

# Databases Project – Spring 2018

---

Team No: 02

Names: Zeineb SAHNOUN – Nourchene BEN ROMDHANE – Mehdi HACHICH

## Contents

Contents.....	1
Deliverable 1 .....	3
Assumptions .....	3
Entity Relationship Schema .....	3
Schema.....	
Description.....	
Relational Schema .....	4
ER schema to Relational schema .....	
DDL.....	
General Comments .....	
Deliverable 2 .....	5
Assumptions .....	
Data Loading .....	8
Query Implementation .....	12
Query a: .....	
Description of logic: .....	
SQL statement.....	
Interface.....	15
Design logic Description.....	
Screenshots.....	
General Comments .....	
Deliverable 3 .....	17
Assumptions .....	
Query Implementation .....	17

Query a: .....

Description of logic: .....

SQL statement.....

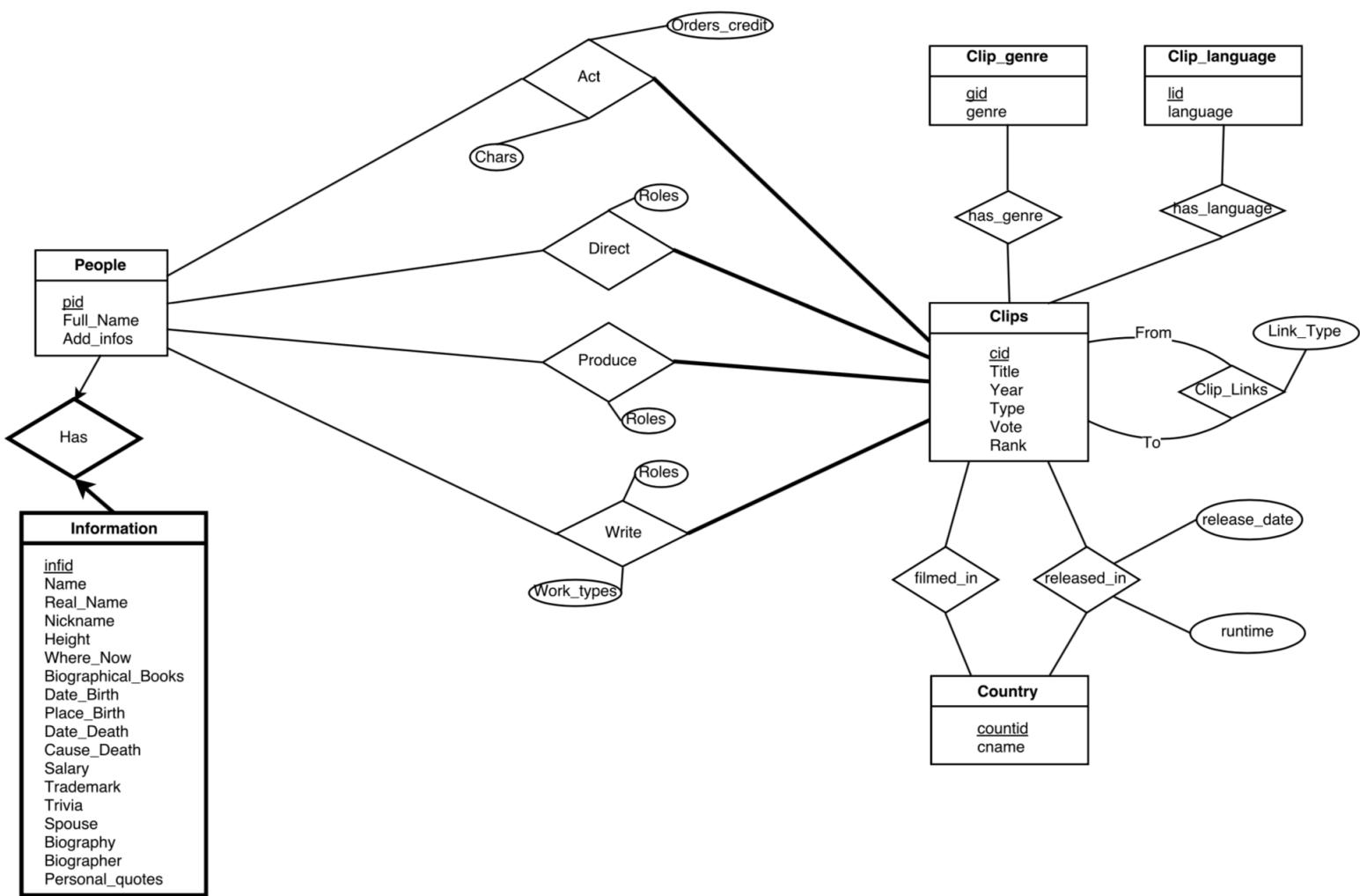
Query Analysis .....	29
Selected Queries (and why) .....	30
Query 1 .....	31
Query 2 .....	31
Query 3 .....	36
Interface.....	36
Design logic Description.....	38
Screenshots.....	38
General Comments .....	41

## Deliverable 1

### Assumptions

- We assumed that an actor/director/producer/writer may not participate in a clip.
- A clip has at least one actor/director/producer/writer that participates in it.
- After examining the data, we found that a person has 0 or 1 biography.

### Entity Relationship Schema



## Description of the ER diagram:

- We choose to create an entity People which gathers the actors, directors, producers and writers, since they have the same attributes. We can differentiate them by their relation with the clip.
- We choose to represent the biography of each person by a weak entity Information because without a person, a biography does not exist.
- We choose to represent the clips by an entity named Clips. Since a clip can be released in many countries, we decided to create an entity named Country. We used the same reasoning for entities Clip\_genre and Clip\_language (A clip can have many genres and languages).
- We choose to create an integer id for each entity, since strings are not efficient.

```

CREATE TABLE People(
    pid INTEGER,
    full_name CHAR(32),
    add_infos CHAR(32),
    PRIMARY KEY (pid))

CREATE TABLE Informations(
    inf_id INTEGER,
    name CHAR(32),
    real_name CHAR(32),
    nickname CHAR(32),
    height CHAR(10),
    where_now CHAR(64),
    biographical_books CHAR(128),
    date_birth CHAR(32),
    place_birth CHAR(32),
    date_death CHAR(32),
    cause_death CHAR(32),
    salary FLOAT,
    trademark CHAR(128),
    trivia CHAR(128),
    spouse CHAR(32),
    biography
    biographer CHAR(32)
    personal_quotes CHAR(128)
    pid CHAR(10) NOT NULL,
    PRIMARY KEY (inf_id, pid),
    FOREIGN KEY (pid) REFERENCES People (pid), ON DELETE CASCADE)

CREATE TABLE Act(
    pid INTEGER,
    cid INTEGER,
    orders_credits CHAR(10)
    chars CHAR(64),
    PRIMARY KEY (pid, cid),
    FOREIGN KEY (pid) REFERENCES People (pid)
    FOREIGN KEY (cid) REFERENCES Clips (cid))

CREATE TABLE Direct(
    pid INTEGER,
    cid INTEGER,
    roles CHAR(64),
    PRIMARY KEY (pid, cid),
    FOREIGN KEY (pid) REFERENCES People (pid)
    FOREIGN KEY (cid) REFERENCES Clips (cid))

CREATE TABLE Produce(
    pid INTEGER,
    cid INTEGER,
    roles CHAR(64),
    PRIMARY KEY (pid, cid),
    FOREIGN KEY (pid) REFERENCES People (pid)
    FOREIGN KEY (cid) REFERENCES Clips (cid))

```

```

CREATE TABLE Write(
    pid INTEGER,
    cid INTEGER,
    roles CHAR(64),
    work_types CHAR(32),
    PRIMARY KEY (pid, cid),
    FOREIGN KEY (pid) REFERENCES People (pid)
    FOREIGN KEY (cid) REFERENCES Clips (cid))

CREATE TABLE Clips(
    cid INTEGER NOT NULL,
    clipid_to INTEGER NOT NULL,
    -- needed by the Clip_links relationship
    title CHAR(64),
    vote INTEGER,
    rank FLOAT,
    year INTEGER,
    link_type CHAR(32),
    PRIMARY KEY (cid))
    FOREIGN KEY (clipid_to) REFERENCES Clips (cid)

CREATE TABLE Country(
    countid INTEGER,
    cname CHAR(32),
    PRIMARY KEY(countid))

CREATE TABLE Released(
    cid INTEGER,
    countid INTEGER,
    release_date CHAR(32),
    release_country CHAR(32),
    PRIMARY KEY (cid,countid),
    FOREIGN KEY (cid) REFERENCES Clips(cid),
    FOREIGN KEY (countid) REFERENCES Country(countid))

CREATE TABLE Clip_genre(
    gid INTEGER,
    gname CHAR(20),
    PRIMARY KEY (gid))

CREATE TABLE Has_genre(
    cid INTEGER,
    gid INTEGER,
    PRIMARY KEY (cid,gid),
    FOREIGN KEY (cid) REFERENCES Clips(cid),
    FOREIGN KEY (gid) REFERENCES Clip_genre(gid))

CREATE TABLE Clip_language(
    lid INTEGER,
    langage CHAR(20),
    PRIMARY KEY (lid))

CREATE TABLE Has_language(
    cid INTEGER,
    lid INTEGER,
    PRIMARY KEY (cid,lid),
    FOREIGN KEY (cid) REFERENCES Clips(cid),
    FOREIGN KEY (lid) REFERENCES Clip_language(lid))

```

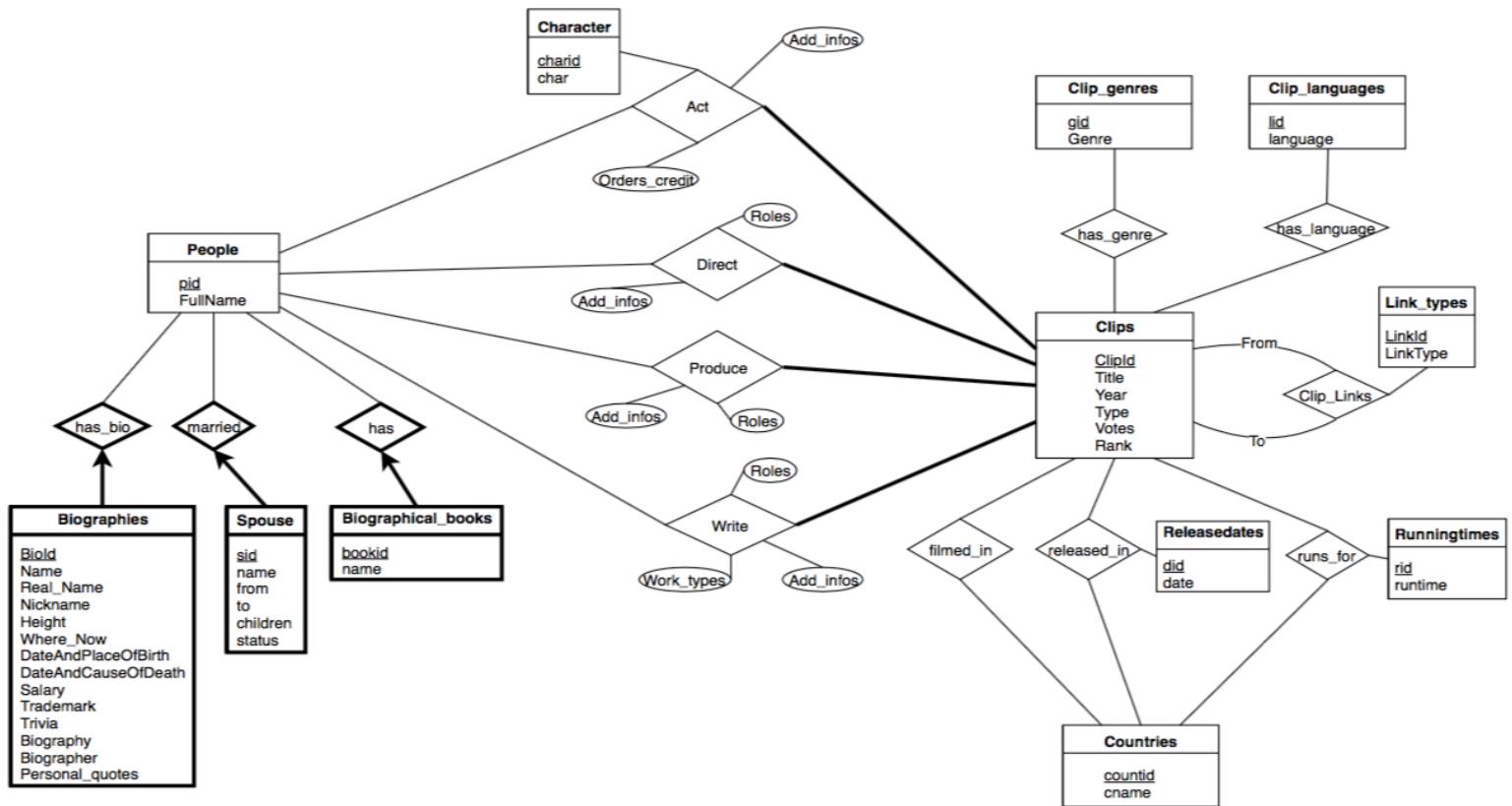
```
CREATE TABLE Filmed_in(  
    countid INTEGER,  
    cid INTEGER,  
    PRIMARY KEY (cid,countid),  
    FOREIGN KEY (cid) REFERENCES Clips(cid),  
    FOREIGN KEY (countid) REFERENCES Country(countid))
```

## Deliverable 2

### *Changes in the ER Diagram*

- Following the feedback we got for deliverable 1, we changed the runs\_for and released\_in relationships as ternary relationships as suggested. We also made ACT a ternary relationship and added the Character entity. We also created new weak entities for spouse and BiographicalBooks.
- When loading the data, we noticed there was a large number of clip\_links which were between the same two clips but with different link\_types, so we decided to make clip\_links a ternary relationship and create a new entity called link\_types (see ER Diagram)
- When loading the data, we noticed that Add\_infos field was separated to many values just like the orders\_credit and Roles, so we made it an attribute of the relation act instead of an attribute of People.

## New the ER Diagram



## New DDL Commands

```
CREATE TABLE people(
    pid INTEGER,
    FullName VARCHAR2(100) NOT NULL,
    PRIMARY KEY (pid));
```

```
CREATE TABLE act(
    pid INTEGER,
    ClipIds INTEGER,
    charid INTEGER,
    OrdersCredit INTEGER,
    AddInfos VARCHAR2(100),
    PRIMARY KEY (pid, ClipIds, charid),
    FOREIGN KEY (pid) REFERENCES People,
    FOREIGN KEY (ClipIds) REFERENCES Clips,
    FOREIGN KEY (charid) REFERENCES Character);
```

```
CREATE TABLE Direct(
    pid INTEGER,
    ClipIds INTEGER,
    Roles VARCHAR2(90),
    AddInfos VARCHAR2(350),
    PRIMARY KEY (pid, ClipIds),
    FOREIGN KEY (pid) REFERENCES People,
    FOREIGN KEY (ClipIds) REFERENCES Clips );
```

```
CREATE TABLE Produce(
    pid INTEGER,
    ClipIds INTEGER,
    Roles VARCHAR2(100),
    AddInfos VARCHAR2(250),
    PRIMARY KEY (pid, ClipIds),
    FOREIGN KEY (pid) REFERENCES People,
    FOREIGN KEY (ClipIds) REFERENCES Clips);
```

```
CREATE TABLE Write(
    pid INTEGER,
    ClipIds INTEGER,
    Roles VARCHAR2(60),
    AddInfos VARCHAR2(400),
    WorkTypes CHAR(32),
    PRIMARY KEY (pid, ClipIds),
    FOREIGN KEY (pid) REFERENCES People,
    FOREIGN KEY (ClipIds) REFERENCES Clips);
```

```
CREATE TABLE Biographical_books(
    bookid INTEGER,
    name VARCHAR2(400) NOT NULL,
    pid INTEGER NOT NULL,
    PRIMARY KEY (bookid, pid),
    FOREIGN KEY (pid) REFERENCES People (pid)
    ON DELETE CASCADE);
```

```
CREATE TABLE biographies(
    Biold INTEGER,
    pid INTEGER NOT NULL,
    FullName VARCHAR(80),
    RealName VARCHAR(250),
    Nickname VARCHAR(400),
    DateAndPlaceOfBirth VARCHAR(150),
    Height VARCHAR(20),
    biography CLOB,
    biographer VARCHAR(150),
    DateAndCauseOfDeath VARCHAR(250),
    Trivia CLOB,
    PersonalQuotes CLOB,
    Salary CLOB,
    Trademark CLOB,
    WhereAreTheyNow CLOB,
    PRIMARY KEY (Biold, pid),
    FOREIGN KEY (pid) REFERENCES People (pid) ON
    DELETE CASCADE);
```

```
CREATE TABLE married(
    pid INTEGER,
    sid INTEGER,
    name VARCHAR2(150),
    since INTEGER,
    untilt INTEGER,
    children VARCHAR2(80),
    status VARCHAR2(80),
    PRIMARY KEY (pid , sid),
    FOREIGN KEY (pid) REFERENCES People ON DELETE
    CASCADE);
```

```
CREATE TABLE Clips(
    ClipIds INTEGER,
    Title VARCHAR2(400),
    Year INTEGER,
    Type VARCHAR2(20),
    Votes INTEGER,
    Rank FLOAT,
    PRIMARY KEY (ClipIds));
```

```
CREATE TABLE Link_types(
    LinkId INTEGER,
    LinkType VARCHAR2(30) NOT NULL,
    PRIMARY KEY(LinkId));
```

CH-1015 Lausanne  
 URL: <http://dias.epfl.ch/>

```

CREATE TABLE Clip_Links(
    ClipFrom INTEGER,
    ClipTo INTEGER,
    LinkId INTEGER,
    PRIMARY KEY (ClipFrom , ClipTo, LinkId),
    FOREIGN KEY (ClipFrom) REFERENCES Clips (ClipIds),
    FOREIGN KEY (ClipTo) REFERENCES Clips (ClipIds),
    FOREIGN KEY (LinkId) REFERENCES Link_types
    (LinkId));

CREATE TABLE Character(
    charid INTEGER,
    Chars VARCHAR2(800) NOT NULL,
    PRIMARY KEY (charid));

CREATE TABLE Clip_languages(
    lid INTEGER,
    Langage VARCHAR2(80) NOT NULL,
    PRIMARY KEY (lid));

CREATE TABLE has_language(
    ClipId INTEGER,
    lid INTEGER,
    PRIMARY KEY (ClipId,lid),
    FOREIGN KEY (ClipId) REFERENCES Clips(ClipIds),
    FOREIGN KEY (lid) REFERENCES Clip_languages(lid));

CREATE TABLE countries(
    countid INTEGER,
    CountryName VARCHAR2(50) NOT NULL,
    PRIMARY KEY(countid));

CREATE TABLE filmed_in(
    ClipId INTEGER,
    countid INTEGER,
    PRIMARY KEY (ClipId, countid),
    FOREIGN KEY (ClipId) REFERENCES Clips(ClipIds),
    FOREIGN KEY (countid) REFERENCES
    Countries(countid));

CREATE TABLE released_in(
    ClipId INTEGER,
    countid INTEGER,
    did INTEGER,
    PRIMARY KEY (ClipId, countid, did),
    FOREIGN KEY (ClipId) REFERENCES Clips(ClipIds),
    FOREIGN KEY (countid) REFERENCES
    Countries(countid),
    FOREIGN KEY (did) REFERENCES Releasedates(did));

CREATE TABLE Releasedates(
    did INTEGER,
    Releasedate INTEGER NOT NULL,
    PRIMARY KEY(did));

CREATE TABLE Clip_genre(
    gid INTEGER,
    Genre VARCHAR2(20) NOT NULL,
    PRIMARY KEY (gid));

CREATE TABLE Has_genre(
    ClipId INTEGER,
    gid INTEGER,
    PRIMARY KEY (ClipId,gid),
    FOREIGN KEY (ClipId) REFERENCES Clips(ClipIds),
    FOREIGN KEY (gid) REFERENCES Clip_genre(gid));

CREATE TABLE Runningtimes(
    rid INTEGER,
    runtime INTEGER NOT NULL,
    PRIMARY KEY(rid));

CREATE TABLE Runs_for(
    ClipId INTEGER,
    countid INTEGER,
    rid INTEGER,
    PRIMARY KEY (ClipId, countid, rid),
    FOREIGN KEY (ClipId) REFERENCES Clips(ClipIds),
    FOREIGN KEY (countid) REFERENCES
    Countries(countid),
    FOREIGN KEY (rid) REFERENCES Runningtimes(rid));

```

## Data Loading

- We had to ignore 7 rows from the Biographies.csv file because they had more entries than expected (more commas ,)
- The running\_times and release\_dates tables given to us had NAN countryname for more than half of their entries. Given our ER Diagram, we cannot have NAN in countryname field, after getting advice from the TAs we decided to set the null countrynames to value 'Other' to avoid dropping data.
- The RunningTime field of the table running\_times was corrupted, each cell had its own time format. We first dropped the entries which had RunningTime =NAN( which represent *7% of the table*) as they don't add any information. Then to make the field uniform we decided to keep all the running times in minutes.  
 For that we created some patterns (example : `str.extract('(\d+)\.|\.\|\s*minutes|\s*mins|\s*min')`) and dropped the entries which does not fit this pattern( which represent *0.1% of the table*) (see fig1 & fig2 below for data before/after cleaning)

	ClipId	ReleaseCountry	RunningTime
	87	1778609	Australia
	1780	1819408	UK
	2087	1827372	Germany
	3834	1857131	NaN
	5020	1875416	UK
	7899	1949906	Australia
	9239	1972720	USA
	9805	1984856	Germany
	12331	2022709	NaN
	14634	2052443	UK

*figure1: some of the data before cleaning*

ClipId	CountryName	runtime
0	USA	83
1	Other	30
2	USA	30
3	Other	30
4	Other	30
5	Other	30

*figure2: some of the data after cleaning*

- After examining the queries (in particular query h of deliverable 3), we decided to represent the NAN values in Act.OrdersCredit with the default value 0 that way we can have the column as INTEGER type.
- The Spouse field of the Biographies table had many informations at once (name of the spouse, beginning and ending of the marriage, number of children, status of the relationship). Thus we created a pattern and extracted those informations under different fields. (some patterns used were `str.extract("\;(.*)")`, `str.extract(")\|(.*)\|")`)  
 There were also many spouses on the same field separated by this character |. So we extracted them each on a different line. (see *fig3 & fig4 below for data before/after cleaning*)

Spouse	pid
'Jacklyn Zeman' (14 February 1979 - 1981) (divorced)	30
'Angela Arabo' (? - ?)	47
'Rosamond Broughton' (10 October 1938 - 16 March 1995) (his death); 6 children	103
'Linda' (? - 5 August 2005) (his death)	143
'Almina Wombwell' (26 June 1895 - 5 April 1923) (his death); 2 children	215
'Ottilie Losch' (1 September 1939 - 1947) (divorced) 'Anne Wendell' (17 July 1922 - 1936) (divorced); 2 children	230
'Jean Wallop' (7 January 1956 - 11 September 2001) (his death); 3 children	237
'Fiona Aitken' (18 February 1998 - present); 1 child 'Jayne Wilby' (16 December 1989 - January 1998) (divorced); 2 children	247
'Dr. Lao Sealey' (23 April 2005 - present)	321
'Krishnaveni Jikki' (? - present)	329
'Maurice Jose de Leon' (24 June 1961 - 11 June 1964) (divorced) 'Joseph E. McDonald' (6 April 1967 - January 1969) (divorced) 'Ronald Fisher' (1970 - 5 January 2010) (her death); 1 child	387

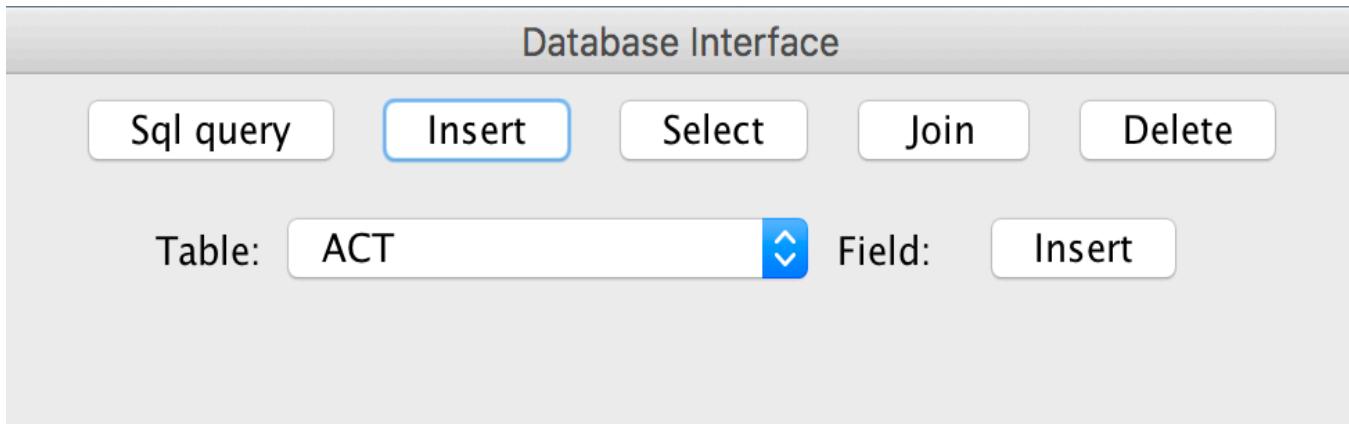
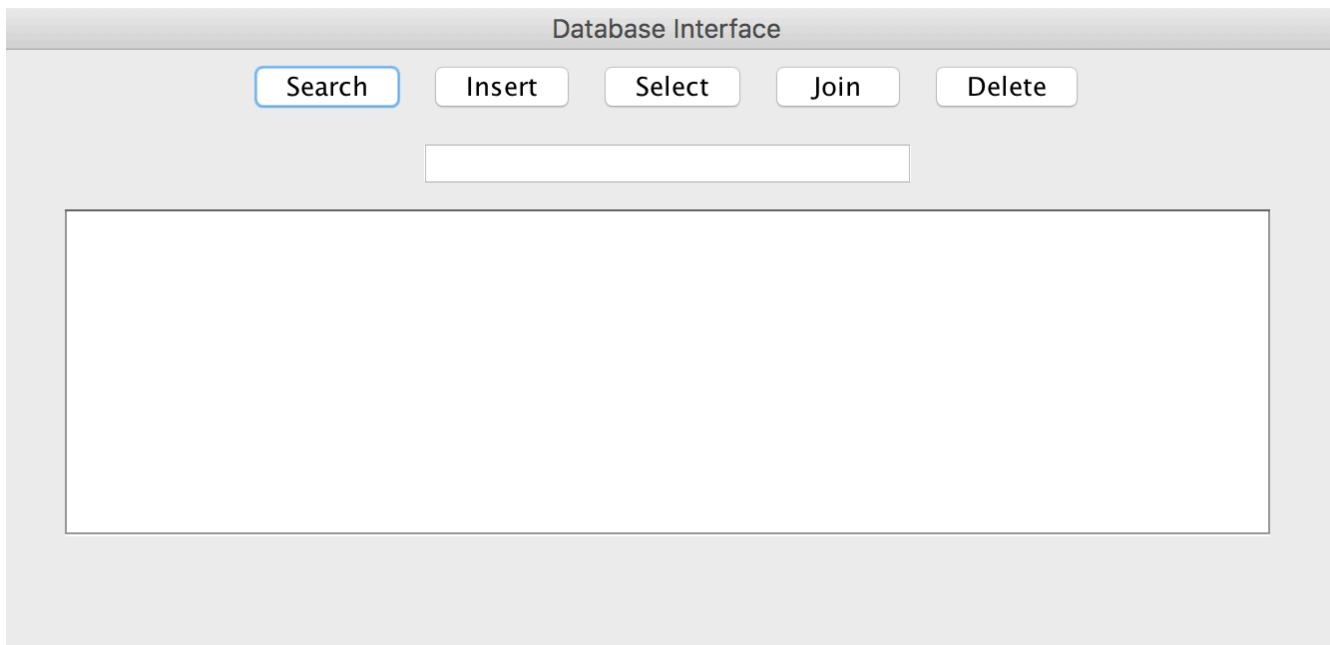
*figure3: data before*

name	pid	sid	from	to	children	status
Jacklyn Zeman	30	40876	1979.0	1981.0	NaN	divorced
Angela Arabo	47	5002	NaN	NaN	NaN	NaN
Rosamond Broughton	103	83183	1938.0	1995.0	6 children	his death
Linda	143	57765	NaN	2005.0	NaN	his death
Almina Wombwell	215	3565	1895.0	1923.0	2 children	his death
Ottilie Losch	230	74510	1939.0	1947.0	NaN	divorced
Anne Wendell	230	6424	1922.0	1936.0	2 children	divorced
Jean Wallop	237	43561	1956.0	2001.0	3 children	his death
Fiona Aitken	247	30327	1998.0	NaN	1 child	NaN
Jayne Wilby	247	43254	1989.0	1998.0	2 children	divorced

*figure4: data after*

## ***Interface***

Here we implemented our first version of the interface. You can find the screenshots below :



Database Interface

Sql query   Insert   **Select**   Join   Delete

Table: ACT   Field: WHERE:  Launch query



Database Interface

Sql query   Insert   Select   Join   Delete

Table: ACT   Field: WHERE:  Launch query



## Deliverable 3

### ***Query Implementation***

#### MILESTONE 2

##### Query a:

*Description of logic:* Prints the name and length of the 10 longest clips that were released in France.

```
SELECT c.title, rt.runtime  
  
FROM runs_for rf, runningTimes rt, clips c, countries cn  
  
WHERE c.CLIPIDS=rf.clipid AND rf.rid=rt.rid AND rf.countid=cn.countid  
  
AND cn.COUNTRYNAME='France'  
  
ORDER BY rt.runtime DESC  
  
FETCH FIRST 10 ROWS ONLY;
```

##### Query b:

*Description of logic:* Computes the number of clips released per country in 2001.

```
SELECT cn.countryname, COUNT(r.clipid) AS number_of_clips  
  
FROM released_in r, releasedates rd, countries cn  
  
WHERE r.did=rd.DID AND rd.RELEASEDATE='2001' AND cn.COUNTID=r.COUNTID  
  
GROUP BY cn.countryname;
```

##### Query c:

*Description of logic:* Computes the numbers of clips per genre released in the USA after 2013.

```
SELECT g.genre, COUNT(hg.clipid) AS number_of_clips  
  
FROM clip_genre g, has_genre hg, released_in r, RELEASEDATES rd, countries cn  
  
WHERE hg.gid=g.gid AND r.DID=rd.DID AND rd.RELEASEDATE>2013 AND r.countid=cn.countid  
  
AND cn.countryname='USA' AND r.CLIPID=hg.CLIPID  
  
GROUP BY g.genre;
```

**Query d :**

*Description of logic:* Prints the name of actor/actress who has acted in more clips than anyone else.

```
SELECT fullname FROM  
(SELECT p.fullname, COUNT(ac.clipids) AS number_of_clips  
FROM people p, act ac, clips c  
WHERE p.pid=ac.PID and ac.CLIPIDS=c.CLIPIDS  
GROUP BY p.fullname  
ORDER BY number_of_clips DESC  
FETCH FIRST 1 ROWS ONLY);
```

**Query e :**

*Description of logic:* Prints the maximum number of clips any director has directed.

```
SELECT number_of_clips FROM  
(SELECT d.pid, COUNT(d.clipids) AS number_of_clips  
FROM direct d, clips c  
WHERE d.CLIPIDS=c.CLIPIDS  
GROUP BY d.pid  
ORDER BY number_of_clips DESC  
FETCH FIRST 1 ROWS ONLY);
```

### Query f:

*Description of logic:* Prints the names of people that had at least 2 different jobs in a single clip.

```
SELECT people.fullname
FROM people
WHERE people.pid IN
(SELECT pid FROM
(SELECT pid, Clipids FROM act GROUP BY pid, Clipids
UNION ALL
SELECT pid, Clipids FROM direct
UNION ALL
SELECT pid, Clipids FROM produce
UNION ALL
SELECT pid, Clipids FROM write)
GROUP BY pid, clipids
HAVING COUNT(*) > 1);
```

### Query g:

*Description of logic:* Print the 10 most common clip languages.

```
SELECT language FROM
(SELECT cl.language, COUNT(hl.clipid) AS number_of_clips
FROM clip_languages cl, has_language hl
WHERE cl.lid=hl.lid
GROUP BY cl.language
ORDER BY number_of_clips DESC
FETCH FIRST 10 ROWS ONLY);
```

**Query h:**

*Description of logic:* Prints the full name of the actor who has performed in the highest number of clips with a user-specified type.

--suppose the user specified type is Fantasy

```
SELECT fullname FROM
(SELECT p.fullname, COUNT(ac.clipids) AS number_of_clips
FROM people p, act ac, clips c, has_genre hg, clip_genre g
WHERE p.pid=ac.PID AND ac.CLIPIDS=c.CLIPIDS
AND c.clipids=hg.clipid AND hg.gid=g.gid AND g.GENRE='Fantasy'
GROUP BY p.fullname
ORDER BY number_of_clips DESC
FETCH FIRST 1 ROWS ONLY);
```

**MILESTONE 3**

**Query a:**

*Description of logic:* Prints the names of the top 10 actors ranked by the average rating of their 3 highest-rated clips that were voted by at least 100 people.

```
SELECT p.fullname
FROM people p
WHERE p.pid IN
(SELECT pid FROM
(SELECT ac.pid,c.rank, row_number()
OVER (PARTITION BY ac.pid ORDER BY c.rank DESC) AS index_clip_per_actor
FROM clips c, act ac
WHERE ac.CLIPIDS=c.CLIPIDS AND c.votes>100 AND ac.pid IN
(SELECT pid FROM
(SELECT DISTINCT ac.pid, ac.clipids
FROM act ac)
GROUP BY pid
HAVING count(*)>5))
WHERE index_clip_per_actor<4
GROUP BY pid
ORDER BY AVG(rank) DESC
FETCH FIRST 10 ROWS ONLY);
```

### **Query b:**

*Description of logic:* Computes the average rating of the top-100 rated clips per decade in decreasing order.

```
SELECT decade, AVG(rank) FROM
(SELECT clipids, rank, decade,
row_number() OVER (PARTITION BY decade ORDER BY rank DESC) AS index_clip_per_decade
FROM
(SELECT clipids, rank, FLOOR(year/ 10) * 10 AS decade
FROM clips
WHERE rank IS NOT NULL and year IS NOT NULL)
WHERE index_clip_per_decade < 101
GROUP BY decade
ORDER BY AVG(rank) DESC;
```

### **Query c:**

*Description of logic:* For any video game director, prints the first year he/she directed a game, his/her name and all his/her game titles from that year.

```
SELECT fullname, year, game_titles FROM
(SELECT year, fullname, game_titles,
row_number() OVER (PARTITION BY fullname ORDER BY year) AS index_year_per_pid
FROM
(SELECT c.year, p.fullname,
LISTAGG(c.title, ', ') WITHIN GROUP (ORDER BY c.title) AS game_titles
FROM direct d, clips c, people p
WHERE d.clipids=c.CLIPIDS AND p.pid=d.pid AND c.type='VG'
GROUP BY p.fullname,c.year))
WHERE index_year_per_pid=1;
```

## Query d:

**Description of logic:** For each year, prints the title, year and rank-in-year of top 3 clips, based on their ranking.

```
SELECT year,  
LISTAGG(title, ' / ') WITHIN GROUP (ORDER BY rank DESC) as top3_clips_titles,  
LISTAGG(rank, ' / ') WITHIN GROUP (ORDER BY rank DESC) as top3_clips_ranks  
FROM  
(SELECT year, title, rank,  
row_number() OVER (PARTITION BY year ORDER BY rank DESC) AS index_clip_per_year  
FROM clips  
WHERE year IS NOT NULL and rank IS NOT NULL)  
WHERE index_clip_per_year<4  
GROUP BY year;
```

### Query e:

**Description of logic:** Prints the names of all directors who have also written scripts for clips, in all of which they were additionally actors (but not necessarily directors) and every clip they directed has at least two more points in ranking than any clip they wrote.

```
SELECT p.fullname FROM people p,  
  
(WITH T AS  
  
(SELECT d.pid, c.rank  
  
FROM direct d, clips c  
  
WHERE c.clipids=d.clipids AND c.rank IS NOT NULL  
  
AND d.pid IN (SELECT DISTINCT pid FROM write)  
  
AND d.pid NOT IN  
  
(SELECT distinct pid FROM  
  
(SELECT w.pid, w.clipids FROM write w
```

```

    WHERE (w.pid, w.clipids) NOT IN
        (SELECT ac.pid, ac.clipids FROM act ac)))
    SELECT T.pid, MIN(T.rank) AS min_rank_directed, MAX(c.rank) AS max_rank_written
    FROM T, WRITE w, clips c
    WHERE w.pid=T.pid AND w.clipids=c.clipids AND c.rank IS NOT NULL
    GROUP BY T.pid) S
    WHERE S.min_rank_directed > S.max_rank_written+2 AND p.pid=S.pid;

```

#### Query f:

*Description of logic:* Prints the names of the actors that are not married and have participated in more than 2 clips that they both acted in and co-directed it.

```

    SELECT DISTINCT p.fullname
    FROM people p, married m
    WHERE p.pid NOT IN m.PID AND p.pid IN
        (SELECT pid FROM
            (SELECT pid, count(*) as nbr_clips_he_acted_directed from
                (SELECT pid, clipids FROM
                    (SELECT pid, clipids, COUNT(*) AS number_of_jobs from
                        (SELECT DISTINCT pid, Clipids FROM act
                        UNION ALL
                        SELECT pid, Clipids FROM direct)
                    GROUP BY pid , clipids)
                WHERE number_of_jobs>1)
            GROUP BY pid)
        WHERE nbr_clips_he_acted_directed>2);

```

### **Query g:**

*Description of logic:* Prints the names of screenplay story writers who have worked with more than 2 producers.

```
SELECT fullname FROM people WHERE PID IN
  (SELECT w.pid FROM
    (SELECT w.pid, w.clipids
     FROM write w
     WHERE w.worktypes='screenplay') w
   LEFT OUTER JOIN
     (SELECT p.clipids, COUNT(*) AS number_of_producers
      FROM produce p
      GROUP BY p.clipids) p
   ON w.clipids=p.clipids
   WHERE number_of_producers>2);
```

### **Query h:**

*Description of logic:* Computes the average rating of an actor's clips (for each actor) when she/he has a leading role.

```
WITH T AS
  (SELECT pid, AVG(rank) AS average
   FROM
     (SELECT DISTINCT ac.pid, ac.clipids, ac.orderscredit, c.rank
      FROM act ac, clips c
      WHERE orderscredit<4 AND orderscredit>0 AND rank IS NOT NULL AND c.clipids=ac.clipids)
   GROUP BY pid)
SELECT p.fullname, T.average
FROM people p, T
WHERE p.pid=T.pid;
```

### Query i:

*Description of logic:* Computes the average rating for the clips whose genre is the most popular genre.

```
SELECT AVG(c.rank)

FROM clips c, has_genre hg,
(SELECT gid, COUNT(*) AS nbr_clips

FROM has_genre

GROUP BY gid

ORDER BY nbr_clips DESC

FETCH FIRST 1 ROWS ONLY) s

WHERE hg.clipid=c.CLIPIDS AND hg.gid=s.gid

GROUP BY hg.gid;
```

### Query j:

*Description of logic:* Prints the names of the actors that have participated in more than 100 clips, of which at least 60% were short but not comedies nor dramas, and have played in more comedies than double the dramas. Prints also the number of comedies and dramas each of them participated in.

```
SELECT p.fullname, l.nbr_of_comedy_clips, l.nbr_of_Drama_clips FROM people p,

(SELECT DISTINCT s1.pid, s1.nbr_of_clips_per_genre AS nbr_of_comedy_clips, s2.nbr_of_clips_per_genre AS
nbr_of_Drama_clips

FROM

(SELECT pid, genre, COUNT(*) AS nbr_of_clips_per_genre FROM

(SELECT pid, cg.genre

FROM act a, clip_genre cg, has_genre hg

WHERE a.clipids=hg.clipid AND hg.gid=cg.gid AND (cg.genre='Comedy' OR cg.genre='Drama'))

GROUP BY pid, genre) s1

JOIN

(SELECT pid, genre, COUNT(*) AS nbr_of_clips_per_genre FROM

(SELECT pid, cg.genre
```

```

FROM act a, clip_genre cg, has_genre hg

WHERE a.clipids=hg.clipid AND hg.gid=cg.gid AND (cg.genre='Comedy' OR cg.genre='Drama')

GROUP BY pid, genre) s2

ON s1.pid=s2.pid

WHERE s1.genre='Comedy' AND s1.nbr_of_clips_per_genre>2*s2.nbr_of_clips_per_genre AND
s2.genre='Drama'

AND s1.pid IN

(SELECT R.pid FROM

(SELECT pid, genre, nbr_of_clips_of_actor, COUNT(*) AS nbr_of_clips_per_genre

FROM

(WITH T AS

(SELECT DISTINCT pid, nbr_of_clips_of_actor FROM

(SELECT a.pid, COUNT(*) AS nbr_of_clips_of_actor

FROM act a

GROUP BY a.pid)

WHERE nbr_of_clips_of_actor>100)

SELECT T.pid, T.nbr_of_clips_of_actor, cg.genre

FROM T, act a, clip_genre cg, has_genre hg

WHERE T.pid=a.pid AND a.clipids=hg.clipid AND hg.gid=cg.gid)

GROUP BY pid, genre, nbr_of_clips_of_actor) R

WHERE (R.nbr_of_clips_per_genre > 0.6 * R.nbr_of_clips_of_actor and R.genre='Short')) I

WHERE p.pid=l.pid;
    
```

### Query k:

*Description of logic:* Prints the number of Dutch movies whose genre is the second most popular one.

--We considered as movies the clips with types 'TV' and 'V'

```
SELECT COUNT(*) AS nbr_of_movies
FROM CLIP_LANGUAGES cl, has_language hl, clips c, has_genre hg
WHERE cl.lid=hl.lid AND cl.LANGUAGE='Dutch' AND c.CLIPIDS=hl.CLIPID AND
c.CLIPIDS=hg.CLIPID AND (c.type='V' or c.type='TV') AND hg.gid IN
(SELECT gid FROM
(SELECT gid, rownum AS rn FROM
(SELECT gid, COUNT(*) AS nbr_clips
FROM has_genre
GROUP BY gid
ORDER BY nbr_clips DESC
FETCH FIRST 2 ROWS ONLY))
WHERE rn=2);
```

### Query l:

*Description of logic:* Prints the name of the producer whose role is coordinating producer, and who has produced the highest number of movies with the most popular genre.

--We considered as movies the clips with types 'TV' and 'V'

```
WITH T AS
(SELECT pid, COUNT(*) as nbr_of_clips from
(SELECT p.pid, p.CLIPIDS
FROM produce p, has_genre hg, clips c
WHERE p.ROLES='coordinating producer' AND p.CLIPIDS=hg.clipid AND p.CLIPIDS=c.clipids
AND (c.type='V' OR c.type='TV') AND hg.GID IN
(SELECT gid FROM
```

```
(SELECT gid, COUNT(*) AS nbr_clips
  FROM has_genre
  GROUP BY gid
  ORDER BY nbr_clips DESC
  FETCH FIRST 1 ROWS ONLY))
GROUP BY pid
ORDER BY nbr_of_clips DESC
FETCH FIRST 1 ROW ONLY)

SELECT p.fullname
  FROM people p, T
 WHERE p.pid=T.pid;
```

## *Query Analysis*

First of all, we rewrote some of our queries, to make them more efficient. We excluded projections that were not necessary and we factored the subqueries that were used multiple times in the same statement. This way we were able to improve the running time of some queries (figures on the next page)

**Costs before**

```
--query a
select p.fullname
from people p
where p.pid in
  (select pid from
   (select pid,avg(rank) from
    (select ac.pid,c.CLIPIDS,c.rank, row_number()
     over (partition by ac.pid order by c.rank desc) as index_clip
    from clips c, act ac
    where ac.CLIPIDS=c.CLIPIDS and c.votes>100 and ac.pid in
      (select pid from
       (select ac.pid, ac.clipids, count(*)
        from act ac
        group by ac.pid, ac.clipids)
       group by pid
      )
     )
    )
   )
  )
  (select pid from
   (select distinct ac.pid, ac.clipids
    from act ac
    group by pid
   )
  )
  group by pid
  having count(*)>5
  where index_clip_per_actor<4
  group by pid
  order by avg(rank) desc
  FETCH FIRST 10 ROWS ONLY);
```

**Costs after**

```
--query a
select p.fullname
from people p
where p.pid in
  (select pid from
   (select ac.pid,c.rank, row_number()
    over (partition by ac.pid order by c.rank desc) as index_c
    from clips c, act ac
    where ac.CLIPIDS=c.CLIPIDS and c.votes>100 and ac.pid in
      (select pid from
       (select distinct ac.pid, ac.clipids
        from act ac
        group by pid
       )
      )
     )
    )
  )
  (select pid from
   (select ac.pid, ac.clipids, count(*)
    from act ac
    group by ac.pid, ac.clipids)
   group by pid
  )
  )
  group by pid
  having count(*)>5
  where index_clip_per_actor<4
  group by pid
  order by avg(rank) desc
  FETCH FIRST 10 ROWS ONLY);
```

**Résultat de requête x**

SQL | Toutes les lignes extraites : 10 en 37,045 secondes

```
--query e
select p.fullname from people p,
  (with T as
   (select d.pid, d.clipids, c.rank
    from direct d, clips c
    where c.clipids=d.clipids and c.rank is not null
    and d.pid in (select distinct pid from write)
    and d.pid not in
      (select distinct pid from
       (select w.pid, w.clipids from write w minus
        (select w.pid, w.clipids from write w, act ac
         where w.pid=ac.pid and w.clipids=ac.clipids)))
      )
    select T.pid, min(T.rank) as min_rank_directed, max(c.rank) as
    from T, write w, clips c
    where w.pid=T.pid and w.clipids=c.clipids and c.rank is not null
    group by T.pid) S
  where S.min_rank_directed > S.max_rank_written+2 and p.pid=S.pid;
```

**Résultat de requête x**

SQL | Toutes les lignes extraites : 10 en 30,776 secondes

**Résultat de script x**

**Résultat de requête x**

SQL | Toutes les lignes extraites : 22 en 15,496 secondes

```
--query l
with T as
  (select pid, count(*) as nbr_of_clips from
  (select p.pid, p.CLIPIDS
   from produce p, has_genre hg, clips c
   where p.ROLES='coordinating producer' and p.CLIPIDS=hg.clipid
   and (c.type='V' or c.type='TV') and hg.GID in
     (select gid from
      (select gid, count(*) as nbr_clips
       from has_genre
       group by gid
       order by nbr_clips desc
       FETCH FIRST 1 ROWS ONLY))
     )
   group by pid
   order by nbr_of_clips desc
   FETCH first 1 row only)
  select p.fullname
  from people p, T
  where p.pid=T.pid;
```

**Résultat de requête x**

SQL | Toutes les lignes extraites : 1 en 0,927 secondes

```
--query l
with T as
  (select pid from
  (select p.pid, p.CLIPIDS
   from produce p, has_genre hg, clips c
   where p.ROLES='coordinating producer' and p.CLIPIDS=hg.clipid
   and (c.type='V' or c.type='TV') and hg.GID in
     (select gid
      from has_genre
      group by gid
      order by count(*) desc
      FETCH FIRST 1 ROWS ONLY))
   group by pid
   order by count(*) desc
   FETCH first 1 row only)
  select p.fullname
  from people p, T
  where p.pid=T.pid;
```

**Résultat de requête x**

SQL | Toutes les lignes extraites : 1 en 0,792 secondes

## Optimizations using indexes :

*Query C*

**Initial Running time:** 1,107 seconds



**Optimized Running time:** 0,936 seconds

Résultat de requête x

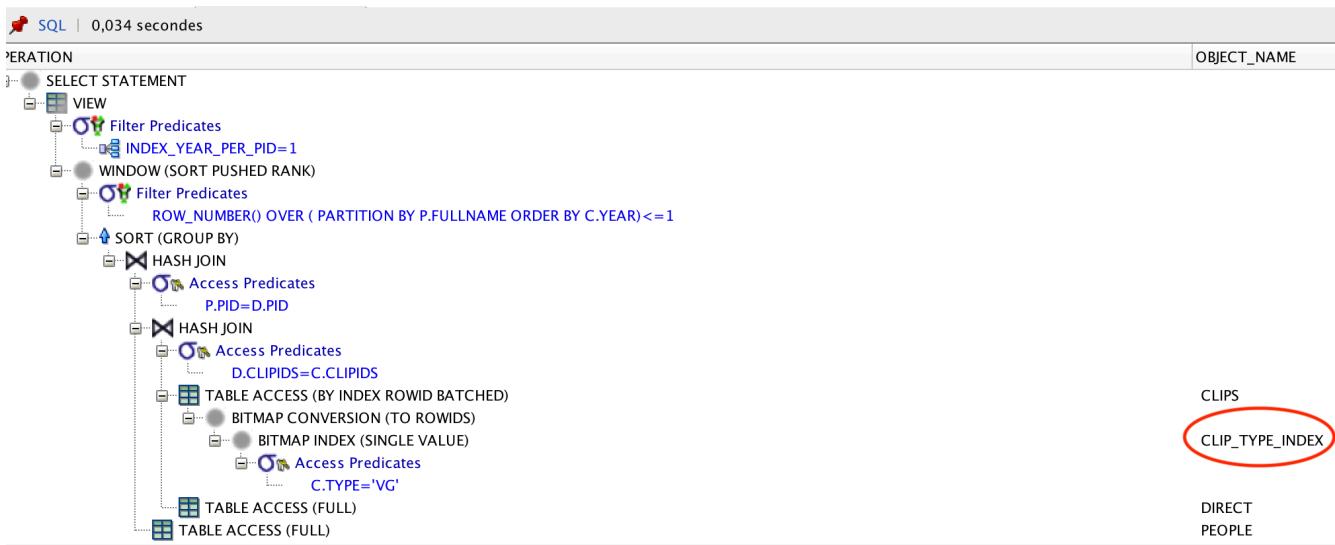
SQL | 50 lignes extraites en 0,936 secondes

FULLNAME	YEAR	GAME_TITLES
1 Abe Goro (II)	...	2003 Atsumare!! Made in Wario
2 Abe Masamichi	...	1995 Tekken 2
3 Abe Naomitsu	...	1997 G Darius
4 Abe Yukio (III)	...	1997 RayStorm
5 Abed Ramzi	...	2008 Slumber Party Slaughterhouse: The Game
6 Adair Peter (I)	...	1995 In the 1st Degree
7 Adams Mike (XII)	...	2003 Wanted: A Wild Western Adventure
8 Agostini De	...	1997 Il tesoro di Venezia
9 Airey Jeremy	...	1997 Clayfighter 63 1/3
10 Akamatsu H.	...	1987 Dracula II: Noroi no fūin
11 Akatsuka Tetsuya	...	2005 Soulcalibur III
12 Akiaten Brandon	...	2007 NBA '08 Featuring the Life: Vol 3
13 Akibayashi Koji	...	1987 Heavy Barrel
14 Akiyama Naoki	...	2005 Radiata Stories

### Initial plan :



### Improved plan :



For this query, we decided to create a bitmap index on the attribute Type of the entity Clips, in order to avoid doing a full scan on the table Clips, and then to minimize the cost. The Bitmap index does not store a pointer to a record in a file sorted on the index, but a value encoded on one bit (true or false) for each value in the column indexed (2 bits for a cardinality 2, 3 for a cardinality 3, etc.) in a file sorted on the key. Bitmap indexes make searching very quick and easy in some cases and that's why we are essentially using these indexes in this project.

*Query J*

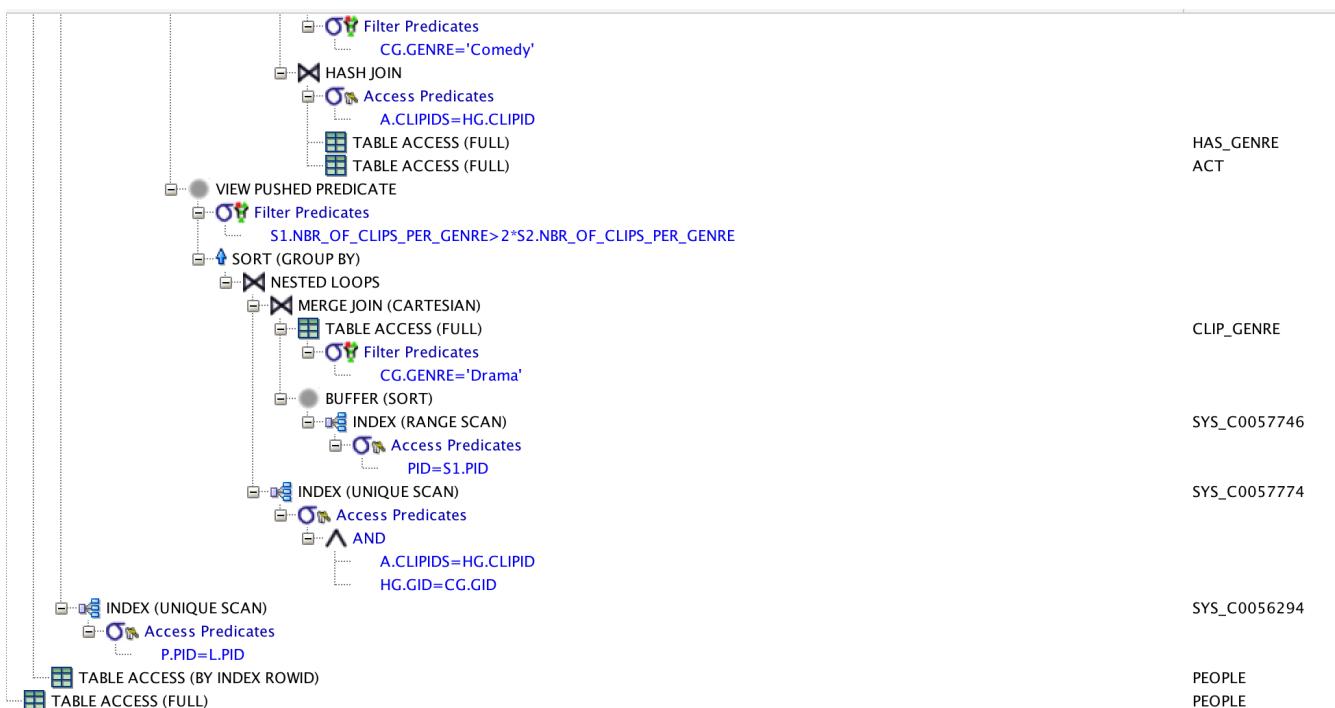
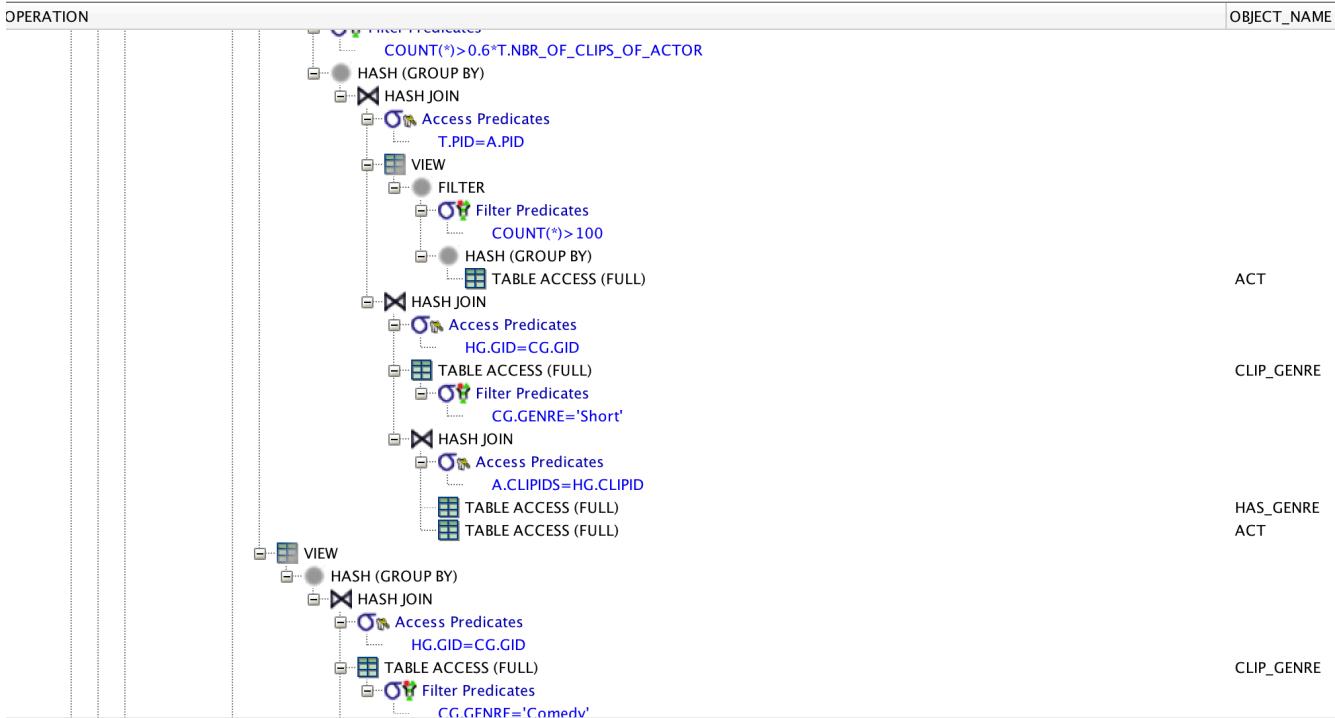
**Initial running time :** 16,775 seconds



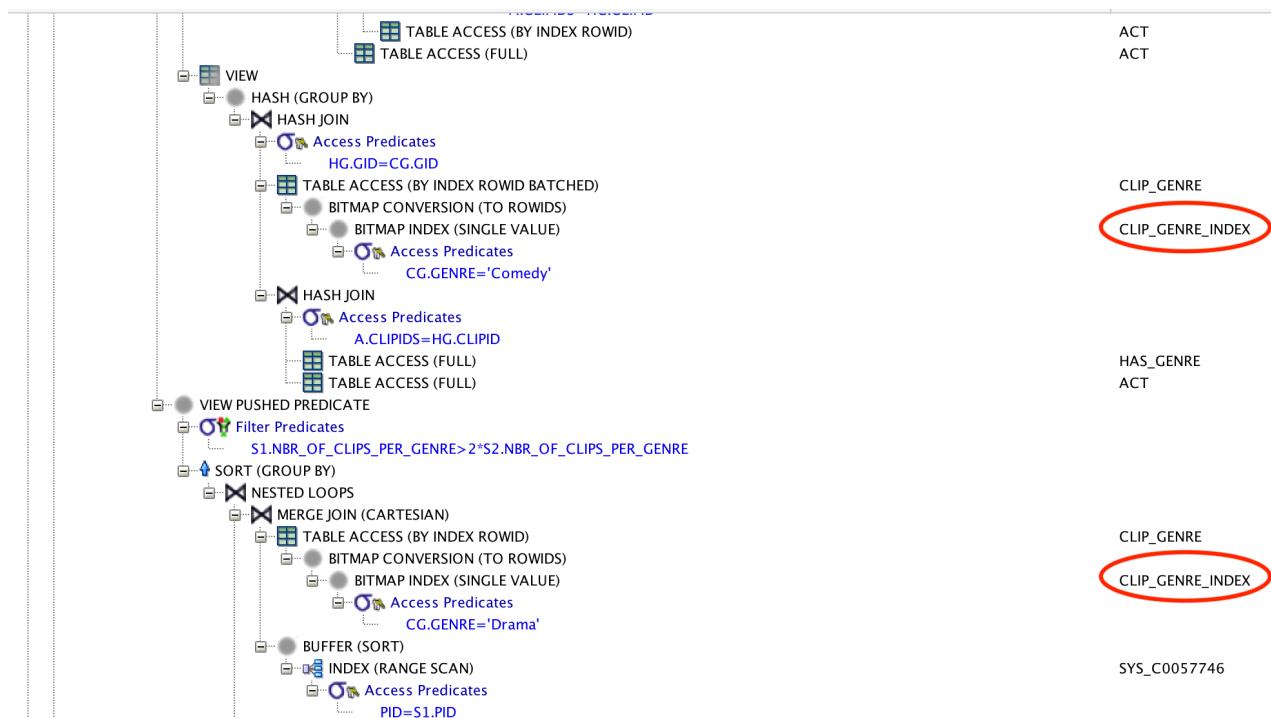
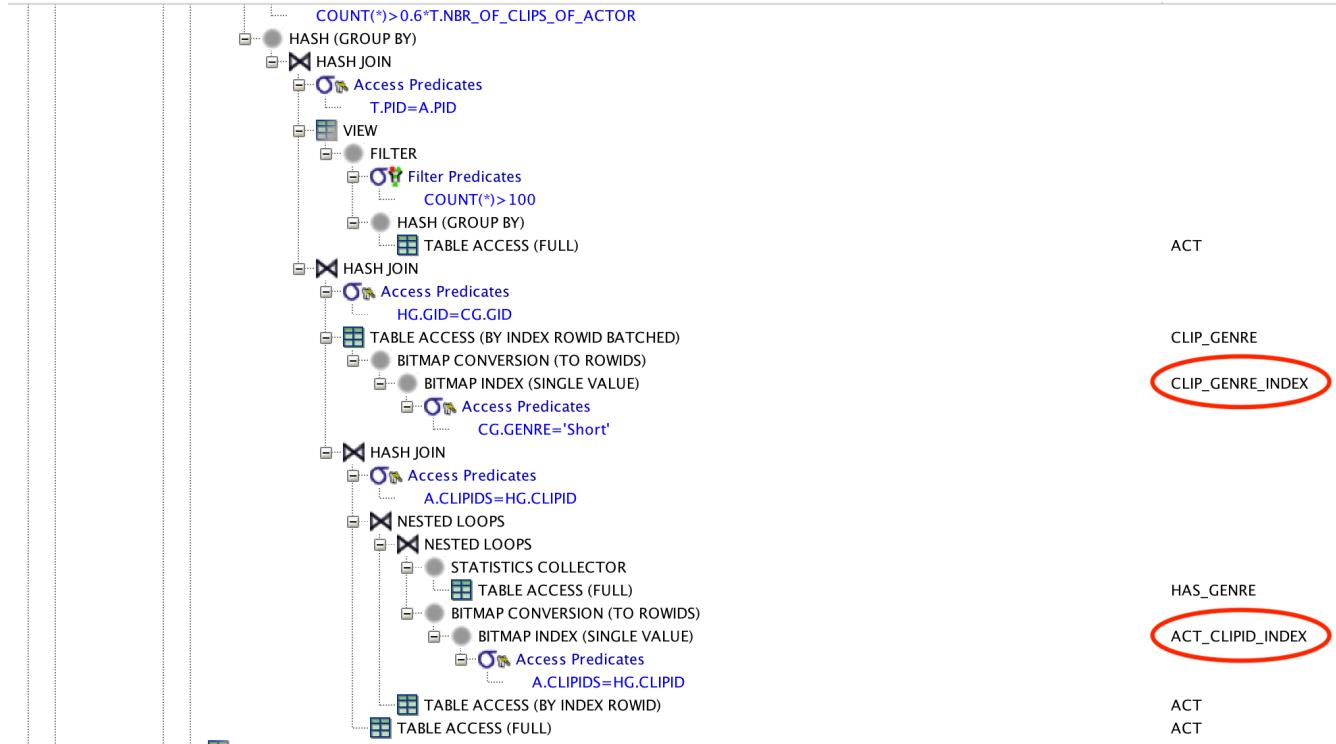
**Optimized running time :** 9,532 seconds

	FULLNAME	NBR_OF_COMEDY_CLIPS	NBR_OF_DRAMA_CLIPS
1	Chamberlin Riley	...	75
2	Turpin Ben	...	114
3	Thomas Billie 'Buckwheat'	...	96
4	Roland Ruth	...	113
5	Smith Pete (I)	...	101
6	Brennan John E.	...	121
7	Burns Bobby (I)	...	132
8	Sterling Ford	...	145
9	Deed André	...	152
10	Questel Mae	...	252
11	Dillon Edward	...	154
12	Arbuckle Roscoe 'Fatty'	...	114
13	Bunny John	...	121
14	Nash Clarence	...	174

**Initial plan :**



**Improved plan :**



For this query, we tried to pay attention to the initial plan, and we noticed that there are several fully scan on the table Clip\_genre to access to the attribute genre. So, we decided to create an index on this attribute in order to minimize the cost and to improve the performance : it was successful.

We also created an index on the attribute ClipIds of the table Act to avoid on fully scan on this table.

### Query K

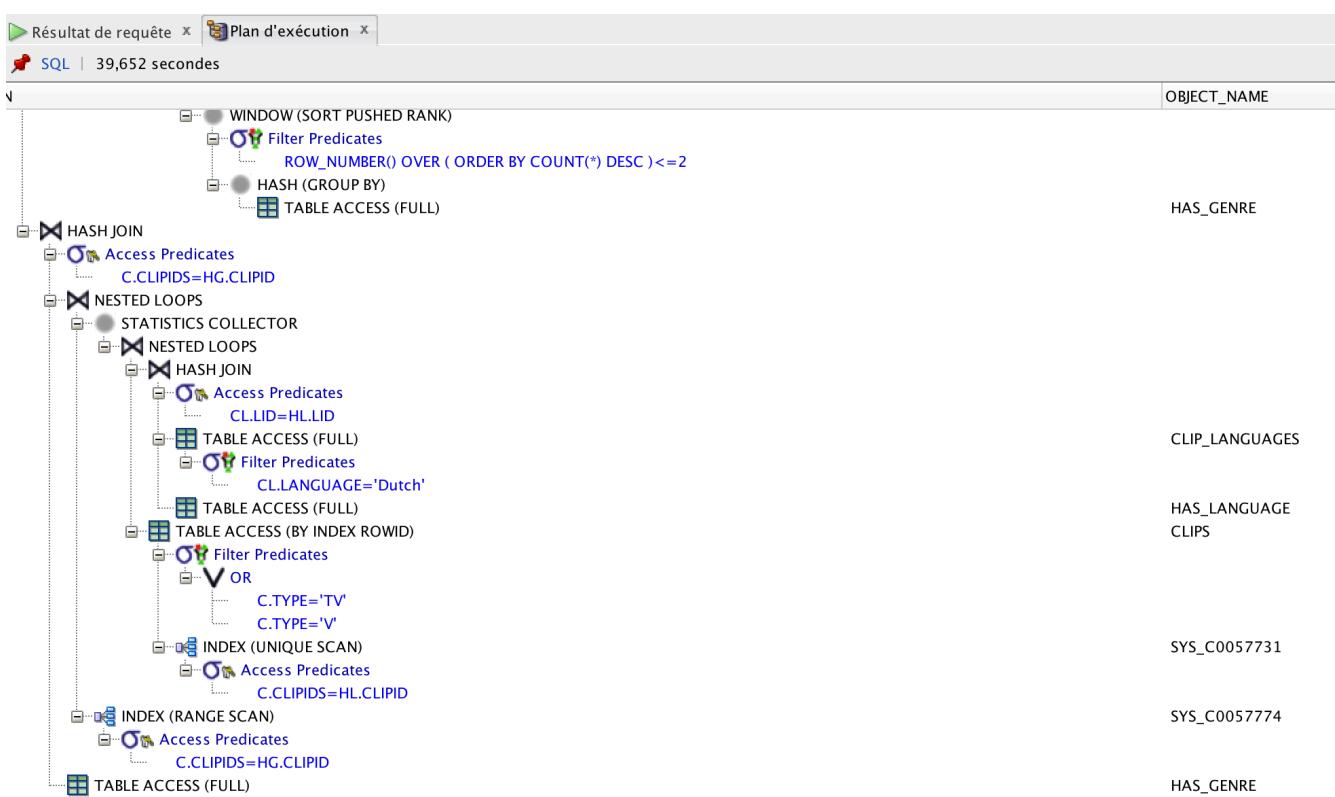
#### Initial running time :

SQL   Toutes les lignes extraites : 1 en 0,679 secondes	
NBR_OF_MOVIES	
1	867

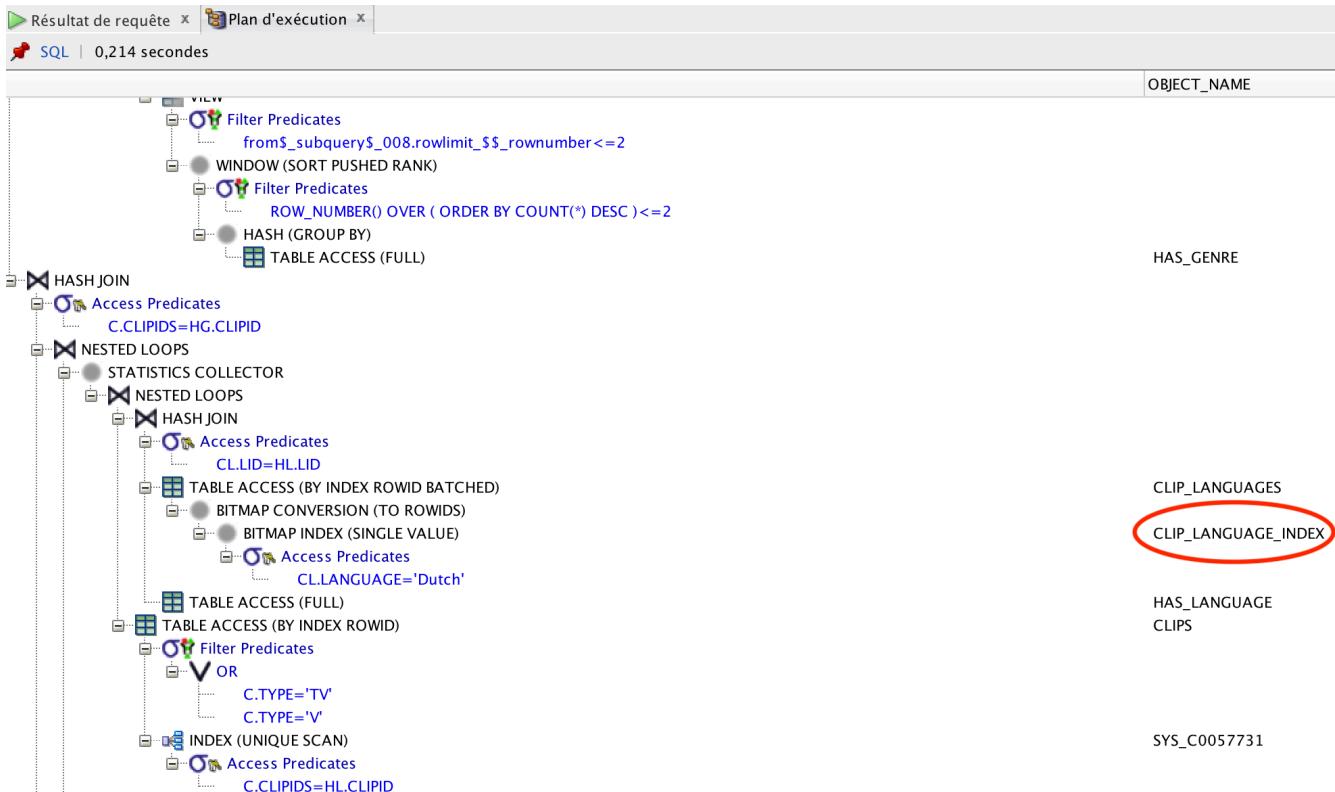
#### Optimized running time :

SQL   Toutes les lignes extraites : 1 en 0,473 secondes	
NBR_OF_MOVIES	
1	867

#### Initial plan :



### Improved plan :



As for the other queries, we tried to avoid some fully scan. Indeed, we created an index on the attribute language of the table Clip\_languages

## Interface

### Design logic Description

We kept the same implementation for the interface (a java application), but we improved it and added the following functionalities:

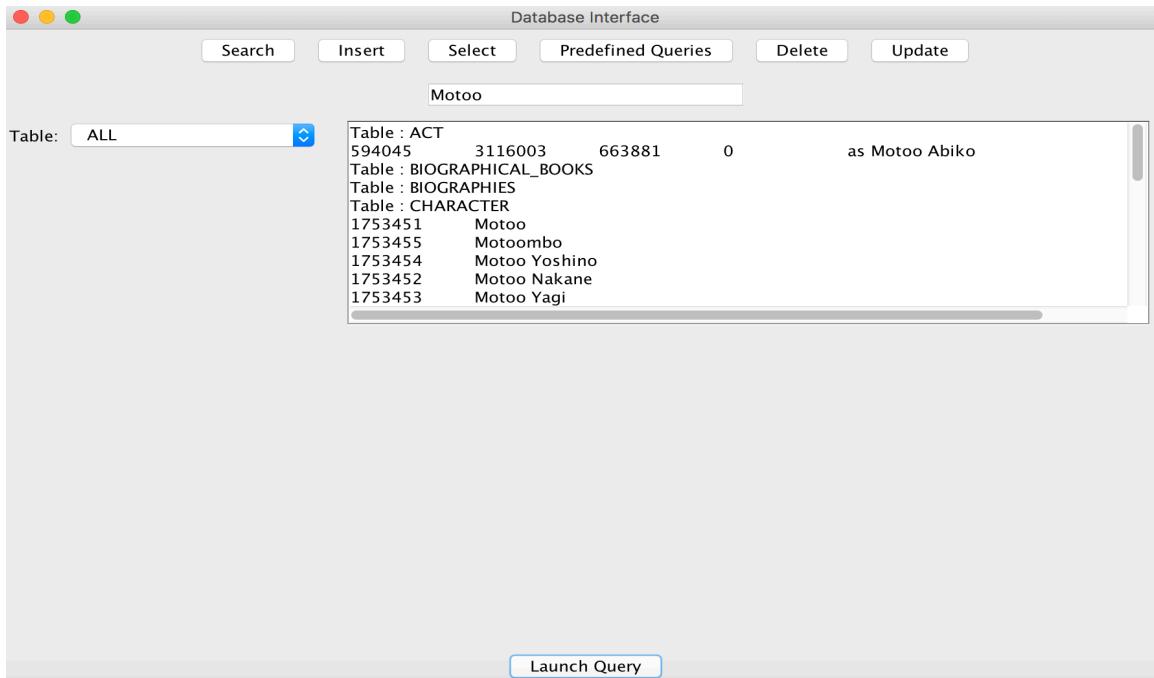
- Predefined queries: Running and obtaining the results of all the queries that we needed to implement for this project.
- Search: Improved the key-word based search: the user can now search in a specific table or in all the tables of our database. We didn't see the utility in searching numbers, we have many tables containing only ids, so our keyword-based search concerns only variables with type char or varchar in the tables that contain at least such an attribute
- Update: We added the possibility to update/modify the values of the attributes of chosen rows in a table.
- Insert, Select and Delete functionalities stayed the same.

### Screenshots

Here are some screen shots of the functionalities of our interface:

#### 1) Keyword based search:

In all the tables:



In a specific table:

The screenshot shows a Mac OS X-style window titled "Database Interface". At the top, there are buttons for Search, Insert, Select, Predefined Queries, Delete, and Update. A search bar contains the text "Kevin". Below the search bar, a dropdown menu labeled "Table: CHARACTER" is open, showing a list of character names and their IDs. The list includes Kevin Sharma, Kevin, Kevin Austin, P.A. Kevin, Kevin Parsons, Kevin Scrimgeour, Kevin Harrelson, Kevin Kincaid #2 (1971-1974), and Kevin Hesse. At the bottom right of the interface is a "Launch Query" button.

ID	Name
1342560	Kevin Sharma
1341272	Kevin
1341393	Kevin Austin
1927282	P.A. Kevin
1342372	Kevin Parsons
1342553	Kevin Scrimgeour
1341890	Kevin Harrelson
1342029	Kevin Kincaid #2 (1971-1974)
1341917	Kevin Hesse

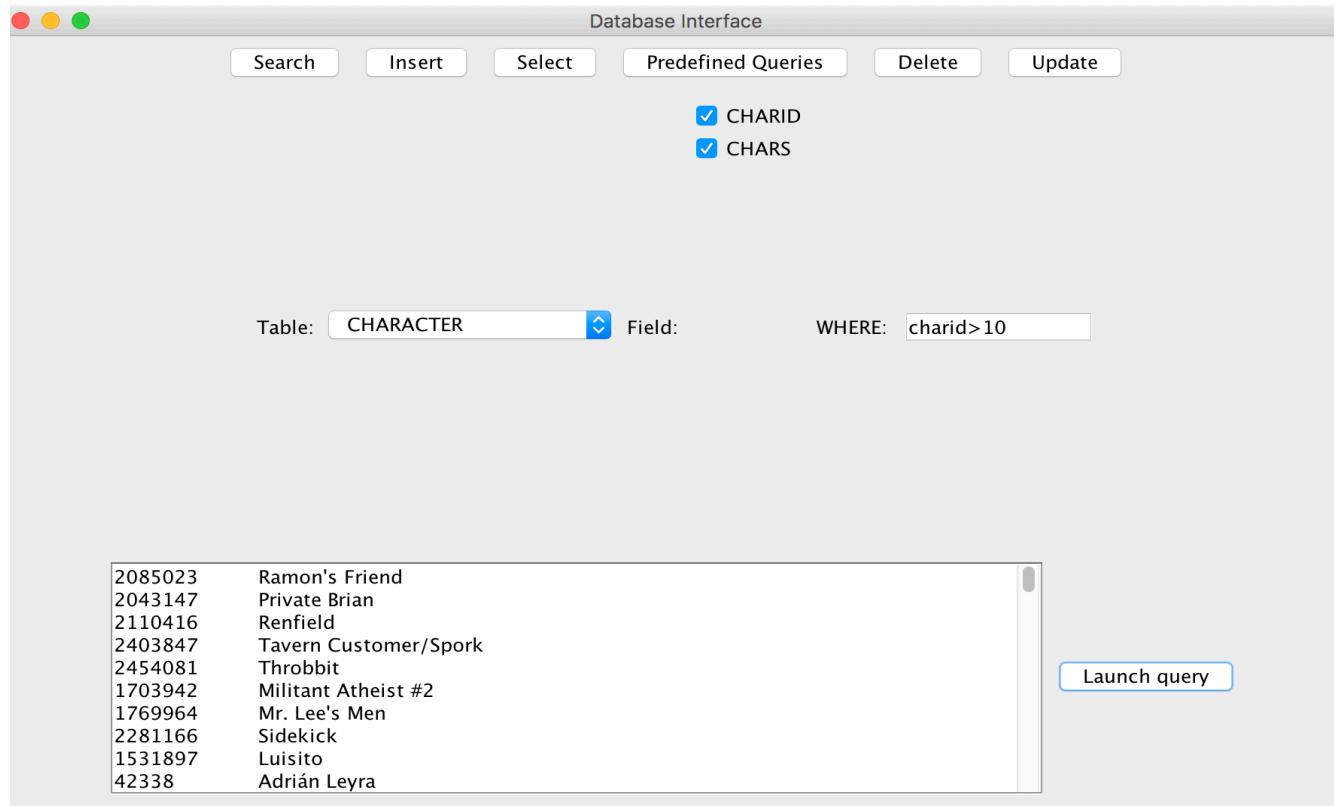
## 2) Insert

Here, for example, we insert a new row in the table CHARACTER with charid = 3244556 and chars = Kevin

The screenshot shows a Mac OS X-style window titled "Database Interface". At the top, there are buttons for Search, Insert, Select, Predefined Queries, Delete, and Update. A dropdown menu labeled "Table: CHARACTER" is open, and a dropdown menu labeled "Field: CHARID" is open, showing the value "3244556". Another dropdown menu labeled "Field: CHARS" is open, showing the value "Kevin". To the right of these dropdowns is an "Insert" button.

### 3) Select

Here, we select all the rows in character that satisfy the condition:



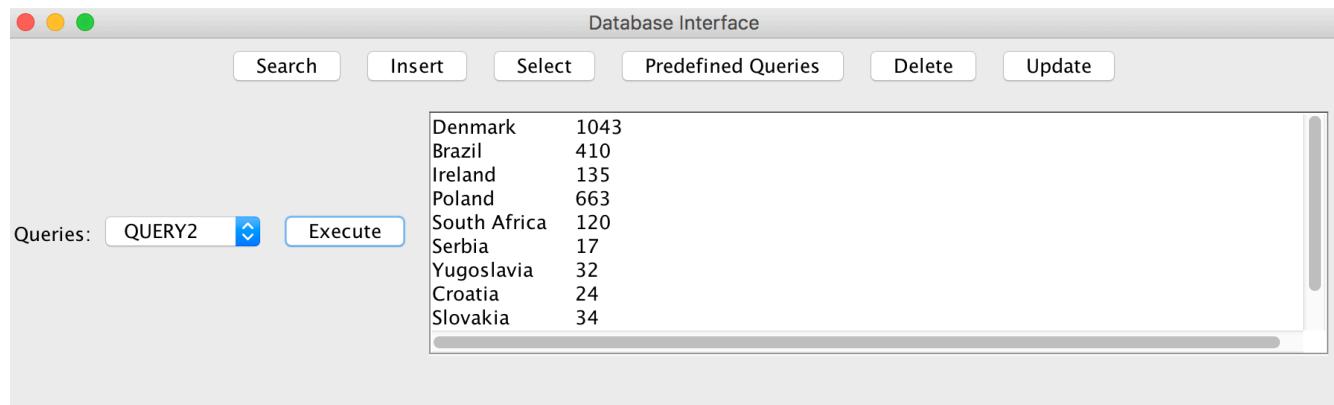
The screenshot shows a "Database Interface" window. At the top, there are buttons for Search, Insert, Select, Predefined Queries, Delete, and Update. Underneath these are two checked checkboxes: CHARID and CHARS. Below the checkboxes, there are dropdown menus for "Table: CHARACTER" and "Field:", and a "WHERE:" field containing "charid>10". A large scrollable table displays the following data:

2085023	Ramon's Friend
2043147	Private Brian
2110416	Renfield
2403847	Tavern Customer/Spork
2454081	Throbbit
1703942	Militant Atheist #2
1769964	Mr. Lee's Men
2281166	Sidekick
1531897	Luisito
42338	Adrián Leyra

A "Launch query" button is located to the right of the table.

### 4) Predefined queries:

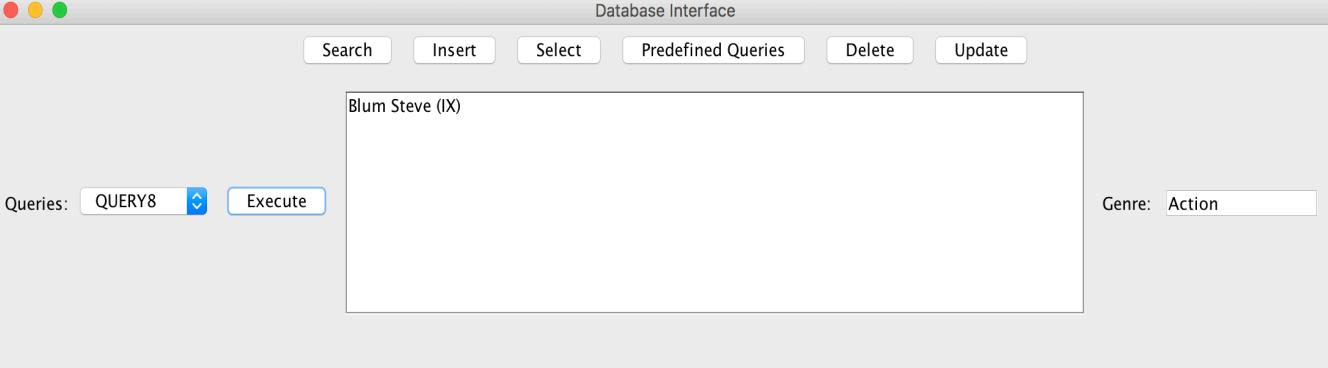
We can see below the result of query b from milestone 2



The screenshot shows a "Database Interface" window. At the top, there are buttons for Search, Insert, Select, Predefined Queries, Delete, and Update. Underneath these are dropdown menus for "Queries: QUERY2" and a "Execute" button. A scrollable table displays the following data:

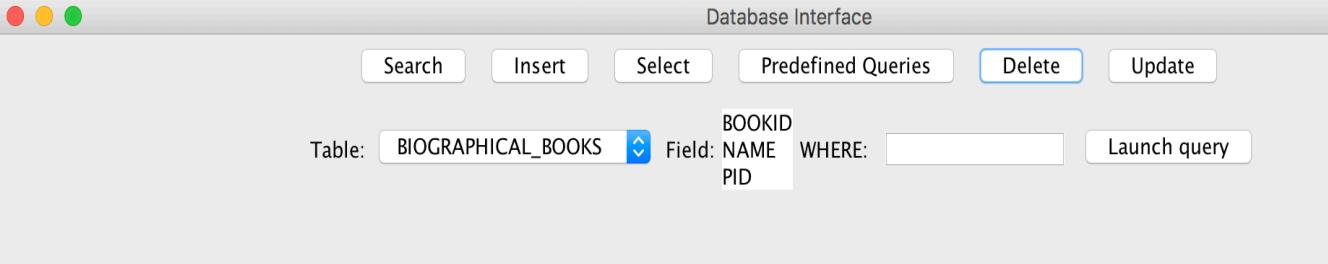
Denmark	1043
Brazil	410
Ireland	135
Poland	663
South Africa	120
Serbia	17
Yugoslavia	32
Croatia	24
Slovakia	34

And here we know that query h from milestone 2 requires a parameter (clip genre) entered by the user. Below we can see of the query for genre = Action



The screenshot shows a 'Database Interface' window. At the top, there are buttons for Search, Insert, Select, Predefined Queries, Delete, and Update. Below these, a search result box displays 'Blum Steve (IX)'. To the left, a 'Queries:' dropdown menu is set to 'QUERY8' with an 'Execute' button next to it. To the right, a parameter entry field shows 'Genre: Action'.

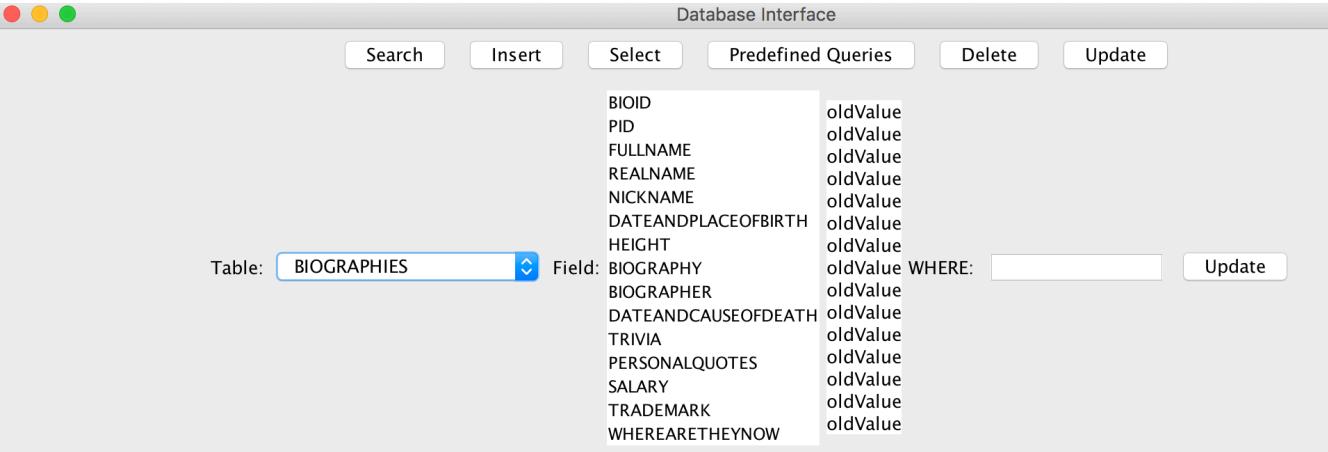
## 5) Delete



The screenshot shows a 'Database Interface' window with the 'Delete' tab selected. At the top, there are buttons for Search, Insert, Select, Predefined Queries, Delete, and Update. Below these, a search interface shows 'Table: BIOGRAPHICAL\_BOOKS' and 'Field: NAME WHERE: PID'. There is also a 'Launch query' button.

## 6) Update

Here we can see that we can update the values of all the rows that satisfy a certain condition. The values of the attributes can stay the same or we can modify the ones we want by replacing oldValue by a new one.



The screenshot shows a 'Database Interface' window with the 'Update' tab selected. At the top, there are buttons for Search, Insert, Select, Predefined Queries, Delete, and Update. Below these, a search interface shows 'Table: BIOGRAPHIES' and 'Field: BIOGRAPHY WHERE: oldValue'. A list of fields and their current values ('oldValue') includes: BIOID, PID, FULLNAME, REALNAME, NICKNAME, DATEANDPLACEOFBIRTH, HEIGHT, BIOGRAPHER, DATEANDCAUSEOFDEATH, TRIVIA, PERSONALQUOTES, SALARY, TRADEMARK, and WHEREARETHEYNOW.