École Polytechnique Fédérale de Lausanne

Databases Project

First and Second deliverable

Authors :
Tristan Overney
Morgan Bruhin
Valentine Arrieta

27 avril 2015

1 First deliverable

1.1 ER model

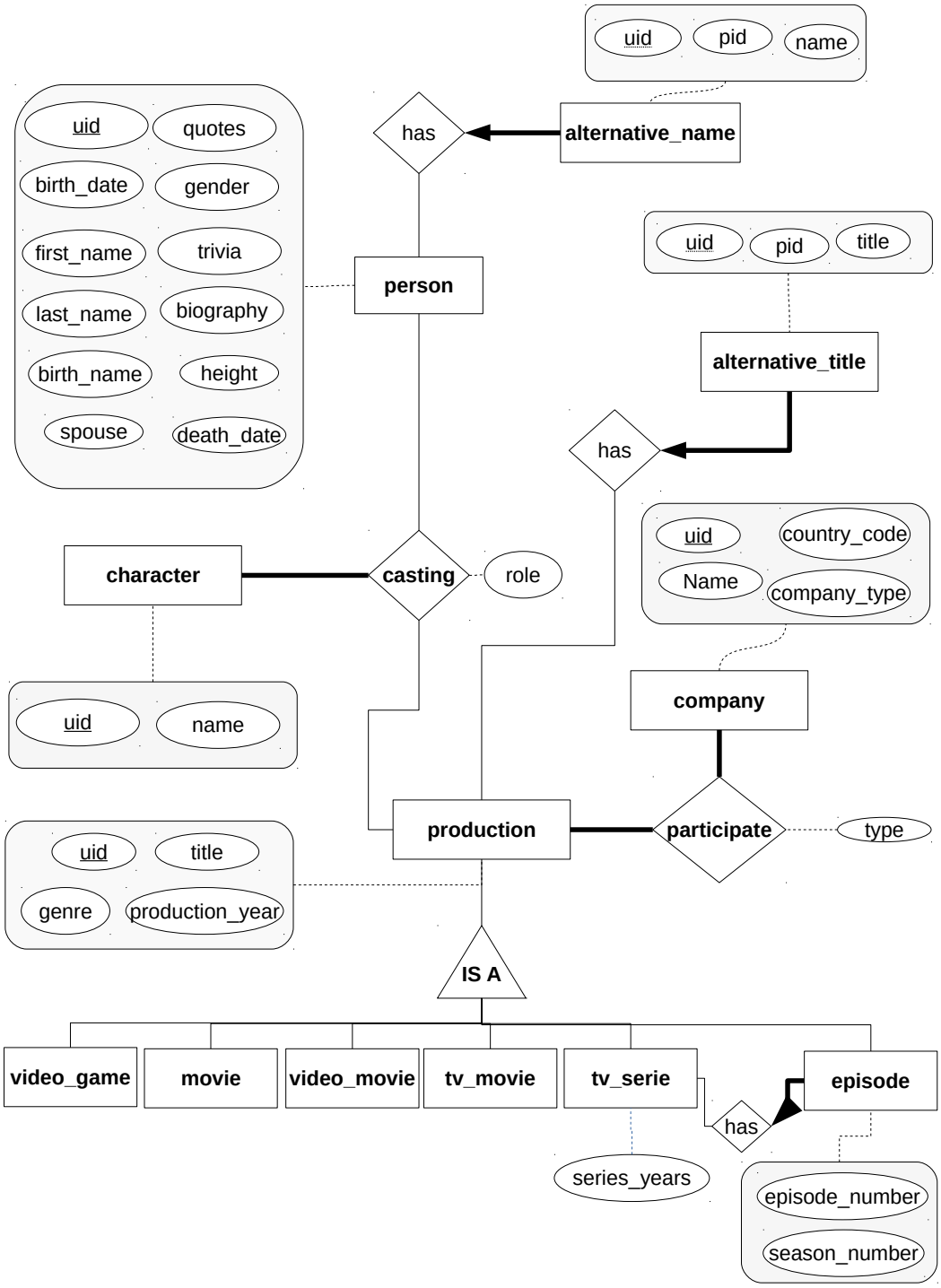

FIGURE 1 – ER model of our DBMS

1.2 Table creation

We will implement the next deliverables using our own server with PostGreSQL. As such the types used in the following CREATE TABLE possibly doesn't work well in an Oracle DB environnement.

```
CREATE TABLE person
   (uid INTEGER NOT NULL,
    first_name CHAR(75) NOT NULL,
    last_name CHAR(75) NOT NULL,
    gender CHAR(1),
    trivia TEXT,
    quotes VARCHAR(4000),
    birth DATE,
    death DATE,
    biography TEXT,
    spouse CHAR(100),
    height DOUBLE PRECISION,
    primary key (uid));

CREATE TABLE alternative_name
   (uid INTEGER NOT NULL,
    pid INTEGER NOT NULL,
    name CHAR(60) NOT NULL,
    primary key (uid),
    foreign key (pid) references person(uid)
    ON DELETE CASCADE);

CREATE TABLE character
   (uid INTEGER NOT NULL,
    name CHAR(60) NOT NULL,
    primary key (uid));

CREATE TYPE CAST_ROLE AS
    ENUM ('actor', 'actress', 'producer', 'writer', 'cinematographer',
    'composer', 'costume␣designer', 'director', 'editor', 'miscellaneous␣crew',
    'production␣designer');

CREATE TABLE production
   (uid INTEGER NOT NULL,
    title CHAR(80) NOT NULL,
    production_year DATE,
    genre CHAR(20),
    primary key (uid));

CREATE TABLE casting
   (cid INTEGER,
    perid INTEGER NOT NULL,
    prodid INTEGER NOT NULL,
    role CAST_ROLE NOT NULL,
    primary key (cid, perid, prodid, role),
    foreign key (cid) references character (uid),
    foreign key (perid) references person (uid),
    foreign key (prodid) references production (uid));
```

```
CREATE TABLE tv_serie
   (series_years CHAR(11),
    primary key (uid)) INHERITS (Production);

CREATE TABLE episode
   (sid INTEGER NOT NULL,
    season SMALLINT NOT NULL,
    episode SMALLINT NOT NULL,
    primary key (uid),
    foreign key (sid) references tv_serie (uid)
    ON DELETE CASCADE) INHERITS (production);

CREATE TABLE tv_movie
    (primary key (uid)) INHERITS (production);

CREATE TABLE movie
    (primary key (uid)) INHERITS (production);

CREATE TABLE video_movie
    (primary key (uid)) INHERITS (production);

CREATE TABLE video_game
    (primary key (uid)) INHERITS (production);

CREATE TYPE COMPANY_TYPE AS ENUM ('distributors', 'production␣company');

CREATE TABLE company
   (uid INTEGER NOT NULL,
    name CHAR(80) NOT NULL,
    country_code CHAR(6),
    primary key (uid));

CREATE TABLE participate
   (pid INTEGER NOT NULL,
    cid INTEGER NOT NULL,
    type COMPANY_TYPE NOT NULL,
    primary key (pid, cid),
    foreign key (pid) references production(uid),
    foreign key (cid) references company(uid));

CREATE TABLE alternative_title
   (uid INTEGER NOT NULL,
    pid INTEGER NOT NULL,
    title CHAR (80) NOT NULL,
    primary key (uid),
    foreign key (pid) references production (uid)
    ON DELETE CASCADE);
```

1.3 Discussion about constraint

A person can have one or more alternative name but an alternative name can describe only one person. We thus have a one-to-many relationship set. Also if the person is deleted from the database, his alternative names (if any) will be too. We thus created a weak entity for alternative name. We applied the same design choice for two similar situations : the production and its alternative title(s) and the tv serie and its episode(s).

A production cast make the connection between the production, the person and the character (if any). We thus made a ternary relationship between the three entity person, character and production. We add the role of the person as an attribute of the relationship set. A character exist only if related to at least one person and one production. The person and production entity don't have this constraint since a production can have only a director that won't have any character.

Companies can be involved into production. They can play roles of different type (for example production, distribution), thus we added an attribute to the relationship "participate" called type instead of having a lot of duplicated entries in the "Company" table. Since that design choice, the data set given has been changed and comforted us in our design choice as the "type" attribute has been moved out of the COMPANY.csv file. A production need to have at least one company involved.

We used enums for both the company type and the cast role because their value options are limited. As such we'll avoid repeating a considerable amount of bytes between the many lines sharing the same value for that field.

And last but not least, instead of having a huge melting pot table called production, we made a "parent table" with all the fields shared between the different kinds of production and then we have a table per production kind which inherit from the production table and where some kind have additional info such as the season number in episode. This is the implementation of an IS A relationship in PostGreSQL. This allow us again to avoid having many entries of a table with some null field that they'd never have a value for.

2 Second deliverable

2.1 First deliverable changes

Since deliverable one we had to do several change on the table creation queries. First of all, the casting table now has its own index as it is indeed not correct to have a nullable element in a primary key ! (its type is serial and its value is set automatically ! Second of all, the PostGreSQL inheritance dream turned sour when we found out that this functionnality works perfectly by itself, as we described it. But you can't have a foreign key pointing to that parent table to reference everything including child table. As such, we were forced to get rid of the inheritance, as we need a foreign key for productions, both in *alternative_title* and *casting* ! Also, we got rid of most $CHAR(some\_int)$ for $TEXT$ because there is often one or two values that are much longer that the others and if you use a $CHAR()$ you'd have to allocate bytes to fit the longest entry for each line even the shorter. With $TEXT$ the allocation is more clever and the number of bytes for a field is variable between lines. This change reduced greatly the size of our database tables. Eventually, we added the field "birth name" to the person table which we forgot to add for the first deliverable. Here are the new table creation instructions as a recap.

```
CREATE TABLE person
   (uid INTEGER NOT NULL,
    first_name TEXT,
    last_name TEXT NOT NULL,
    gender CHAR(1),
    trivia TEXT,
    quotes TEXT,
    birth DATE,
    death DATE,
    biography TEXT,
    spouse TEXT,
    height DOUBLE PRECISION,
    birth_name TEXT,
    primary key (uid));

CREATE TABLE alternative_name
   (uid INTEGER NOT NULL,
    pid INTEGER NOT NULL,
    name TEXT NOT NULL,
    primary key (uid),
    foreign key (pid) references person(uid)
    ON DELETE CASCADE);

CREATE TABLE character
   (uid INTEGER NOT NULL,
    name TEXT NOT NULL,
    primary key (uid));

CREATE TYPE CAST_ROLE AS
    ENUM ('actor', 'actress', 'producer', 'writer', 'cinematographer',
    'composer', 'costume␣designer', 'director', 'editor', 'miscellaneous␣crew',
    'production␣designer');

CREATE TYPE PRODUCTION_KIND AS
    ENUM ('tv␣series', 'episode', 'movie', 'video␣movie',
    'tv␣movie', 'video␣game');
```

```
CREATE TABLE production
   (uid INTEGER NOT NULL,
    title TEXT NOT NULL,
    production_year INT,
    kind PRODUCTION_KIND,
    genre CHAR(20),
    primary key (uid));

CREATE TABLE casting
   (uid SERIAL,
    cid INTEGER,
    perid INTEGER NOT NULL,
    prodid INTEGER NOT NULL,
    role CAST_ROLE NOT NULL,
    primary key (uid),
    foreign key (cid) references character (uid),
    foreign key (perid) references person (uid),
    foreign key (prodid) references production (uid));


CREATE TABLE tv_series
   (uid INTEGER NOT NULL,
    series_years CHAR(10),
    primary key (uid),
    foreign key (uid) references production (uid));

CREATE TABLE episode
   (uid INTEGER NOT NULL,
    sid INTEGER NOT NULL,
    season SMALLINT,
    episode INTEGER,
    primary key (uid),
    foreign key (uid) references production (uid),
    foreign key (sid) references tv_serie (uid)
    ON DELETE CASCADE);

CREATE TYPE COMPANY_TYPE AS ENUM ('distributors', 'production␣companies');

CREATE TABLE company
   (uid INTEGER NOT NULL,
    country_code CHAR(6),
    name TEXT NOT NULL,
    primary key (uid));

CREATE TABLE participate
   (uid INTEGER NOT NULL,
    pid INTEGER NOT NULL,
    cid INTEGER NOT NULL,
    type COMPANY_TYPE NOT NULL,
    primary key (uid),
    foreign key (pid) references production(uid),
    foreign key (cid) references company(uid));

CREATE TABLE alternative_title
```

```
(uid INTEGER NOT NULL,
 pid INTEGER NOT NULL,
 title TEXT NOT NULL,
 primary key (uid),
 foreign key (pid) references production (uid)
 ON DELETE CASCADE);
```

2.2 SQL queries

Here are the specific queries required in the 2nd deliverable specifications.
A) Number of movies per production year

```
SELECT production_year, COUNT(*) FROM production
WHERE (kind = 'movie'
    OR kind = 'tv movie'
    OR kind = 'video movie')
    AND production_year IS NOT NULL
GROUP BY production_year
ORDER BY production_year ASC;
```

B) Top 10 countries with the most production companies

```
SELECT c.country_code, COUNT(DISTINCT c.uid) AS count
FROM company c
LEFT JOIN participate par ON c.uid = par.cid
WHERE par.type = 'production companies'
    AND c.country_code IS NOT NULL
GROUP BY c.country_code
ORDER BY count DESC LIMIT 10;
```

C) Compute the min, max and average career duration

```
SELECT avg(duration) AS avg_duration, MAX(duration) AS max_duration,
MIN(duration) AS min_duration
FROM (
    SELECT (1+(MAX(prod.production_year)-MIN(prod.production_year)))
    AS duration
    FROM casting c, production prod
    WHERE c.prodid = prod.uid
    GROUP BY c.perid
    ) AS career_duration;
```

D) Compute the min, max and average number of actors in a production

```
SELECT avg(numb) AS avg_nb_act, MAX(numb) AS max_nb_act,
MIN(numb) AS min_nb_act
FROM (  SELECT count(*) AS numb
        FROM casting
        WHERE role = 'actor'
        GROUP BY prodid
) AS number;
```

E) Min, Max, Avg height of female persons

```
SELECT MIN(height) AS min_height, MAX(height) AS max_height,
AVG(height) AS avg_height
FROM person
WHERE gender = 'f';
```

F) List all pairs of persons and movies where the person has both directed the movie and acted in the movie. Do not include tv and videos movies

```
SELECT p2.uid, p1.uid, person.first_name, person.last_name,
prod.title, p1.role, p2.role
FROM casting p1
LEFT JOIN casting p2 ON p1.prodid = p2.prodid AND p1.perid = p2.perid
LEFT JOIN person ON p1.perid = person.uid
LEFT JOIN production prod ON p1.prodid = prod.uid
WHERE p1.role='director' AND p2.role='actor';
```

G) List the three most popular character names

```
SELECT name, COUNT(*) AS amount
FROM character
GROUP BY name
ORDER BY amount DESC LIMIT 3;
```

2.3 Index Creation

In order to increase our queries execution speed (which are relatively long as the server hosting our PostGreSQL database is only running a intel Atom D2500!) we created indexes on all foreign key fields in our tables. And, in addition to that we also created an index on *role* in casting, *last_name* and *gender* in person, *production_year*, *genre* and *kind* in production and *country_code* in company. Here is one of the create index instructions :

```
CREATE INDEX CONCURRENTLY casting_perid_index ON casting(perid);
```

2.4 Graphical User Interface Design

For the method to apply on a graphical user interface (GUI) allowing to query our movie database, we choose an opensource internet framework written in python, Django [1] (preset queries where implemented with a pyscopg2 query but the keyword search is implemented with django Models system, but we're using the raw() function which allows us to write SQL normally for the request!). The graphical design was focused on giving an open space to the user rather than a high-density function in a unique window.

1. Django Software Fondation, https ://www.djangoproject.com

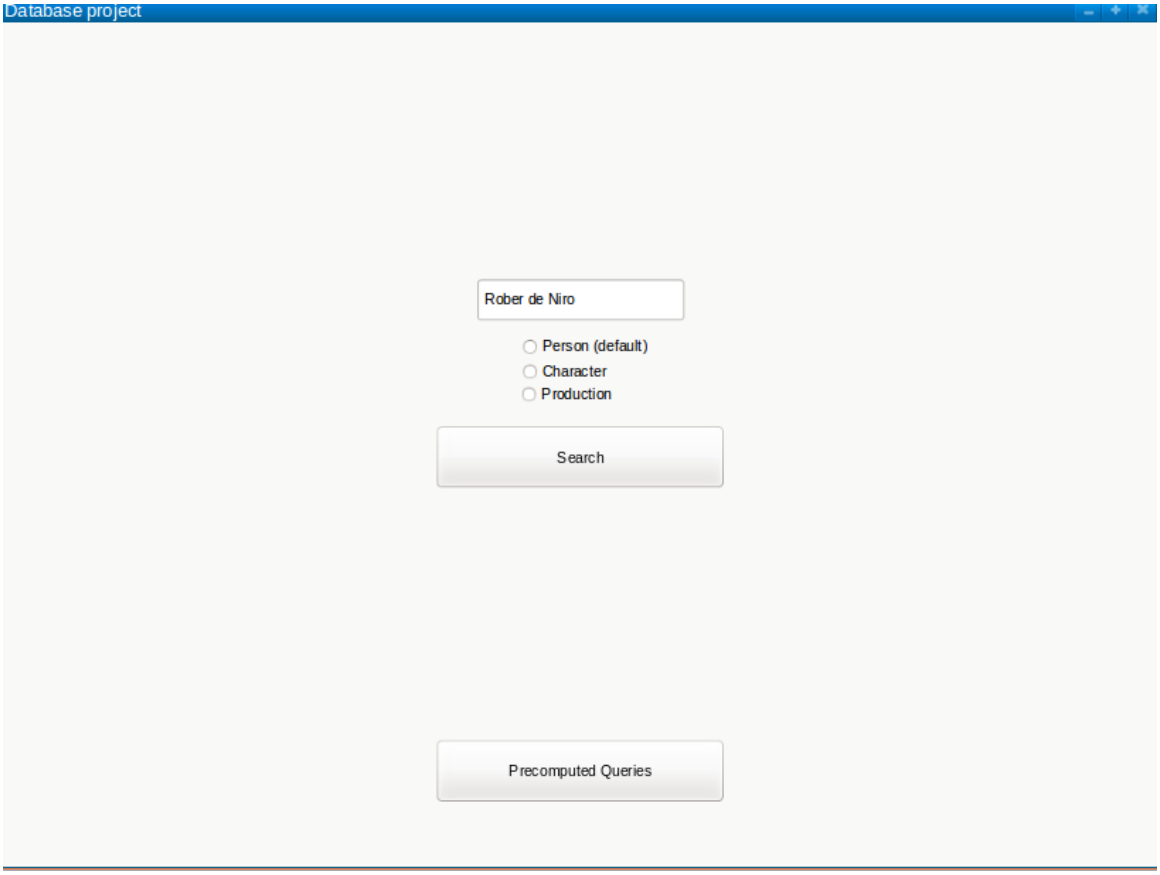

FIGURE 2 – Main page v0.12

Based on that, we design a main window with only the general search, meaning a text entry, a validation button and three radioboxes to specified what kind of search the user intend to do : by person, by movie or by casting. Once the user made a search, the results are display in a form of a list summarizing the informations retrieved from the database. The result window keeps the three general search components in case the user would want to adapt his search. We also add the possibility to recieve further informations on a part of the result by cliquing on it, this will in fact run a search query base on the type of the object of the user interest.

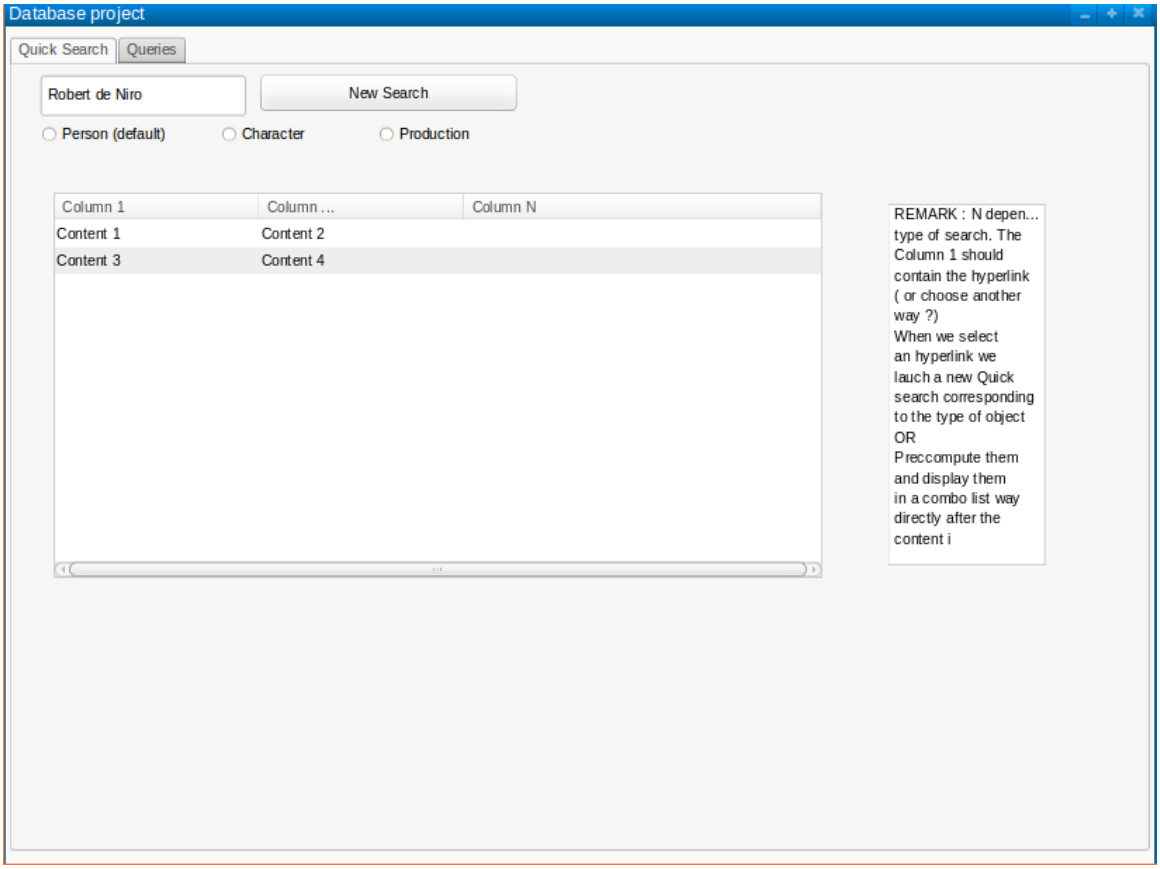

FIGURE 3 – Result of the keyword search v0.12

Then, to display the deliverable queries and their results, we used a menu list (comparable to an onglet system) to switch from the general search (default window) to the list of the deliverable queries (at the end 21 queries). Also here clicking on one component of this list will trigger the associated SQL query and display the results in a new window to easily adapt the layout of the results. Indeed, the kind of results may diverge greatly from a unique row to several hundreds of them.

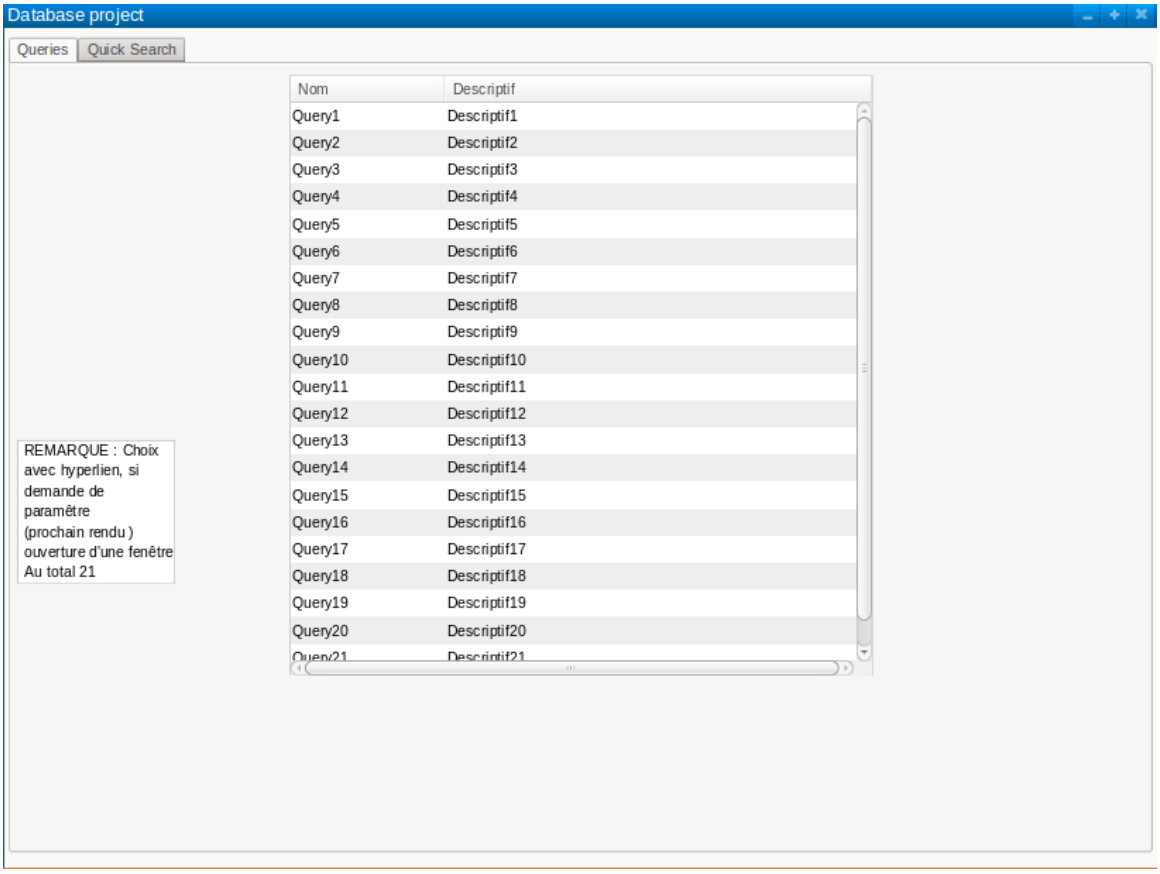

FIGURE 4 – List of the preset queries v0.12