Database Project - Deliverable 3

Guillaume Leclerc, Pedro Amorim, Matvey Khokhlov

May 2015

Chapter 1

Design Document

We decided to put the relation in file PRODUCTION_CAST.csv into one single relationship that links the Production, the Person and the Character if there is one.

We decided to remove the actress role as this information is already present in Person.gender. This type of diagram is not expressive enough to specify the participation and key constraints for ternary relationships, so we have to say it here.

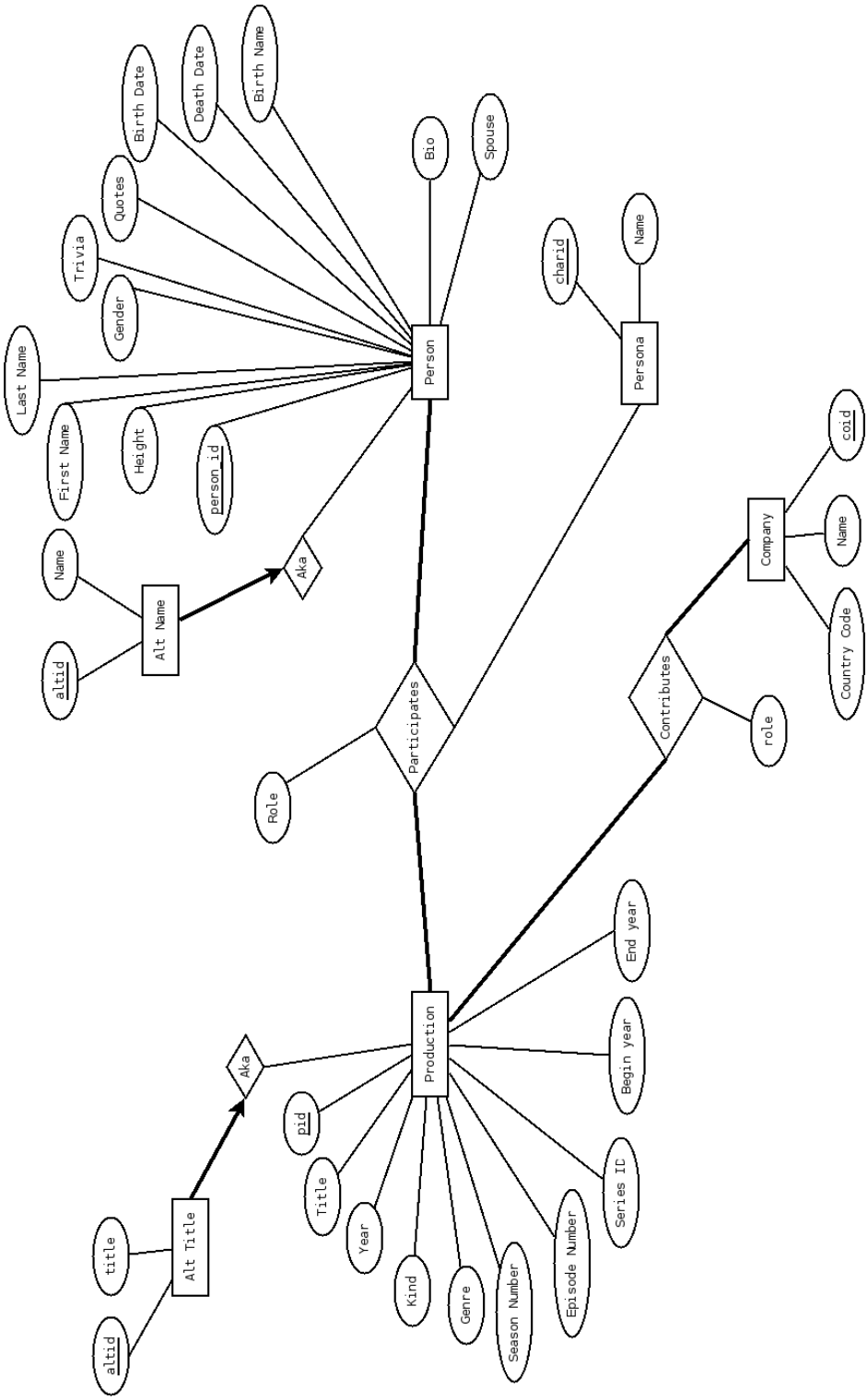
- Persona has a total participation with Production and Person
- Person has a total participation with Production because every person is either an actor or plays another role in the production.
- Production has a partial participation with the two others entity sets because some of them do not have actors or characters (eg. "Baraka")
- Production $\rightarrow$ Person is a one-to-many relationship
- Production $\rightarrow$ Persona is a one-to-many relationship
- Production $\rightarrow$ Persona is a one-to-many relationship

We decided to represent Episode and TV Series as subclasses of the Production entity. This allows us to have attributes specific to the subclasses while retaining the common attributes of a Production. An episode cannot be a TV Serie thus this is a Non-overlapping ISA relation but there is no covering constraint because a Production might neither be an Episode nor a TV Serie.

Chapter 2

ER Diagram

See next page (full diagram)


Person
Birth Name
Death Date
Birth Date
Quotes
Trivia
Gender
Last Name
First Name
Height
person_id
Bio
Spouse
Aka
Alt Name
Name
altid
Persona
charid
Name
Participates
Role
Contributes
role
Company
coid
Name
Country Code
Production
Aka
Alt Title
title
altid
pid
Title
Year
Kind
Genre
Season Number
Episode Number
Series IC
Begin year
End year

Chapter 3

DDL SQL code

```
-- DROP TABLES

DROP TABLE IF EXISTS ActorPlaysProduction;
DROP TABLE IF EXISTS PersonParticipatesProduction;
DROP TABLE IF EXISTS Persona;
DROP TABLE IF EXISTS CompanyContributesProduction;
DROP TABLE IF EXISTS Company;
DROP TABLE IF EXISTS Episode;
DROP TABLE IF EXISTS TV_Series;
DROP TABLE IF EXISTS ProductionAltName;
DROP TABLE IF EXISTS Production;
DROP TABLE IF EXISTS PersonAltName;
DROP TABLE IF EXISTS Person;

-- CREATE TABLES

CREATE TABLE Person
    (pid INTEGER NOT NULL AUTO_INCREMENT,
    first_name CHAR(100) NULL,
    last_name CHAR(127) NOT NULL,
    gender BOOLEAN NOT NULL,
    trivia TEXT NULL,
    quotes TEXT NULL,
    birth_date DATE NULL,
    death_date DATE NULL,
    birth_name CHAR(255) NULL,
    bio TEXT NULL,
    spouse CHAR(128) NULL,
    height FLOAT NULL,
   PRIMARY KEY (pid));
```

```
CREATE TABLE PersonAltName
    (altid INTEGER NOT NULL AUTO_INCREMENT,
    name CHAR(255) NOT NULL,
        pid INTEGER NOT NULL,
    PRIMARY KEY (altid),
    FOREIGN KEY (pid)
        REFERENCES Person(pid));

CREATE TABLE Production
    (prodid INTEGER NOT NULL AUTO_INCREMENT,
    title CHAR(255) NOT NULL,
    year INTEGER NULL,
    seriesid INTEGER NULL,
    seasonnum INTEGER NULL,
    epnum INTEGER NULL,
    beginyear INTEGER NULL,
    endyear INTEGER NULL,
    kind ENUM('tv_series', 'tv_movie', 'episode', 'movie',
                'video_movie', 'video_game') NOT NULL,
    genre ENUM('Action', 'Adventure', 'Animation', 'Biography',
                'Comedy', 'Crime', 'Documentary', 'Drama', 'Family',
                'Fantasy', 'Film-Noir', 'Game-Show', 'History',
                'Horror', 'Music', 'Musical', 'Mystery', 'News',
                'Reality-TV', 'Romance', 'Sci-Fi', 'Short', 'Sport',
                'Talk-Show', 'Thriller', 'War', 'Western') NULL,
    PRIMARY KEY (prodid));

CREATE TABLE ProductionAltName
    (altid INTEGER NOT NULL AUTO_INCREMENT,
    title CHAR(255) NOT NULL,
        prodid INTEGER NOT NULL,
    PRIMARY KEY (altid),
    FOREIGN KEY (prodid)
        REFERENCES Production(prodid));

CREATE TABLE Company
    (coid INTEGER NOT NULL AUTO_INCREMENT,
    country_code CHAR(4) NULL,
    name CHAR(255) NOT NULL,
    PRIMARY KEY (coid));

CREATE TABLE CompanyContributesProduction
    (prodid INTEGER NOT NULL,
    coid INTEGER NOT NULL,
        role ENUM('distributors', 'production_companies') NOT NULL,
```

```
    PRIMARY KEY (prodid, coid, role),
    FOREIGN KEY (prodid) REFERENCES Production(prodid),
    FOREIGN KEY (coid) REFERENCES Company(coid));

CREATE TABLE Persona
    (charid INTEGER NOT NULL,
    name CHAR(100) NOT NULL,
        PRIMARY KEY (charid));

CREATE TABLE PersonParticipatesProduction
        (pppid BIGINT NOT NULL AUTO_INCREMENT,
        pid INTEGER NOT NULL,
        prodid INTEGER NOT NULL,
        role ENUM('actor', 'actress', 'producer', 'writer', 'cinematographer', '
                'costume designer', 'director', 'editor',
                'miscellaneous crew', 'production designer') NOT NULL,
        charid INTEGER NULL,
        PRIMARY KEY (pppid),
        FOREIGN KEY (pid) REFERENCES Person(pid),
        FOREIGN KEY (charid) REFERENCES Persona(charid),
        FOREIGN KEY (prodid) REFERENCES Production(prodid));
```

Chapter 4

SQL Queries for Milestone 2

```
SELECT year, COUNT(*) FROM Production WHERE kind NOT IN ('episode', 'tv_series',
SELECT country_code as country, COUNT(*) as number from Company WHERE NOT isNull
ORDER BY number DESC LIMIT 10;
SELECT MIN(duration) as min, MAX(duration) as max, AVG(duration) as average from

-- compute the min, max and average number of actors in a production
SELECT MIN(c) as min, MAX(c) as max, AVG(c) as avg FROM
    (SELECT DISTINCT pid, prodid, COUNT(*) as c FROM PersonParticipatesProductio

-- compute the min, max and average height of female persons
SELECT MIN(height), MAX(height), AVG(height) FROM Person WHERE gender = 0;

--f) Get all the actors that are also directors in a production, excluding tv m
SELECT Person.first_name, Person.last_name, Production.title
FROM PersonParticipatesProduction Pers1
    JOIN PersonParticipatesProduction Pers2
        ON (Pers1.pid = Pers2.pid AND Pers1.prodid = Pers2.prodid
            AND Pers1.role IN ('actor', 'actress') AND Pers2.role = 'director')
        JOIN Production ON (Production.prodid = Pers1.prodid
        AND Production.kind = 'movie')
        JOIN Person ON (Pers1.pid = Person.pid);

--g) Get the three most popular character names
SELECT Persona.name, appearances FROM Persona JOIN (SELECT DISTINCT charid, coun
```

Chapter 5

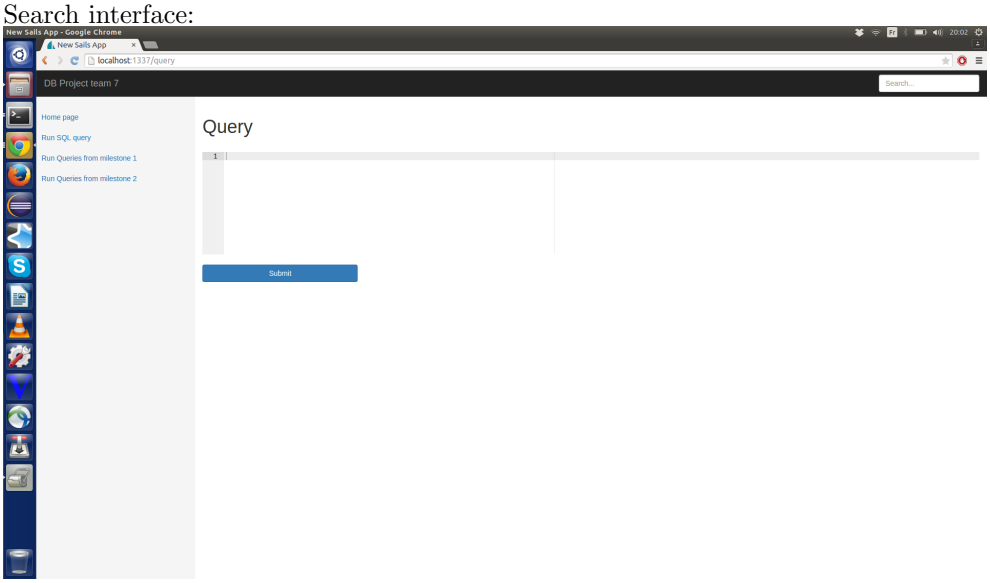
Basic interface for Milestone 2

We parsed the database and implemented a basic interface to run the simple search queries for milestone 2. We have done this using the Sails.js framework. Screenshots on the pages that follow.

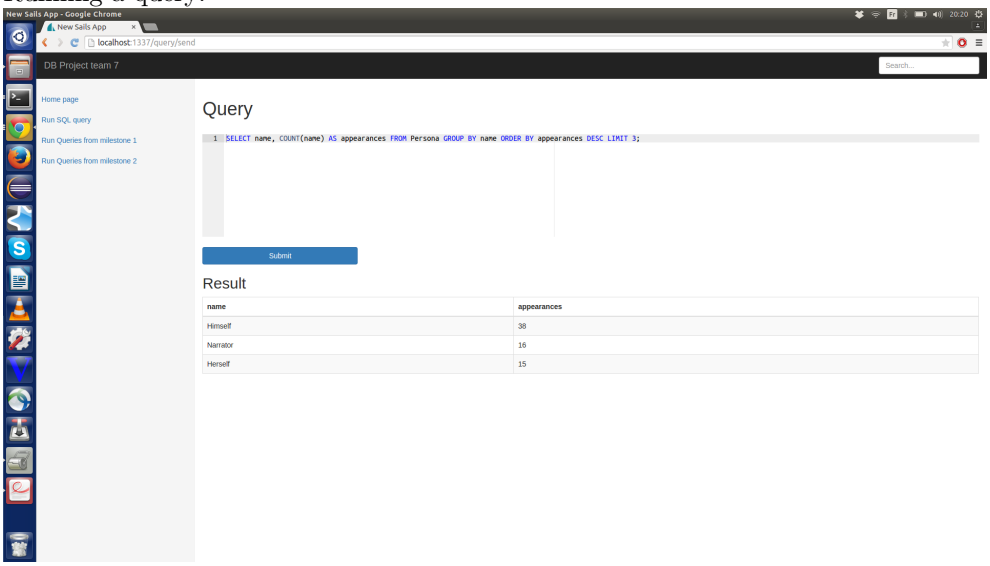
Homepage:


New Sails App - Google Chrome
New Sails App
localhost:1337
DB Project team 7
Search...
Home page
Run SQL query
Run Queries from milestone 1
Run Queries from milestone 2
Semester project in DBMS
Group 7
Guillaume Leclerc
Pedro Amorim Caldeira
Matvey Khokhlov
Use the navbar to explore the project.

Search interface:



Running a query:



Chapter 6

SQL Queries for Milestone 2

```
-- f) Compute the average number of episodes per season
-- (7,25 sec)
SELECT seasonnum, AVG(c) FROM
    (SELECT seasonnum, COUNT(*) AS c FROM Production
    WHERE kind LIKE 'episode'
    AND seasonnum IS NOT NULL GROUP BY seriesid, seasonnum) AS T
GROUP BY seasonnum;

-- g) Compute the average number of seasons per series
-- (6,83 sec)
SELECT AVG(c) FROM
    (SELECT COUNT(*) as c FROM
            (SELECT DISTINCT seriesid, seasonnum FROM Production
            WHERE kind LIKE 'episode' AND seasonnum IS NOT NULL) AS T
    GROUP BY seriesid) as G;

-- h) Compute the top ten tv-series (by number of seasons)
-- (6,24 sec)
SELECT title, c
    FROM (SELECT seriesid as id, COUNT(*) as c FROM
        (SELECT DISTINCT seriesid, seasonnum FROM Production
        WHERE kind LIKE 'episode' AND seasonnum IS NOT NULL) AS T
    GROUP BY seriesid ORDER BY c DESC LIMIT 10) as G
JOIN Production as P2 ON id = P2.prodid;

-- i) Compute the top ten tv-series (by number of episodes per season)
-- (6,04 sec)
SELECT title, c FROM
```

```
    (SELECT seriesid as id, COUNT(*) AS c FROM Production
    WHERE kind LIKE 'episode'
    AND seasonnum IS NOT NULL
    GROUP BY seriesid, seasonnum ORDER BY c DESC LIMIT 10) as G
JOIN Production as P2 ON id = P2.prodid;

-- j) Find actors, actresses and directors who have movies (incl. tv and video
-- movies) released after their death (14,06 sec) without index
SELECT DISTINCT first_name, last_name FROM Production
NATURAL JOIN PersonParticipatesProduction
NATURAL JOIN Person WHERE death_date IS NOT NULL
AND YEAR(death_date) < Production.year
AND role IN ('director', 'actor', 'actress')
AND kind IN ('tv_movie', 'movie', 'video_movie');
```

Chapter 7

Analysis of Queries

7.1 Query f)

7.1.1 Query Description

Find actors, actresses and directors who have movies released after their death

7.1.2 Query Code

```
SELECT DISTINCT first_name, last_name
FROM "Production" NATURAL JOIN "PersonParticipatesProduction"
NATURAL JOIN "Person"
WHERE death_date IS NOT NULL
AND EXTRACT(YEAR FROM death_date) < year
AND role IN ('direc\tor', 'actor','actress')
AND kind IN ('tv movie', 'movie', 'video movie');
```

7.1.3 Analysis

Query Plan without Index

```
HashAggregate  (cost=1612225.26..1612266.07 rows=4081 width=230)
  ->  Nested Loop  (cost=227014.93..1612103.20 rows=24411 width=230)
    Join Filter: ("Person".death_date) < ("Production".year)::double precision)
    ->  Hash Join  (cost=227014.50..1504355.88 rows=217825 width=238)
          Hash Cond: ("PersonParticipatesProduction".pid = "Person".pid)
          ->  Seq Scan on "PersonParticipatesProduction"  (cost=0.00..886189.22 rows=25931594 width=8)
                Filter: (role = ANY ('{director,actor,actress}'::personparticipatesproduction_role[]))
          ->  Hash  (cost=226504.44..226504.44 rows=40805 width=238)
                ->  Seq Scan on "Person"  (cost=0.00..226504.44 rows=40805 width=238)
                      Filter: (death_date IS NOT NULL)
    ->  Index Scan using "Production_pkey" on "Production"  (cost=0.43..0.47 rows=1 width=8)
          Index Cond: (prodid = "PersonParticipatesProduction".prodid)
          Filter: (kind = ANY ('{"tv movie",movie,"video movie"}'::production_kind[]))

Time: 17195,901 ms
```

The biggest cost in the query execution comes from the nested loop joining the PersonParticipatesProduction and Person table with the Production table. We can see that the optimizer of the database management system correctly

inferred that the condition about the death date should apply during the join and not as a filter after the full table join. Given that the Production table is joined with the help of the primary key the nested loop only needs to be run once and thus there should be no way of optimizing it.

Sequential scans are often places where indexes can offer performance gains and we can see that the database management system decided to do a sequential scan on PersonParticipatesProduction to find all the tuples with a director, actress and actor role. This indicates that an index on the role column could be a good candidate on optimizing this query. There is another sequential scan on the death date column so an index could potentially speed up the whole query.

Here is the query plan with an index on role from PersonParticipatesProduction and an index on the death date of the person relation.

Query Plan with Indexes

```
HashAggregate  (cost=1279454.20..1279497.44 rows=4324 width=230)
  ->  Nested Loop  (cost=102159.51..1279326.65 rows=25511 width=230)
      Join Filter: ("Person".death_date) < ("Production".year)::double precision)
      ->  Hash Join  (cost=102159.08..1165643.26 rows=229783 width=238)
            Hash Cond: ("PersonParticipatesProduction".pid = "Person".pid)
            ->  Index Scan using "PersonParticipatesProduction_role_idx" on "PersonParticipatesProduction"
(cost=0.57..835274.84 rows=25818523 width=8)
                  Index Cond: (role = ANY ('{director,actor,actress}'::personparticipatesproduction_role[]))
            ->  Hash  (cost=101618.05..101618.05 rows=43237 width=238)
                  ->  Bitmap Heap Scan on "Person"  (cost=811.52..101618.05 rows=43237 width=238)
                        Recheck Cond: (death_date IS NOT NULL)
                        ->  Bitmap Index Scan on "Person_death_date_idx"  (cost=0.00..800.71 rows=43237 width=0)
                              Index Cond: (death_date IS NOT NULL)
      ->  Index Scan using "Production_pkey" on "Production"  (cost=0.43..0.47 rows=1 width=8)
            Index Cond: (prodid = "PersonParticipatesProduction".prodid)
            Filter: (kind = ANY ('{"tv movie",movie,"video movie"}'::production_kind[]))

Time: 15541,305 ms
```

As we can see the execution time was improved by about 1600 ms indicating and the indexes are effectively being used indicating that they were effective in improving the execution time of the query.