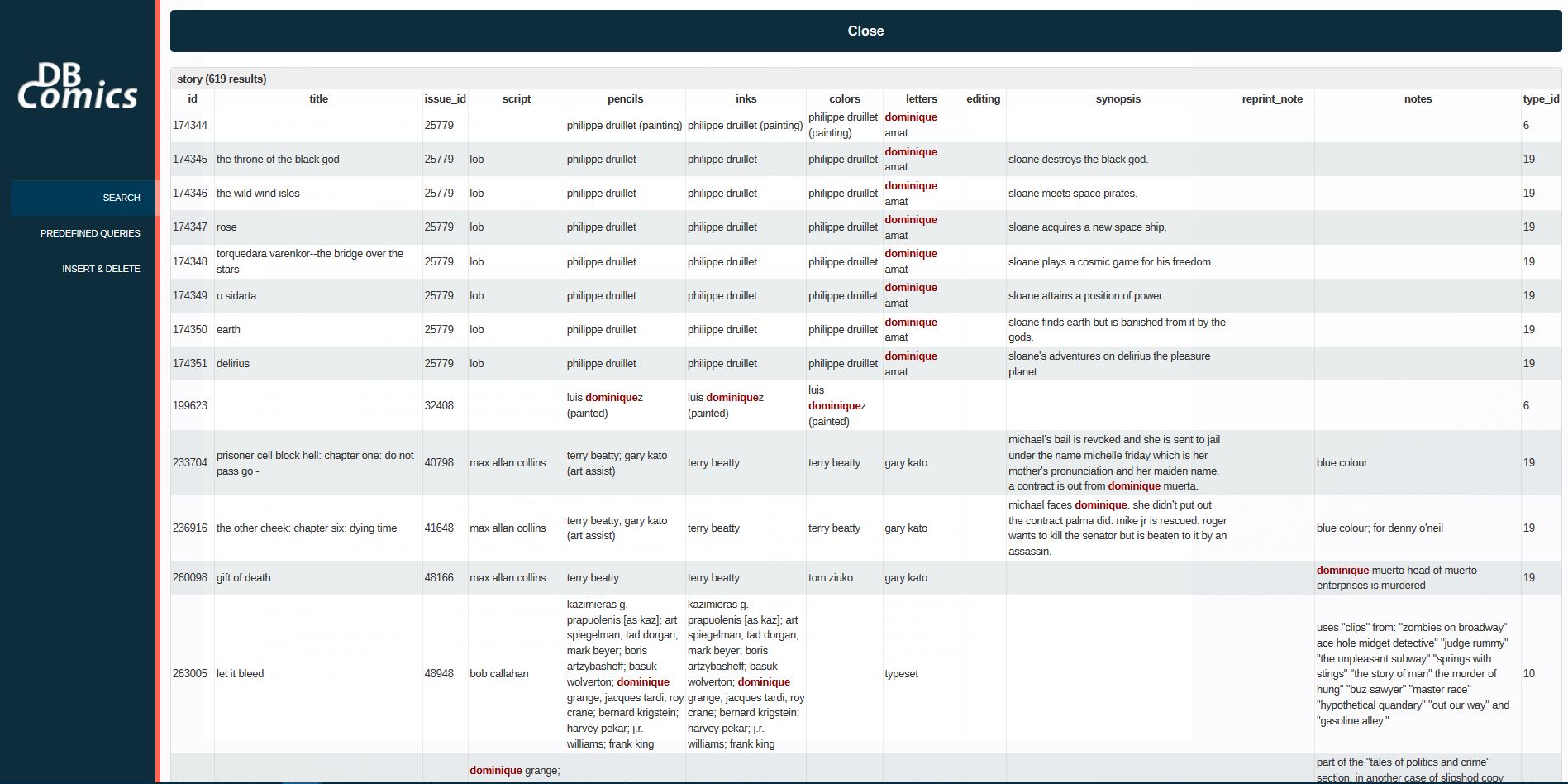
One can type what we are searching for …



Then, the results are given for each table. A progressbar shows the progression of the research, but globally the running time is very fast, since the research is done in parallel for each table (using AJAX).



By default, a maximum of 5 results per table are displayed (to keep the interface clean and confortable). We have the possibility to display all the results for a given table, by clicking on the “Display all results” button.



## General Comments

<In this section write general comments about your deliverable (comments and work allocation between team members>

* Data loading for all tables except : character, editing, feature, participate, person
* All queries implementations
* Interface : insert , search : all is handmade

# Deliverable 3

# Assumptions

<In this section write down the assumptions you made about the data. Write a sentence for each assumption you made>

## Query Implementation

### Query a: Print the series names that have the highest number of issues which contain a story whose type (e.g., cartoon) is not the one occurring most frequently in the database (e.g, illustration).

#### Description of logic:

We use a nested schema to filter the data as early as possible. In the first nested query (the innermost), we select the type occurring the most frequently in the database. We then select the issue ids of all stories who are not of this type.

**Indexes**:

An index on the field story.type\_id is important. Without this index, the query would take more than 45 seconds. However, we don’t have to keep an index on story.issue\_id, otherwise the execution time become very bad.

Execution time : 13.416 seconds

#### SQL statement

**SELECT** name**,** temp**.**nb **FROM** series se

**INNER** **JOIN** **(**

**SELECT** series\_id**,** **COUNT(\*)** **AS** nb **FROM** issue

**INNER** **JOIN** **(**

**SELECT** **distinct** issue\_id **as** iid **FROM** story

**WHERE** type\_id **!=** **(**

**SELECT** type\_id **FROM** story

**GROUP** **BY** type\_id

**ORDER** **BY** **count(\*)** **DESC**

**LIMIT** 1

**)**

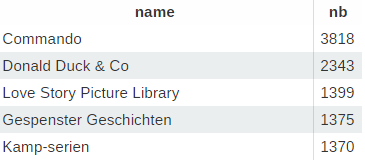
**)** st **ON** issue**.**id **=** st**.**iid

**GROUP** **BY** series\_id

**)** **as** temp **ON** se**.**id **=** temp**.**series\_id

**ORDER** **BY** temp**.**nb **DESC**

**48’913 results (screenshot of firsts)**:



### Query b: Print the names of publishers who have series with all series types.

#### Description of logic:

We do 2 queries separately: one to count the number of series publication type existing, and the second to get the number of different publication type for one publisher. This number thus have to be the total number of types, counted in the first query.

**Indexes**:

An index on series.publisher\_id is very important for this query. If the index doesn’t exist, the query would be very slow.

Execution time: 0.0251 seconds

#### SQL statement

**SELECT** p**.**name **FROM** publisher **as** p **WHERE** **(**

**SELECT** **COUNT(distinct** sp**.**name**)** **FROM** series **as** s

**INNER** **JOIN** series\_publication\_type **as** sp **ON** sp**.**id **=** s**.**publication\_type\_id

**WHERE** s**.**publisher\_id **=** p**.**id

**)** **= (**

**SELECT** **COUNT(distinct** sp**.**name**)**

**FROM** series\_publication\_type **as** sp

**)**

**155 results (screeshot of firsts):**



### Query c : Print the 10 most-reprinted characters from Alan Moore's stories.

#### Description of logic:

The nested query get the id of the heroes for which the writer of the story (script) is Alan Moore. We then simply make a join of this ids with the hero table to get the name of heroes. We also order by descending order by number of reprinting to keep only the 10 most reprinted.

Execution time: 0.017 seconds

#### SQL statement

**SELECT** h**.**name **FROM** **(**

**SELECT** charac**.**hero\_id**,** **COUNT(\*)** **as** nb

**FROM** story\_reprint sr

**INNER** **JOIN** story s **ON** s**.**id **=** sr**.**origin\_id

**INNER** **JOIN** characters charac **ON** charac**.**story\_id **=** s**.**id

**INNER** **JOIN** participate part **ON** part**.**story\_id **=** s**.**id **AND** part**.**role **=** 'script'

**INNER** **JOIN** person pers **ON** pers**.**id **=** part**.**person\_id

**WHERE** pers**.**name **LIKE** '%alan moore%'

**GROUP** **BY** charac**.**hero\_id

**)** **as** interm

**INNER** **JOIN** hero h **ON** h**.**id **=** interm**.**hero\_id

**ORDER** **BY** interm**.**nb **DESC**

**LIMIT** 10

### Query d: Print the writers of nature-related stories that have also done the pencilwork in all their nature-related stories.

#### Description of logic:

We retrieve the artists (persons) with the “script” role who has the same number of nature-related stories in which they are the pencils that the total number of nature-related stories in which they worked as pencils.

Since nature could be in the middle of the title, we search with a like and not with an equality.

#### SQL statement

**SELECT** **distinct** p**.**name **FROM** person p

**INNER** **JOIN** participate part **ON** part**.**person\_id **=** p**.**id

**WHERE** part**.**role **=** 'script' **AND** **(**

**SELECT** **COUNT(**pa\_penc**.**person\_id**)** **FROM** story s**,** participate pa\_penc

**WHERE** part**.**story\_id **=** s**.**id **AND** pa\_penc**.**person\_id **=** part**.**person\_id **AND**

pa\_penc**.**story\_id **=** s**.**id **AND** pa\_penc**.**role **=** 'pencil' **AND** s**.**title **LIKE** '%nature%'

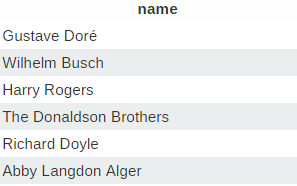
**)** **=** **(**

**SELECT** **COUNT(**pa\_penc**.**person\_id**)** **FROM** story s

**INNER** **JOIN** participate pa\_penc **ON** s**.**id **=** pa\_penc**.**story\_id

**WHERE** pa\_penc**.**role **=** 'pencil' **AND** s**.**title **LIKE** '%nature%'

**)**



### Query e: For each of the top-10 publishers in terms of published series, print the 3 most popular languages of their series.

#### Description of logic:

First we get the ids of the 10 publishers having the higher number of published series along with the number of these publisher series. This is done in the inner query. As always, we group by id, order by the number of series per publisher in descending order and keep only the first 10 results.

Then, we can get the series for these publisher with a join of the ids with the series, in order to retrieve the language of these series.

Execution time : 0.292 seconds

#### SQL statement

**SELECT** l**.**name**,** **count(\*)** **as** nb **FROM** series se

**INNER** **JOIN** **(**

**SELECT** p**.**id**,** **COUNT(**p**.**id**)** **as** num **FROM** series s

**INNER** **JOIN** publisher p **ON** s**.**publisher\_id **=** p**.**id

**GROUP** **BY** p**.**id **DESC**

**ORDER** **BY** **COUNT(**p**.**id**)** **DESC**

**LIMIT** 10

**)** **as** temp **ON** temp**.**id **=** se**.**publisher\_id

**INNER** **JOIN** **language** l **ON** l**.**id **=** se**.**language\_id

**GROUP** **BY** l**.**id

**ORDER** **BY** nb **DESC**

**LIMIT** 3

3 results:



### Query f: Print the languages that have more than 10000 original stories published in magazines, along with the number of those stories.

#### Description of logic:

The query select the languages corresponding to the series of type magazine who was not reprinted (original stories) along with the number of those stories, then filter this result (with HAVING) to keep only the languages for which the number of associated stories is > 10000. See analyzed queries below for more informations.

Execution time: 6.939 seconds

#### SQL statement

**SELECT** l**.**name**,** **COUNT(\*)** **as** nb **FROM** **language** l

**INNER** **JOIN** series se **ON** se**.**language\_id **=** l**.**id

**INNER** **JOIN** issue i **ON** i**.**series\_id **=** se**.**id

**INNER** **JOIN** story s **ON** s**.**issue\_id **=** i**.**id

**WHERE**

se**.**publication\_type\_id **=** **(SELECT** id **FROM** series\_publication\_type **WHERE** name **=** 'magazine'**)** **AND**

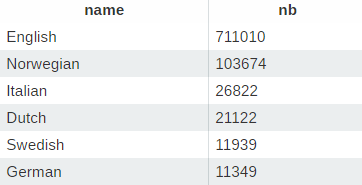
**(SELECT** **COUNT(\*)** **FROM** story\_reprint sr **WHERE** sr**.**target\_id **=** s**.**id**)** **=** 0

**GROUP** **BY** l**.**name

**HAVING** nb **>** 10000

### **ORDER** **BY** nb **DESC**

### 6 results:



### Query g: Print all story types that have not been published as a part of Italian magazine series.

#### Description of logic:

We simply join the information of the 3 table story, issue and series together and thus apply the filter on our data. This request was not particularly hard to write, but take me a long time. Please see the analysis of this query in the next section below, to understand why it took me a long time, to see two versions of this query and more information about it.

Execution time: 0.0551 seconds

#### SQL statement

**SELECT** **distinct** st**.**name

**FROM** story\_type st

**INNER** **JOIN** story s **ON** s**.**type\_id **=** st**.**id

**INNER** **JOIN** issue i **ON** i**.**id **=** s**.**issue\_id

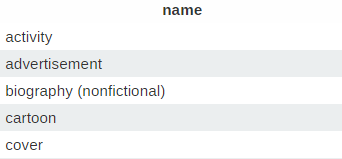
**INNER** **JOIN** series se **ON** se**.**id **=** i**.**series\_id

**WHERE**

se**.**country\_id **!=** **(SELECT** id **FROM** **language** **WHERE** code **=** 'it'**)** **AND**

se**.**publication\_type\_id **=** **(SELECT** id **FROM** series\_publication\_type **WHERE** name**=**'magazine'**)**

**23 results (screenshot of first records)**:



### Query h: Print the writers of cartoon stories who have worked as writers for more than one indicia publisher.

#### Description of logic:

We simply join all the necessary tables to be able to keep only the scripts who participated in the stories of type “cartoon” and as we have the number of record per story who is the number of indicia publisher associated to this story, we can group by writers and take only those who has a counter > 1.

#### SQL statement

**SELECT** p**.**name **FROM** person p

**INNER** **JOIN** participate part **ON** part**.**person\_id **=** p**.**id

**INNER** **JOIN** story s **ON** s**.**id **=** part**.**story\_id

**INNER** **JOIN** issue i **ON** i**.**id **=** s**.**issue\_id

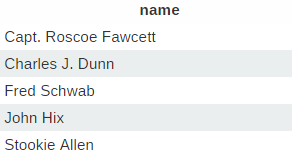
**INNER** **JOIN** indicia\_publisher ip **ON** ip**.**id **=** i**.**indicia\_publisher\_id

**WHERE** s**.**type\_id **=** **(SELECT** id **FROM** story\_type **WHERE** name **=** 'cartoon'**)** **AND** part**.**role **=** 'script'

**GROUP** **BY** p**.**name

**HAVING** **COUNT(\*)** **>** 1

**Screenshot of the first results:**



### Query i: Print the 10 brand groups with the highest number of indicia publishers.

#### Description of logic:

Very obvious, we take all the brand groups associated to the indicia publishers, we group the results by brand name to have the counter of record for each brand name, and finally sort by this counter in descending order to keep only the 10 highest numbers.

Execution time: 0.0455 seconds

#### SQL statement

**SELECT** **distinct** b**.**name**,** **COUNT(\*)** **as** nb **FROM** indicia\_publisher i

**INNER** **JOIN** brand\_group b **ON** b**.**publisher\_id **=** i**.**publisher\_id

**GROUP** **BY** b**.**name

**ORDER** **BY** nb **DESC**

**LIMIT** 10

**10 results (screenshot of the firsts):**



### Query j: Print the average series length (in terms of years) per indicia publisher.

#### Description of logic:

Since the data type of the fields year\_ended and year\_began are “YEAR” who is an unsigned integer, and that we can’t subtract them being unsigned, we cast them as signed before to do the subtraction. Otherwise, we just take care to keep only the records who has consistent values for year began and ended, that is non zero values and when began is < than ended (just to be sure our final value is not polluted if ended is reversed with began or inversely).

Execution time: 1.804 seconds

#### SQL statement

**SELECT** name**,** **AVG(**diff**)** **as** average **FROM** **(**

**SELECT** ip**.**name**,** **(CAST(**s**.**year\_ended **AS** SIGNED**)-CAST(**s**.**year\_began **AS** SIGNED**))** **as** diff

**FROM** series s

**INNER** **JOIN** indicia\_publisher ip **ON** ip**.**publisher\_id **=** s**.**publisher\_id

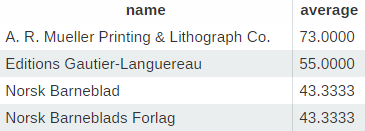
**WHERE** year\_began **<** year\_ended **AND** year\_began **>** 0 **AND** year\_ended **>** 0

**)** **as** res

**GROUP** **BY** name

**ORDER** **by** average **DESC**

**3412 results (screenshot of the firsts):**



### Query k: Print the top 10 indicia publishers that have published the most single-issue series.

#### Description of logic:

<What does the query do and how do I decide to solve it>

Execution time : 2.449 seconds

#### SQL statement

**SELECT** i**.**name**,** **COUNT(\*)** **as** nb

**FROM** series s

**INNER** **JOIN** indicia\_publisher i **ON** s**.**publisher\_id **=** i**.**publisher\_id

**WHERE** s**.**first\_issue\_id **=** s**.**last\_issue\_id

**GROUP** **BY** i**.**name

**ORDER** **BY** nb **DESC**

**LIMIT** 10

**10 results (screenshot of the firsts):**



### Query l: Print the 10 indicia publishers with the highest number of script writers in a single story.

#### Description of logic:

<What does the query do and how do I decide to solve it>

#### SQL statement

**SELECT** ip**.**name**,** **COUNT(**s**.**id**)** **as** nb **FROM** story s

**INNER** **JOIN** participate part **ON** part**.**story\_id **=** s**.**id

**INNER** **JOIN** issue i **ON** i**.**id **=** s**.**issue\_id

**INNER** **JOIN** indicia\_publisher ip **ON** ip**.**id **=** i**.**indicia\_publisher\_id

**WHERE** part**.**role **=** 'script'

**GROUP** **BY** s**.**id**,** ip**.**name

**ORDER** **BY** nb **DESC**

**LIMIT** 10



### Query m: Print all Marvel heroes that appear in Marvel-DC story crossovers.

#### Description of logic:

<What does the query do and how do I decide to solve it>

#### SQL statement

**SELECT** **distinct** h**.**name **FROM** hero h

**INNER** **JOIN** characters ch **ON** ch**.**hero\_id **=** h**.**id

**INNER JOIN** story s1 **ON** ch**.**story\_id **=** s1**.**id

**INNER JOIN** issue i1 **ON** s1**.**issue\_id **=** i1**.**id

**INNER JOIN** indicia\_publisher ip1 **ON** ip1**.**id **=** i1**.**indicia\_publisher\_id

**INNER JOIN** feature fe **ON** fe**.**hero\_id **=** h**.**id

**INNER JOIN** story s2 **ON** fe**.**story\_id **=** s2**.**id

**INNER JOIN** issue i2 **ON** i2**.**id **=** s2**.**issue\_id

**INNER JOIN** indicia\_publisher ip2 **ON** ip2**.**id **=** i2**.**indicia\_publisher\_id

**WHERE** ip1**.**name **LIKE** '%Marvel%' **AND** ip2**.**name **LIKE** '%Marvel%DC%'

### Query n: Print the top 5 series with most issues

#### Description of logic:

<What does the query do and how do I decide to solve it>

Execution time : 0.6867 seconds

#### SQL statement

**SELECT** s**.**name **FROM** series s

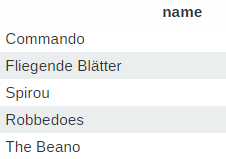
**INNER** **JOIN** issue i **ON** i**.**series\_id **=** s**.**id

**GROUP** **BY** s**.**id

**ORDER** **BY** **COUNT(**s**.**name**)** **DESC**

**LIMIT** 5

**5 results**:



### Query o: Given an issue, print its most reprinted story.

#### Description of logic:

<What does the query do and how do I decide to solve it>

**Indexes**:

An index on story.issue\_id is important

Execution time : 0.007 seconds

#### SQL statement

**SELECT** s**.**title **FROM** story s

**INNER** **JOIN** issue i **ON** s**.**issue\_id **=** i**.**id

**INNER** **JOIN** story\_reprint sr **ON** sr**.**origin\_id **=** s**.**id

**WHERE** i**.**id **=** **?** **AND** s**.**title **IS** **NOT** **NULL**

**GROUP** **BY** s**.**id

**ORDER** **BY** **count(**s**.**id**)** **DESC**

**LIMIT** 1

**Result for i.id = 68 :**



## Query Analysis

### Selected Queries (and why)

Sometimes, the use of an index was not sufficient or sometimes I had to delete an index that I had defined for another query otherwise my query took an infinite execution time. Despite the indexes and improvements, sometimes the execution time for some queries remains a little high.

### Query g: Print all story types that have not been published as a part of Italian magazine series.

**Initial Running time**: 5.071 seconds

**Optimized Running time:** 0.0573 seconds

**Explain the improvement:**

In fact, at the beginning this improved plan was the (very) bad plan, with more than 45 seconds, and the initial plan was the improved one.

The improvement is mainly due to the adding of an index, but in this case, surprisingly the use of the nested queries in the initial plan slow down the query. The fact that the initial plan is worse than the improved one make me confuse because my reasoning for the initial plan is the following: Since the nested queries are executed from innermost to outermost, I first filter the data in the innermost nested query to limit the number of results as early as possible before to join the data between them. This way, the join is made on much less data and the query is greatly improved. But after some days I run again both of these queries and the initial plan presented here who was much better before, become bad compared to the improved one presented below who was very very bad before. The improvement thus come mainly from the index.

**Initial plan**

**SELECT** **distinct** st**.**name **FROM** story s

**LEFT** **JOIN** story\_type st **ON** st**.**id **=** s**.**type\_id

**LEFT** **JOIN** **(**

**SELECT** i**.**id **FROM** issue i

**RIGHT** **JOIN** **(**

**SELECT** se**.**id **FROM** series se **WHERE**

se**.**language\_id **!=** **(SELECT** id **FROM** **language** **WHERE** code **=** 'it'**)** **AND**

se**.**publication\_type\_id **=** **(SELECT** id **FROM** series\_publication\_type **WHERE** name**=**'magazine'**)**

**)** **as** temp **ON** temp**.**id **=** i**.**series\_id

**)** **as** iss **ON** s**.**issue\_id **=** iss**.**id

**Improved plan**

**SELECT** **distinct** st**.**name

**FROM** story\_type st

**LEFT** **JOIN** story s **ON** s**.**type\_id **=** st**.**id

**LEFT** **JOIN** issue i **ON** i**.**id **=** s**.**issue\_id

**LEFT** **JOIN** series se **ON** se**.**id **=** i**.**series\_id

**WHERE**

se**.**country\_id **!=** **(SELECT** id **FROM** **language** **WHERE** code **=** 'it'**)** **AND**

se**.**publication\_type\_id **=** **(SELECT** id **FROM** series\_publication\_type **WHERE** name**=**'magazine'**)**

### Query f

**Initial Running time**: seconds

**Optimized Running time**: 6.9872 seconds

**Explain the improvement**:

The two queries stay approximately the same, the only difference is that the improved plan is written in a more beautiful way. But what has highly influenced the running time here is especially the adding of indexes on all fields who needed one to run faster, that is: series.language\_id, issue.series\_id, story.issue\_id, story\_reprint.target\_id. AND also the deletion in story\_reprint of a multiple index on (id, origin\_id, target\_id).

**Initial plan**

**SELECT** temp**.**name**,** temp**.**nb **FROM** **(**

**SELECT** **distinct** l**.**name**,** **COUNT(\*)** **as** nb

**FROM** **language** l**,**

series se**,**

story s**,**

issue i

**WHERE** l**.**id **=** se**.**language\_id **AND**

se**.**id **=** i**.**series\_id **AND**

i**.**id **=** s**.**issue\_id **AND**

se**.**publication\_type\_id **=** **(SELECT** id **FROM** series\_publication\_type **WHERE** name **=** 'magazine'**)** **AND**

**(SELECT** **COUNT(\*)** **FROM** story\_reprint sr **WHERE** sr**.**target\_id **=** s**.**id**)** **=** 0

**GROUP** **BY** l**.**name

**)** **as** temp

**WHERE** temp**.**nb **>** 10000

**ORDER** **BY** temp**.**nb **DESC**

**Improved plan**

**SELECT** l**.**name**,** **COUNT(\*)** **as** nb **FROM** **language** l

**INNER** **JOIN** series se **ON** se**.**language\_id **=** l**.**id

**INNER** **JOIN** issue i **ON** i**.**series\_id **=** se**.**id

**INNER** **JOIN** story s **ON** s**.**issue\_id **=** i**.**id

**WHERE**

se**.**publication\_type\_id **=** **(SELECT** id **FROM** series\_publication\_type **WHERE** name **=** 'magazine'**)** **AND**

**(SELECT** **COUNT(\*)** **FROM** story\_reprint sr **WHERE** sr**.**target\_id **=** s**.**id**)** **=** 0

**GROUP** **BY** l**.**name

**HAVING** nb **>** 10000

### **ORDER** **BY** nb **DESC**

### Query b

**Initial Running time**: seconds

**Optimized Running time**: 0.02003 seconds

**Explain the improvement**:

In pre-computing the result of the inner query, we can save a little execution time. But it’s a very slight improvement who is almost not visible.

**Initial plan**

**SELECT** p**.**name **FROM** publisher **as** p **WHERE** **(**

**SELECT** **COUNT(distinct** sp**.**name**)** **FROM** series **as** s

**INNER** **JOIN** series\_publication\_type **as** sp **ON** sp**.**id **=** s**.**publication\_type\_id

**WHERE** s**.**publisher\_id **=** p**.**id

**)** **= (**

**SELECT** **COUNT(distinct** sp**.**name**)**

**FROM** series\_publication\_type **as** sp

**)**

**Improved plan**

**SELECT** p**.**name **FROM** publisher **as** p **WHERE** **(**

**SELECT** **COUNT(distinct** sp**.**name**)** **FROM** series **as** s

**INNER** **JOIN** series\_publication\_type **as** sp **ON** sp**.**id **=** s**.**publication\_type\_id

**WHERE** s**.**publisher\_id **=** p**.**id

**)** **= 3**

#### Query I

**Initial Running time**: 0.0771 seconds

**Optimized Running time**: 0.0394 seconds

**Explain the improvement:**

In the initial plan, we join the publisher table to get its id, but in fact we can directly bypass the publisher table, because brand\_group and indicia\_publisher tables both contains a field publisher\_id. This way, we avoid a join and save a little execution time. The time for the initial plan is twice the time for the improved one, but since both stay reasonable and relatively low the difference is not really visible without analysing.

**Initial plan**

**SELECT** **distinct** b**.**name**,** **COUNT(\*)** **as** nb **FROM** indicia\_publisher i

**INNER** **JOIN** publisher p **ON** i**.**publisher\_id **=** p**.**id

**INNER** **JOIN** brand\_group b **ON** b**.**publisher\_id **=** p**.**id

**GROUP** **BY** b**.**name

**ORDER** **BY** nb **DESC**

**LIMIT** 10

**Improved plan**

**SELECT** **distinct** b**.**name**,** **COUNT(\*)** **as** nb **FROM** indicia\_publisher i

**INNER** **JOIN** brand\_group b **ON** b**.**publisher\_id **=** i**.**publisher\_id

**GROUP** **BY** b**.**name

**ORDER** **BY** nb **DESC**

**LIMIT** 10

# Interface

### Design logic Description

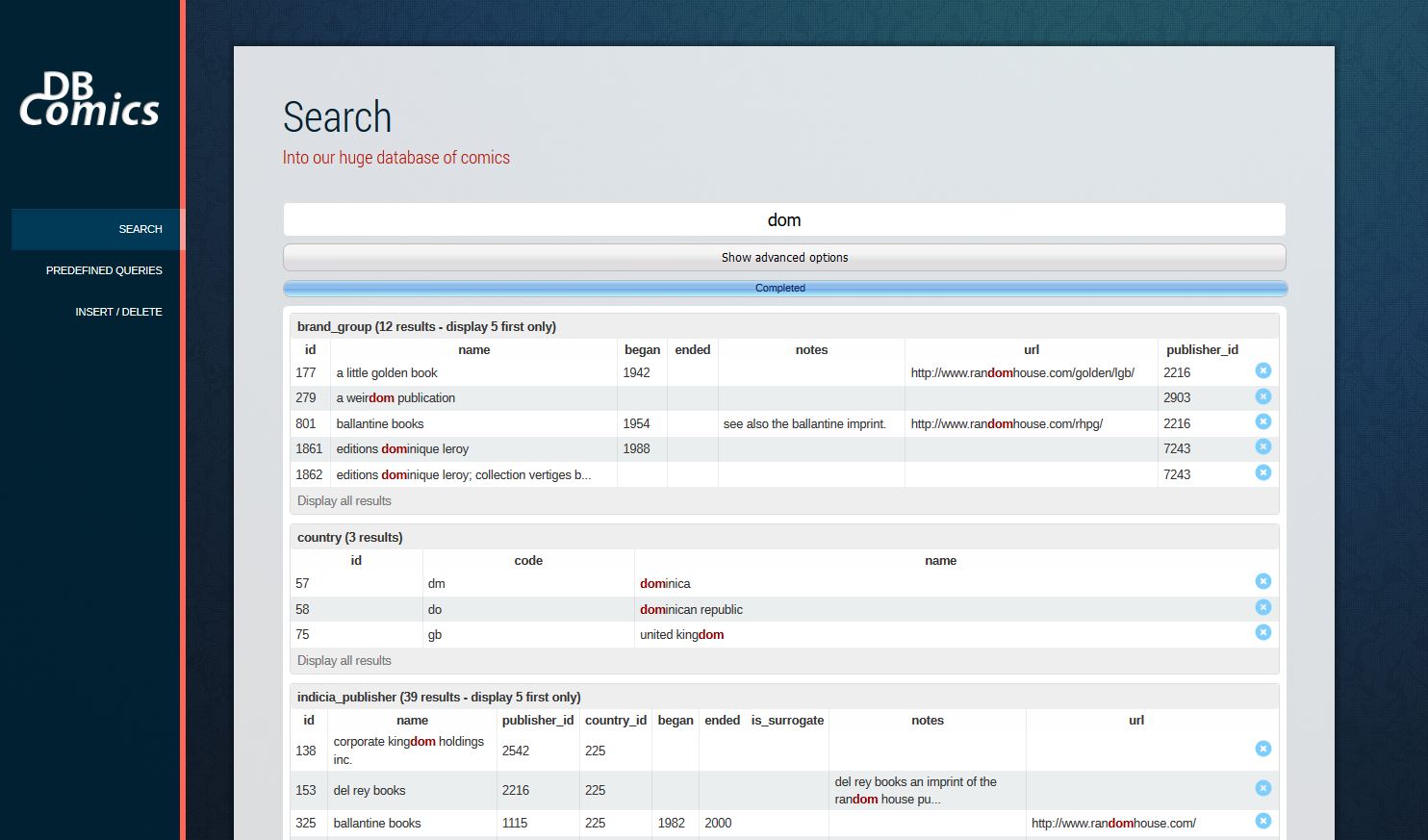
**Deletion**

The deletion were implemented in the search page and the insertion page. In fact the insertion page just display the 5 last records in order to see our new inserted record, but there is a button who allow to search and delete any wanted record. It just redirect us on the search page with only a search on the selected table. We can then do a search by id or whatever we want and to delete the record. There is a confirmation before the deletion of any record, with a popup window provided by jQuery UI.

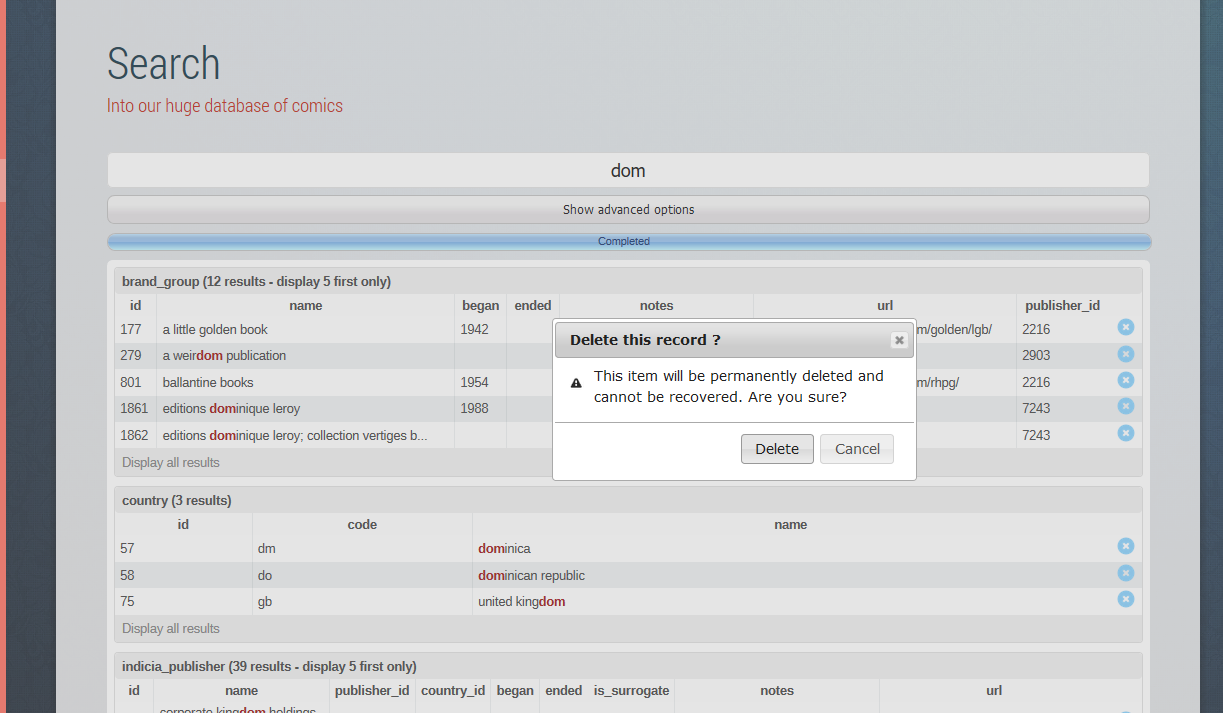
### Screenshots

**Deletion**

To the right, we have a “delete” button for each record.

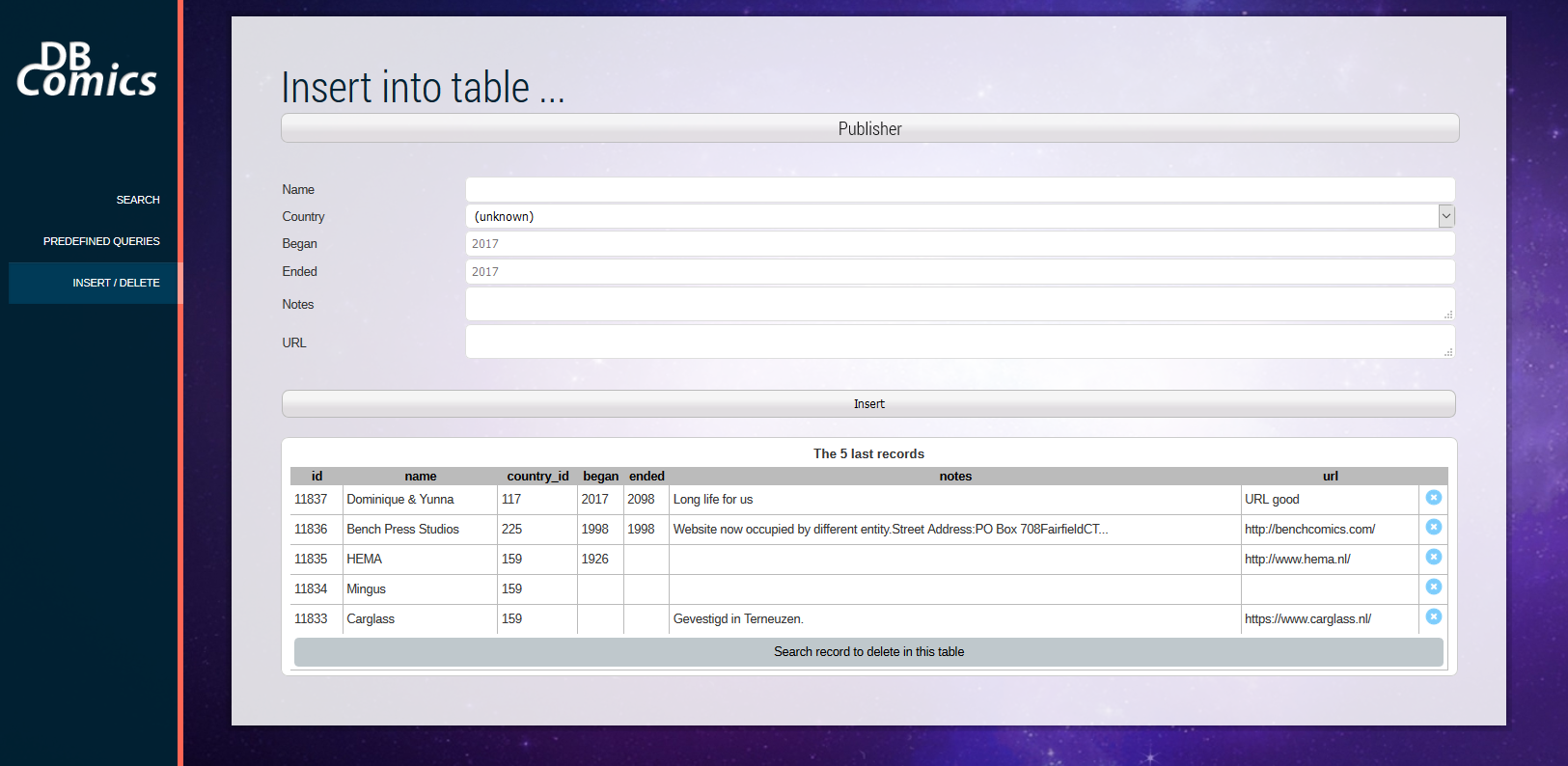


When we click the button for one of these items, a confirmation is asked, to avoid deleting a record unintentionally.

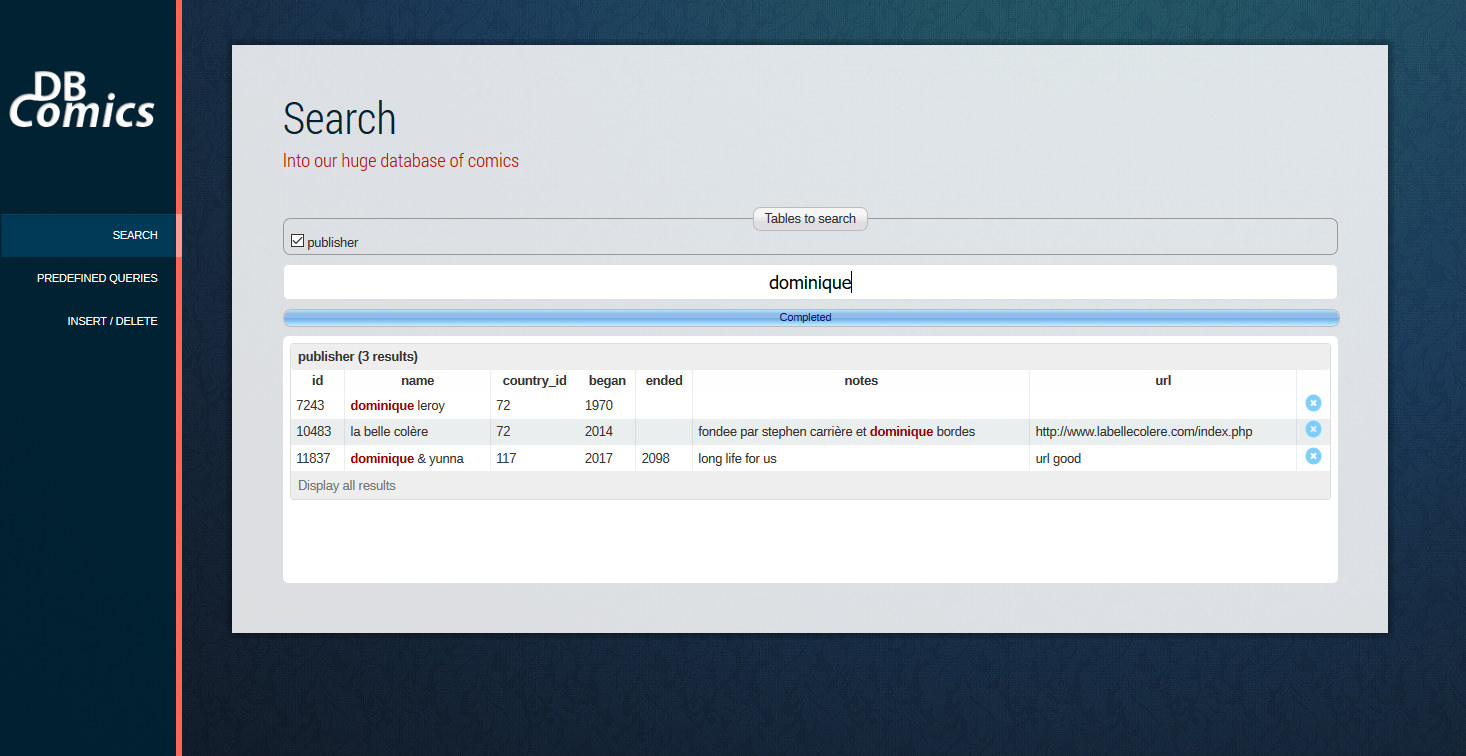


Once the record is deleted, the row is directly deleted without refreshing the page (the deletion is made in ajax as every other actions).

The area with the 5 last records were modified to display a “delete” button also here, the confirmation before deleting too, and a button at the bottom was added, to find and delete any record in the current selected table, from the search engine.

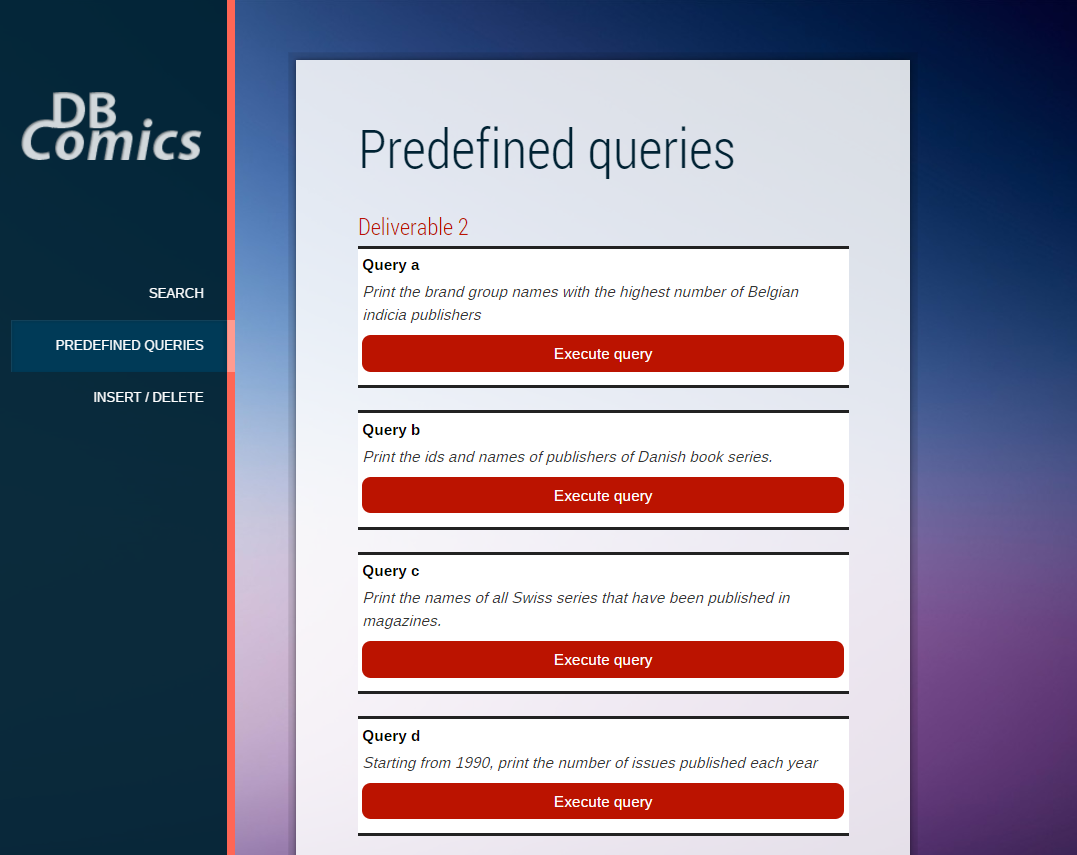


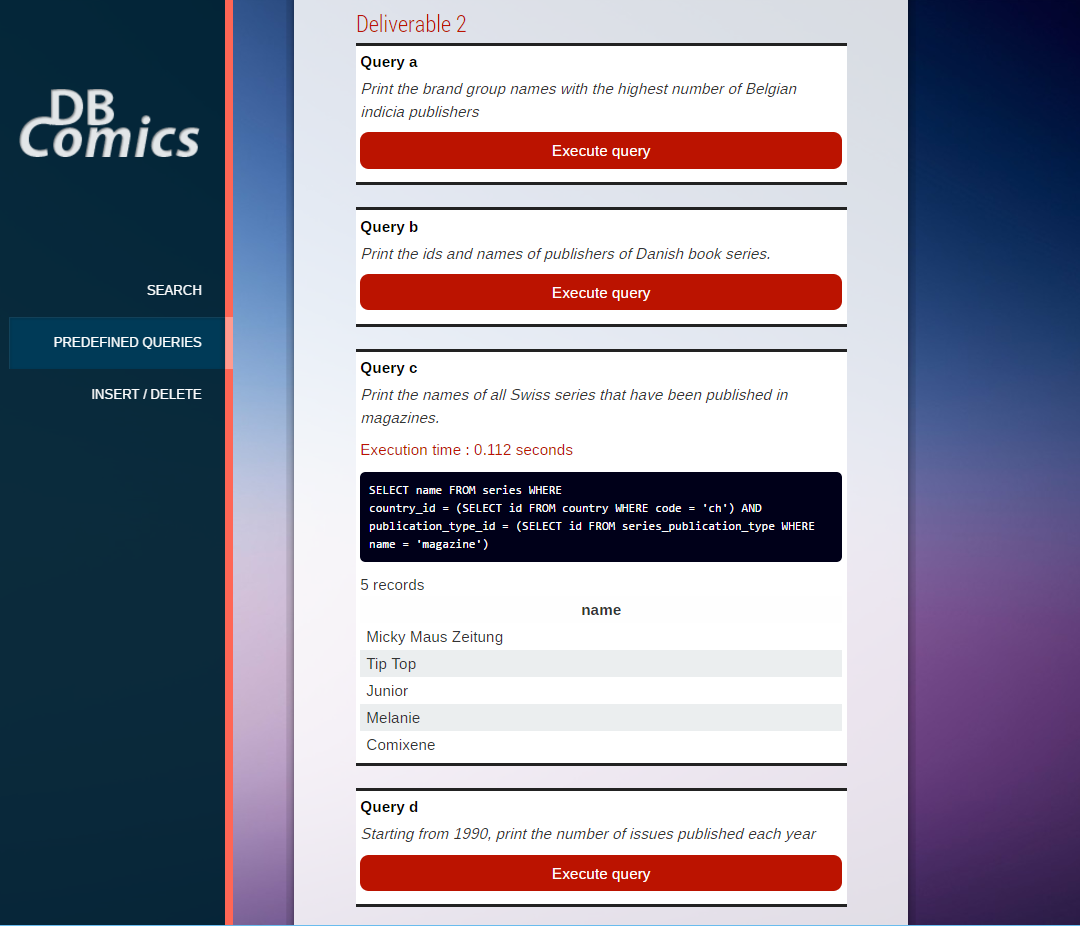
Once we click this button, we thus arrive in the search page, with only the search made on this table :

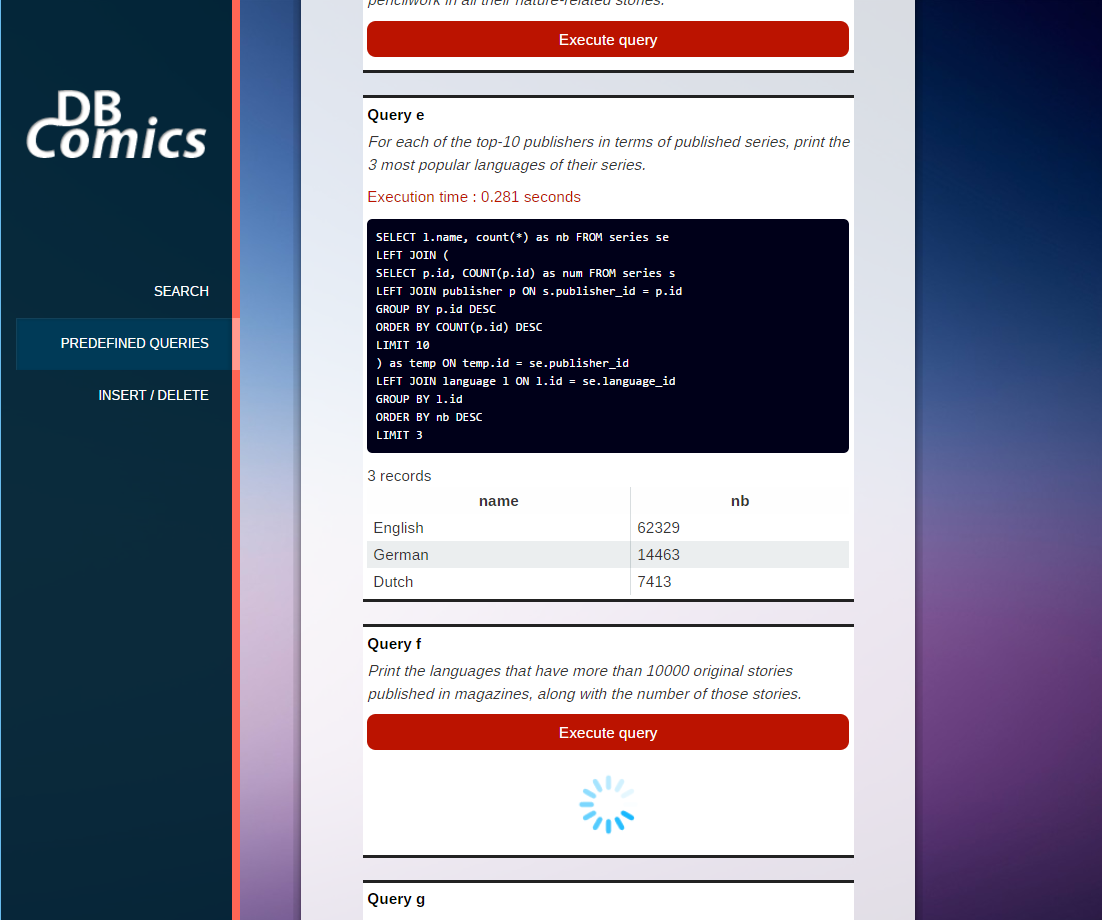


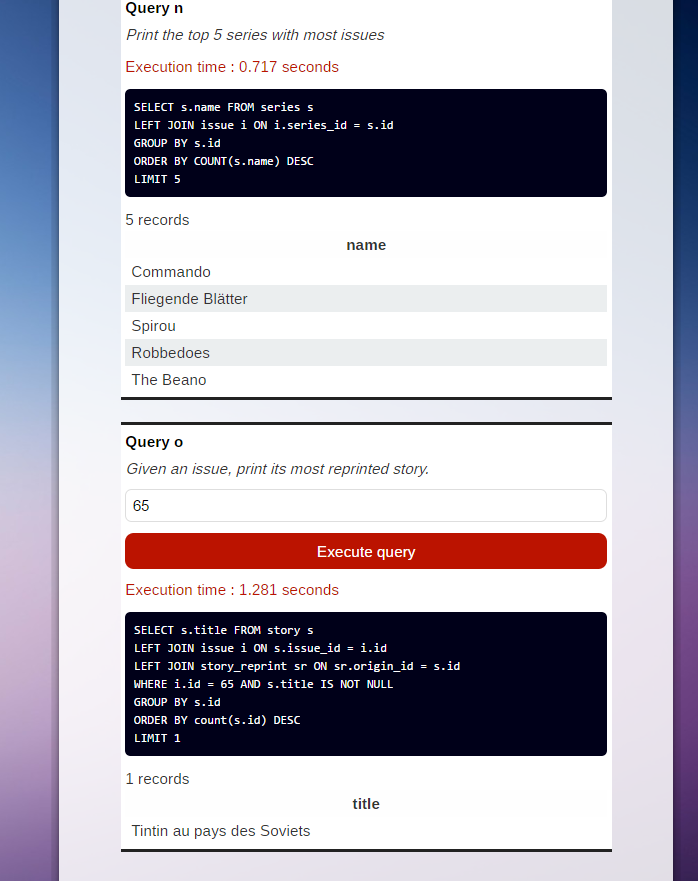
We can then easily delete any record wanted.

**Predefined queries**









# General Comments

<In this section write general comments about your deliverable (comments and work allocation between team members>

* Interface : deletion
* Interface : predefined queries
* Write queries and explain queries
* Improving queries
* Analysing queries