Projecto de Bases de Dados (CC2005) - parte 2

1. Elementos do grupo

Grupo nº 14

Nº mecanográfico	Nome
202203947	Gonçalo Martins Esteves
202206195	Nuno Santiago Ribeiro Gomes

2. Ajustes ao modelo da BD

Em suma, no modelo de classes UML, removeu-se a classe-associação *Person_Job* e criou-se uma associação ternária entre as classes *Show*, *Person* e *Job*.

Estas representam, respectivamente:
→ Todos os filmes e séries na BD e os seus atributos;
→ Todos os indivíduos mencionados na BD;
→ Todos os trabalhos que um indivíduo pode realizar num *show* (actor ou director).

Para além disso, o modelo relacional foi modificado de modo a representar correctamente estas mudanças, criando-se uma tabela “Show_Person_Job” que é capaz de representar esta nova relação ternária.

3. Povoamento de tabelas

Os dados utilizados no povoamento das tabelas recaíram no dataset indicado no relatório da 1ª Parte do Projeto. Mais ainda, dado o dataset constituir uma única tabela com toda a informação, foi necessário adaptá-lo de forma a que os dados fossem distribuídos pelas múltiplas tabelas (referenciadas no modelo relacional).

Assim, através tanto do *Python* como do *Jupyter Notebook*, foi-nos possível criar um script capaz de gerar a base de dados posteriormente utilizada na Aplicação Web. Claro é, que o script vai ser disponibilizado juntamente com o *source code* da aplicação. Neste é possível encontrar não só a forma como se manipulou o Dataset de forma a obter as várias tabelas mencionadas no modelo relacional (através de queries em SQL e o consequente povoamento das mesmas) que constituíram a nossa Base de Dados.

Nome da tabela	**Nº de entradas**
Show	5331
Rating	14
Type	2
Country	109
Genre	42
Person	29337
Job	2
Show_Genre	11855
Show_Country	6876
Show_Person_Job	48659

4. Aplicação Python

“Endpoint”	**Funcionalidade**
/	Página de entrada
/genres/	Dispõe todos os géneros existentes na BD
/genres/<int:id>	Dispõe os *shows* (filmes e séries) com um determinado género
/countries/	Dispõe todos os países existentes na BD
/countries/<int:id>	Dispõe os *shows* (filmes e séries) produzidos num determinado país
/ratings/	Dispõe todos os *ratings* existentes na BD
/ratings/<int:id>	Dispõe os *shows* (filmes e séries) com um determinado rating
/movies/P<int:page_number>	Dispõe todos os filmes existentes na base de dados através de um sistema de paginação
/movies/<int:id>	Dispõe todo o *staff* (actores e directores) envolvido num determinado filme bem como as especificações do mesmo
/tvshows/	Dispõe todas as séries existentes na base de dados
/tvshows/<int:id>	Dispõe todo o *staff* (actores e directores) envolvido numa determinada série
/actors/P<int:page_number>	Dispõe de todos os actores existentes na base de dados através de um sistema de paginação
/actors/<int:id>	Dispõe todos os *shows* (filmes e séries) em que cada ator participou
/directors/	Dispõe de todos os directores existentes na base de dados
/directors/<int:id>	Dispõe todos os *shows* (filmes e séries) em que cada director participou
/search/<string:input_search>	Dispõe todos os filmes, séries, actores e directores aos quais corresponde um determinado input

```
# Endpoint: '/genres/'
# LIST ALL GENRES
def list_genres():
genres = db.execute(
    '''
    SELECT g.genre_id, g.name
    FROM Genre g
    ORDER BY g.name;
    '''
  ).fetchall()
```

A query acima descrita irá devolver todos os géneros existentes na base de dados, estando estes ordenados por ordem alfabética.

```
# Endpoint: '/genres/<int:id>'
def get_shows_genre(id):
  genre = db.execute(
    '''
    SELECT g.name
    FROM Genre g
    WHERE g.genre_id = ?
    ''', [id]).fetchone()

  movies_genre = db.execute(
    '''
    SELECT s.show_id, s.title, s.release_year, s.duration
    FROM Show s JOIN Show_Genre sg JOIN Genre g JOIN Type t
    ON (s.show_id = sg.show_id AND sg.genre_id = g.genre_id AND s.type_id =
t.type_id)
    WHERE t.type = 'Movie' AND g.genre_id = ?
    ORDER BY s.title;
    ''', [id]).fetchall()

  tvshows_genre = db.execute(
    '''
    SELECT s.show_id, s.title, s.release_year, s.duration
    FROM Show s JOIN Show_Genre sg JOIN Genre g JOIN Type t
    ON (s.show_id = sg.show_id AND sg.genre_id = g.genre_id AND s.type_id =
t.type_id)
    WHERE t.type = 'TV Show' AND g.genre_id = ?
    ORDER BY s.title;
    ''', [id]).fetchall()
```

Mais ainda, de forma a ser possível mostrar todos os shows que se enquadrem num determinado género, foram utilizadas múltiplas queries. Inicialmente, é procurado o nome do género ao qual corresponde o número disponibilizado como input.

Posteriormente, as queries *movies_genre* e *tvshows_genre* irão recolher todos os shows (filmes e séries respectivamente) que contenham este género. Para tal, recorre-se à junção das tabelas Show, Genre e *Show_Genre* com o intuito de obter os IDs, títulos, anos de estreia e durações dos shows que contenham o género selecionado.

```
# Endpoint: '/countries/'
def list_countries():
  countries = db.execute(
    '''
    SELECT c.country_id, c.name
    FROM Country c
    ORDER BY c.name;
    ''').fetchall()
```

A query que se encontra descrita no interior da função *list_countries* permite listar todos os países (ID e nome) sob os quais foi possível produzir pelo menos um dos shows existentes na base de dados.

```
# Endpoint: '/countries/<int:id>'
def get_shows_countries(id):
  country = db.execute(
    '''
    SELECT c.name
    FROM Country c
    WHERE c.country_id = ?;
    ''', [id]).fetchone()

  movies_countries = db.execute(
    '''
    SELECT s.show_id, s.title, s.release_year, s.duration
    FROM SHOW s JOIN Country c JOIN Show_Country sc JOIN Type t
    ON (s.show_id=sc.show_id AND sc.country_id=c.country_id AND
s.type_id=t.type_id)
    WHERE c.country_id = ? AND t.type = 'Movie'
    ORDER BY s.title;
    ''', [id]).fetchall()

  tvshows_countries = db.execute(
    '''
    SELECT s.show_id, s.title, s.release_year, s.duration
    FROM SHOW s JOIN Country c JOIN Show_Country sc JOIN Type t
    ON (s.show_id=sc.show_id AND sc.country_id=c.country_id AND
s.type_id=t.type_id)
    WHERE c.country_id = ? AND t.type = 'TV Show'
    ORDER BY s.title;''', [id]).fetchall()
```

Analogamente ao caso disposto anteriormente relativo à listagem dos múltiplos shows que contivessem um determinado género, a função *get_shows_countries* recorre a várias queries de forma a ser possível listar todos os shows que foram produzidos num determinado país a partir do ID do mesmo. Inicialmente, é procurado o nome do país que se pretende analisar. Deste modo, as queries finais procuram, através da junção das tabelas *Show*, *Country* e *Show_Country* (e posterior filtragem da informação), recolher as informações dos *shows* (ID, título, ano de lançamento, duração) que foram produzidos no país em análise.

```
# Endpoint: '/ratings/'
def list_ratings():
  ratings = db.execute(
    '''
    SELECT r.rating_id, r.acronym, r.description
    FROM Rating r
    ORDER BY r.rating_id;
    ''').fetchall()
```

A query utilizada na função *list_ratings* procura, somente recolher os IDs, acrónimos e descrições de todos os ratings existentes na base de dados, ordenando-os por ordem alfabética.

```
# Endpoint: '/ratings/<int:id>'
def get_shows_ratings(id):

  rating_acr = db.execute(
    '''
    SELECT r.acronym
    FROM Rating r
    WHERE r.rating_id = ?
    ''', [id]).fetchone()

  movies_rating = db.execute(
    '''
    SELECT s.show_id, s.title, s.release_year, s.duration
    FROM Show s JOIN Type t
    ON (s.type_id = t.type_id)
    WHERE s.rating_id = ? AND t.type = 'Movie'
    ORDER BY s.title;
    ''', [id]).fetchall()

  tvshows_rating = db.execute(
    '''
    SELECT s.show_id, s.title, s.release_year, s.duration
    FROM Show s JOIN Type t
```

```
    ON (s.type_id = t.type_id)
    WHERE s.rating_id = ? AND t.type = 'TV Show'
    ORDER BY s.title;
    ''', [id]).fetchall()
```

Analogamente ao caso disposto anteriormente relativo à listagem dos múltiplos shows que contivessem um determinado género, a função *get_shows_ratings* recorre a várias queries de forma a ser possível listar todos os shows a que foi atribuído um determinado rating a partir do ID do mesmo. Inicialmente, é procurado o acrónimo do rating que se pretende analisar. Deste modo, as queries finais procuram, através da junção das tabelas *Show* e *Rating* (e posterior filtragem da informação), recolher as informações dos *shows* (ID, título, ano de lançamento, duração) que têm o rating em causa.

```
# Endpoint: '/movies/P<int:page_number>'
def list_movies(page_number):
  size = 1000
  start_limit = (page_number - 1) * size
  end_limit = (page_number) * size

  movies = db.execute(
    '''
    SELECT s.show_id, s.title, s.release_year, s.duration
    FROM Show s NATURAL JOIN Type t
    WHERE t.type = 'Movie'
    ORDER by s.title
    LIMIT ?, ?;
    ''', [start_limit, end_limit]).fetchall()
```

A query que se encontra descrita no interior da função *list_movies* permite listar todos os filmes existentes na base de dados (incluindo ID, Título, Ano de Lançamento e Duração). Para além disso, os dados recolhidos nesta query são agrupados em páginas com 1000 filmes cada.

```
# Endpoint: '/movies/<int:id>'
def get_movie(id):
  movie = db.execute('''
  SELECT s.show_id, s.title, s.release_year, s.duration, s.description
  FROM Show s NATURAL JOIN Type t
  WHERE t.type = 'Movie' AND s.show_id = ?
  ORDER by s.show_id;
  ''', [id]).fetchone()

  rating = db.execute('''
    SELECT r.acronym
    FROM Show s JOIN Rating r
```

```
    ON (s.rating_id=r.rating_id)
    WHERE s.show_id = ?
    ''', [id]).fetchone()

  genres = db.execute('''
  SELECT g.name
  FROM Show s JOIN Show_Genre sg JOIN Genre g
  ON (s.show_id = sg.show_id AND sg.genre_id=g.genre_id)
  WHERE s.show_id = ?
  ORDER BY g.name;
  ''', [id]).fetchall()

  countries = db.execute('''
  SELECT c.name
  FROM Show s JOIN Show_Country sc JOIN Country c
  ON (s.show_id = sc.show_id AND sc.country_id=c.country_id)
  WHERE s.show_id = ?
  ORDER BY c.name;
  ''', [id]).fetchall()

  actors = db.execute('''
  SELECT p.person_id, p.name
  FROM Show_Person_Job spj JOIN Job j JOIN Person p JOIN Show s JOIN Type t
  ON (spj.person_id = p.person_id AND spj.job_id = j.job_id AND spj.show_id =
s.show_id AND s.type_id = t.type_id)
  WHERE t.type = 'Movie' AND spj.show_id = ? AND j.name = 'cast'
  ORDER by p.name;
  ''', [id]).fetchall()

  directors = db.execute('''
  SELECT p.person_id, p.name
  FROM Show_Person_Job spj JOIN Job j JOIN Person p JOIN Show s JOIN Type t
  ON (spj.person_id = p.person_id AND spj.job_id = j.job_id AND spj.show_id =
s.show_id AND s.type_id = t.type_id)
  WHERE t.type = 'Movie' AND spj.show_id = ? AND j.name = 'director'
  ORDER by p.name;
  ''', [id]).fetchall()
```

As queries evidentes neste pedaço de código vão listar vários elementos acerca dum único filme:

- O ID, Título, Ano de Lançamento, Duração e Descrição do filme em *movie*
- O Acrónimo do rating de tal filme em *rating*
- O(s) Género(s) do filme em questão em *genres*
- O(s) País(es) que participaram na produção desse filme em *countries*
- O(s) Actor(es) que tomaram parte no filme em *actors*
- O(s) Director(es) que tomaram parte no filme em *directors*

→ Analogamente foi feito o mesmo para os TV Shows, alterando somente o *type* do *Show* nas queries

```
# Endpoint: '/actors/P<int:page_number>'
def list_actors(page_number):
  size = 4400
  start_limit = (page_number - 1) * size
  end_limit = (page_number) * size

  actors = db.execute('''
  SELECT DISTINCT p.person_id, p.name
  FROM Show_Person_Job spj JOIN Person p JOIN Job j
  ON (spj.person_id = p.person_id AND spj.job_id = j.job_id)
  WHERE j.name = 'cast'
  ORDER by p.name
  LIMIT ?, ?;
  ''', [start_limit, end_limit]).fetchall()
```

A query anterior, descrita no interior da função *list_actors* permite listar todos os actores existentes na BD (incluindo ID e Nome). Tal como em *list_movies*, os dados recolhidos nesta query são agrupados em páginas com 4400 actores cada.

```
# Endpoint: '/actors/<int:id>'
def get_actor(id):
  actor = db.execute('''
  SELECT DISTINCT p.person_id, p.name
  FROM Show_Person_Job spj JOIN Person p JOIN Job j
  ON (spj.person_id=p.person_id AND spj.job_id=j.job_id)
  WHERE j.name = 'cast' AND p.person_id = ?
  ORDER by p.name;
  ''', [id]).fetchone()

  productions_movies = db.execute('''
  SELECT DISTINCT s.show_id, s.title, s.release_year, s.duration
  FROM Show s JOIN Type t JOIN Show_Person_Job spj JOIN Job j JOIN Person p
  ON (spj.show_id = s.show_id AND spj.person_id = p.person_id AND spj.job_id =
j.job_id AND s.type_id = t.type_id)
  WHERE p.person_id = ? AND t.type = 'Movie'
  ORDER by s.title;
  ''', [id]).fetchall()

  productions_tvshows = db.execute('''
  SELECT DISTINCT s.show_id, s.title, s.release_year, s.duration
  FROM Show s JOIN Type t JOIN Show_Person_Job spj JOIN Job j JOIN Person p
  ON (spj.show_id = s.show_id AND spj.person_id = p.person_id AND spj.job_id =
j.job_id AND s.type_id = t.type_id)
```

```
  WHERE p.person_id = ? AND t.type = 'TV Show'
  ORDER by s.title;
  ''', [id]).fetchall()
```

As queries mostradas aqui recolhem várias informações acerca dum único actor:

- O ID e Nome do actor em *actor*
- Os IDs, Títulos, Anos de Lançamento e Durações dos vários filmes em que o actor participou em *productions_movies*
- Os IDs, Títulos, Anos de Lançamento e Durações das várias séries em que o actor tomou parte em *productions_tvshows*

→ Analogamente foram utilizadas queries para obter as informações dos diretores, mudando apenas o *job* da pessoa

```
# Endpoint: '/search/<string:input_search>'
def search(input_search):
  input_search_with_wildcards = f"%{input_search}%"

  # Query to Search the Movies
  movies_query = '''
          SELECT s.show_id, s.title, s.release_year, s.duration
          FROM Show s JOIN Type t on (s.type_id = t.type_id)
          WHERE s.title LIKE :search_term AND t.type='Movie'
          ORDER BY s.title;
          '''

  movies_results = db.execute(movies_query, {"search_term":
input_search_with_wildcards}).fetchall()

  # Query to Search the TV Shows
  tvshow_query = '''
          SELECT s.show_id, s.title, s.release_year, s.duration
          FROM Show s JOIN Type t on (s.type_id = t.type_id)
          WHERE s.title LIKE :search_term AND t.type='TV Show'
          ORDER BY s.title;
          '''

  tvshows_results = db.execute(tvshow_query, {"search_term":
input_search_with_wildcards}).fetchall()

  # Query to Search the Actors
  actor_query = '''
          SELECT DISTINCT p.person_id, p.name
          FROM Person p JOIN Show_Person_Job spj JOIN Job j
          ON (p.person_id=spj.person_id AND spj.job_id=j.job_id)
```

```
          WHERE p.name LIKE :search_term AND j.name='cast'
          ORDER BY p.name;'''

  actors_results = db.execute(actor_query, {"search_term":
input_search_with_wildcards}).fetchall()

  # Query to Search the Directors
  director_query = '''
          SELECT DISTINCT p.person_id, p.name
          FROM Person p JOIN Show_Person_Job spj JOIN Job j
          ON (p.person_id=spj.person_id AND spj.job_id=j.job_id)
          WHERE p.name LIKE :search_term AND j.name='director'
          ORDER BY p.name;'''

  directors_results = db.execute(director_query, {"search_term":
input_search_with_wildcards}).fetchall()
```

Nesta última query, o programa recebe input do utilizador e pesquisa na BD todos os filmes, séries, actores e directores que tenham no seu título/nome o input recebido.