Relatório Base de Dados

Industria Automóvel

PARTE 2

Cadeira: Bases de Dados (CC2005)
Docente: Eduardo Resende Brandão Marques

Projecto de Bases de Dados (CC2005) - parte 2

1. Elementos do grupo

Grupo nº G54

Nº mecanográfico	***Nome***
202108234	*Ana Cláudia Preto Batista*
202107858	*Alexandre Tomás Lusquinhos Fernandes Moreira Carneiro*
202105821	*Gonçalo Luís Garcias Monteiro*

2. Correção do modelo da BD

2.1. Correcção ao universo considerado e fontes de dados

1) O nome da Entidade-Tipo "SEGMENTS" foi substituída por "Body-type"; esta alteração não afeta os relacionamentos pois os "body-types" e "segments" são atributos muito parecidos de um carro;

2) A Chave-Primária de "GRUPOS" e "MAKES" passou a ser um "Id"; Group_Id e Make_Id;

2.2. Correcção de requisitos

1)A relação entre "MAKES" E "GRUPOS" mais detalhada:

Marcas('MAKES') pertencem a Grupos('GRUPOS');Um Grupo possui pelo menos um Líder (a liderança pode ser exercida por um grupo de Marcas, ou seja mais do que uma marca); Não há Marcas sem Grupo correspondente; Se um Grupo contém apenas uma marca participante, então o Grupo representa-se pela própria Marca sendo o seu líder a mesma marca; Se uma Marca não é Líder, então é obrigatoriamente Subsidiária de um Grupo;Um Grupo não necessita de Marcas Subsidiárias; A condição necessária para a existência de um Grupo, é a existência de uma Marca(ou grupo de Marcas) Líder;
Cada Marca pertence a um e um só Grupo; (1:N)

2) Foram acrescentados atributos "Cars";

3) O País(Country) de um Grupo é um multi-valor já que depende do país dos Lideres do Grupo

2.3. Correcção do modelo ER

1) A relação entre marca e grupo foi dividia em marcas que subsidiam um grupo e marcas que lideram um grupo:

LEADERS(Makes, Grupo)	*N<>1*	*Parcial<>Total*
SUBSIDIZE(Makes, Grupo)	*N<>1*	*Parcial<>Total*

2) Alteração no nome dos relacionamentos para(LEADERS, SUBSIDIZE, MANUFACTURE, DISTINGUISHED e BODY_TYPE_MAKES);

2.4. Correcção do modelo relacional

1)Foi acrescentada a tabela "CON";

Tabela "CON" não faz parte do esquema inicial. Porém, devido a algumas dificuldades no preenchimento da tabela "COUNTRIES" ,criar uma tabela auxiliar "CON" foi a única solução encontrada para a resolução do problema;

3. Povoamento de tabelas

Para o povoamento das tabelas usamos a seguinte base de dados:
https://database-downloads.com/product/free-download-car-database-1945-2020-metric-excel/
A partir desta conseguimos uma extensa lista de carros, assim como informações sobre os mesmos, que serviram principalmente para povoar a tabela 'cars'. Os dados das restantes tabelas surgiram de uma mistura de informações da base de dados com pesquisa de dados complementares. O povoamento das tabelas foi feito empregando LOAD DATA.

```
LOAD DATA INFILE 'C:\\Users\\alexa\\Desktop\\test\\BODYTYPE.csv'
INTO TABLE BODYTYPE
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
```

Nome da tabela	Número de entradas
BODYTYPE	17
BTMAKES	670
CARS	47174
CON	19
COUNTRIES	51
GRUPOS	48
LEADER	52
MAKES	131

4. Aplicação Python

"Endpoint"	*Funcionalidade*
/	*Página de entrada*
/cars/	*Lista de todos os cars e atributos associados.*
/cars/<int:code>/	*Página individual de cada car com os respetivos atributos.*
/cars/search/<expr>/	*Página com todos os cars que tem a expressão 'expr' no nome do modelo do carro (Model).*
/makes/	*Lista de todas as makes e atributos associados.*
/makes/<int:id>/	*Página individual de cada make com os respetivos atributos.*
/makes/search/<expr>/	*Página com todas as makes que tem a expressão 'expr' no nome (Make)*
/groups/	*Lista de todos os grupos e atributos associados.*
/groups/<int:id>/	*Página individual de cada grupo com os respetivos atributos.*
/bodytypes/	*Lista de todos os bodytypes e atributos associados.*
/bodytypes/<int:id>/	*Página individual de cada bodytype com os respetivos atributos.*

Endpoints

'/'

Código:

```
@APP.route('/')
def index():
        stats = {}
        x = db.execute('SELECT COUNT(*) AS Cars FROM cars').fetchone()
        stats.update(x)
        x = db.execute('SELECT COUNT(*) AS Makes FROM makes').fetchone()
        stats.update(x)
        x = db.execute('SELECT COUNT(*) AS Grupos FROM grupos').fetchone()
        stats.update(x)
        logging.info(stats)
        return render_template('index.html',stats=stats)
```

Em stats ficam guardadas o número de entradas das principais tabelas cars, makes e grupos.

'/cars/'

Código:

```
cars = db.execute(
        '''
        SELECT CarCode, Model, Generation, YearFrom, YearTo, Trim, MotorType, Cylinders,
EngineType
        FROM cars
        ORDER BY CarCode
        ''').fetchall()
```

- *Recolhe todos os dados que cada car fornece (excepto Make e Bodytype), ordenando-os por CarCode.*

'/cars/<int:code>/'

Código:

```
car = db.execute(
        '''
        SELECT CarCode, Model, Generation, YearFrom, YearTo, Trim, MotorType, Cylinders,
EngineType
        FROM cars
        WHERE CarCode = %s
        ''', code).fetchone()

if car is None:
    abort(404, 'Car Code {} does not exist.'.format(code))
makes = db.execute(
        '''
        SELECT Make_Id, Make
        FROM cars NATURAL JOIN makes
        WHERE CarCode = %s
        ''', code).fetchone()

bodytypes = db.execute(
        '''
        SELECT Code, Name
        FROM cars JOIN bodytype ON(cars.BodyType=bodytype.Code)
        WHERE CarCode = %s
        ''', code).fetchone()
```

- *Através de um CarCode, a APP procura o carro com o respetivo código. Se este existir, torna-se um endpoint que resulta da junção de 3 tabelas (CARS, MAKES e BODYTYPE) e retorna a informação associada ao carro.*

'/cars/search/<expr>/'

Código:

```
cars = db.execute(
        '''
        SELECT CarCode, Make_Id, Model, Trim, Generation
        FROM cars
        WHERE Model LIKE %s
        ''', expr).fetchall()
```

- *A APP procura e retorna todos os carros que tenham a expressão 'expr' contida no nome do modelo (Model).*

'/makes/'

Código:

```
makes = db.execute('''
        SELECT Make_Id, Make, Found_Year, OgCountry
        FROM makes
        ORDER BY Make_Id
        ''').fetchall()
```

- *Recolhe todas as marcas (makes) e respetivas características ordenando-as por Make_Id.*

'/makes/<int:id>/'

Código:

```
make = db.execute(
        '''
        SELECT Make_Id, Make, Found_Year, CON.Country
        FROM MAKES JOIN CON ON MAKES.OgCountry=CON.Country_Id
        WHERE Make_Id = %s
        ''', id).fetchone()   #id??

 if make is None:
        abort(404, 'Make id {} does not exist.'.format(id))

groups = db.execute(
        '''
        SELECT Group_Id, GroupName
        FROM makes JOIN grupos ON (makes.Subsidiary=grupos.Group_Id)
        WHERE Make_Id = %s
        ''', id).fetchone()

BrandBodyTypes=db.execute(
        '''
        SELECT BodyType, Name
        FROM BTMAKES JOIN BODYTYPE ON (BTMAKES.BodyType=BODYTYPE.Code)
        WHERE Make_Id = %s
        ''', id).fetchall()
```

- *A APP procura a marca correspondente a um certo Make_Id (caso exista) e devolve as informações que lhe estão associadas, incluindo o grupo a que pertence. Para além disso, devolve também todos bodytypes produzidos pela marca.*

'/makes/search/<expr>/'

Código:

```
makes = db.execute(
        ' SELECT Make_Id, Make'
        ' FROM makes '
        ' WHERE Make LIKE \'%' + expr + '%\''
        ).fetchall()
```

- *A APP procura e retorna todas as marcas que tenham a expressão 'expr' contida no nome.*

'/groups/'

Código:

```
groups = db.execute('''
        SELECT Group_Id, GroupName, Found_Year
        FROM grupos
        ORDER BY Group_Id
        ''').fetchall()
```

- *Recolhe todos os grupos e respetivas informações ordenando-as pelo Id do grupo.*

'/groups/<int:id>/'

Código:

```
 group = db.execute(
        '''
        SELECT Group_Id, GroupName,Found_Year
        FROM grupos
```

```
        WHERE Group_Id = %s
        ''', id).fetchone()

if group is None:
        abort(404, 'Group id {} does not exist.'.format(id))

makes = db.execute(
        '''
        SELECT Make, Make_Id
        FROM makes
        WHERE Subsidiary = %s
        ORDER BY Make
        ''', id).fetchall()

lider=db.execute(
        '''
        SELECT Make_Id,Make
        FROM LEADER NATURAL JOIN MAKES
        WHERE Group_Id = %s
        ''', id).fetchall()

coun=db.execute(
        '''
        SELECT CON.Country
        FROM COUNTRIES JOIN CON ON (COUNTRIES.Country=CON.Country_Id)
        WHERE Group_Id = %s
        ''', id).fetchall()
```

- *Caso o Id (Group_Id) recebido pela APP exista, esta cria um endpoint que usa as tabelas GRUPOS, MAKES para obter as informações gerais de cada grupo assim como as marcas subsidiárias de cada um. Usa também a tabela LEADER para obter a marca líder e as tabelas COUNTRIES e CON para obter os países a que grupo pertence.*

'/bodytypes/'

Código:

```
bodytype = db.execute(
        '''
        SELECT Code, Name
        FROM bodytype
        ORDER BY CODE
        ''').fetchall()
```

- *Recolhe todos os bodytypes ordenando-os pelo código (Code).*

'/bodytypes/<int:id>/'

Código:

```
bodytype = db.execute(
        '''
        SELECT Code, Name
        FROM bodytype
        WHERE Code = %s
        ''', id).fetchone()

if bodytype is None:
        abort(404, 'Bodytype id {} does not exist.'.format(id))
```

- *A APP procura pelo bodytype através do Code; caso este não exista a search é abortada.*