iscte
UNIVERSITY INSTITUTE OF LISBON

2020/2021(1º SEMESTRE)

TRABALHO 1: IMPORTAÇÃO DE JSON PARA MYSQL

Armazenamento para Big Data

Professor Pedro Ramos

GRUPO 12 | LCD-PL | "YELP"

Catarina Castanheira Nº 92478
Guilherme Firmino Nº 92811
João Martins Nº 93259
Joel Paula Nº 93392

Índice

Introdução

Para este trabalho foi escolhido um *dataset* que é um subconjunto de negócios, *reviews* e dados de utilizadores do Yelp. Este *dataset* contém 5 ficheiros em formato JSON que, por sua vez contêm informação sobre 5,200,000 *reviews* de utilizadores, informação sobre 209 393 negócios que abrangem 11 áreas metropolitanas em quatro países.

Para a sua realização será necessária uma divisão de tarefas que consistem em:

1. Análise e seleção dos dados;
2. Importação dos dados para MongoDB;
3. Exploração e análise dos dados;
4. Criação de um modelo relacional;
5. Transformação e exportação dos dados em MongoDB para formato CSV;
6. Ajustes ao modelo decorrentes das vicissitudes técnicas;
7. Criação de uma base de dados relacional em MySQL;
8. Importação dos CSV para MySQL;
9. Manipulação e limpeza dos dados importados, até que se ajustem ao modelo relacional.

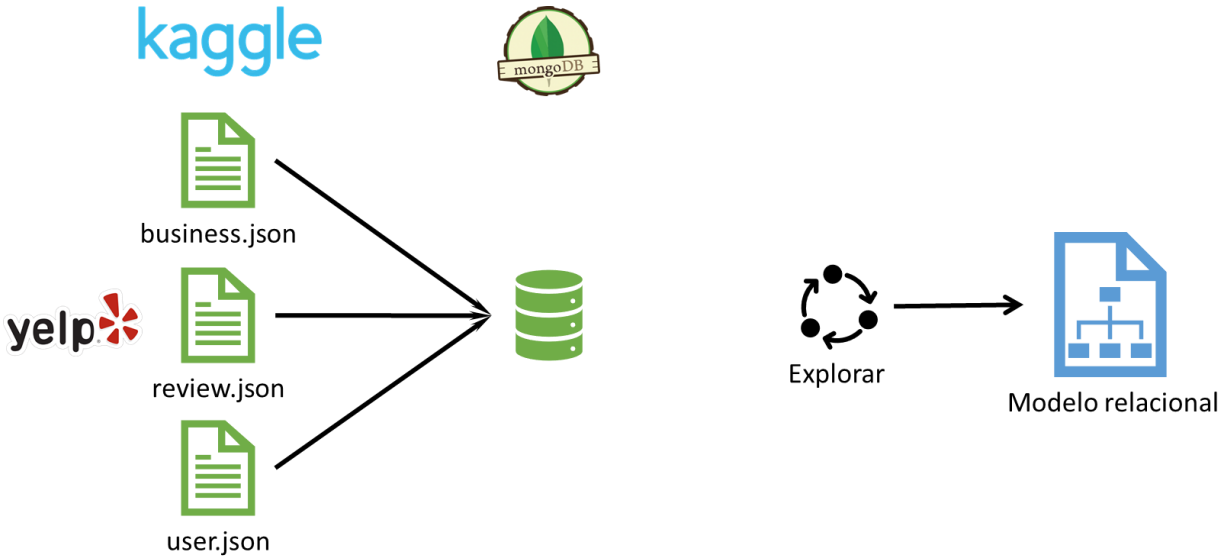

Figure 1 - Etapa 1

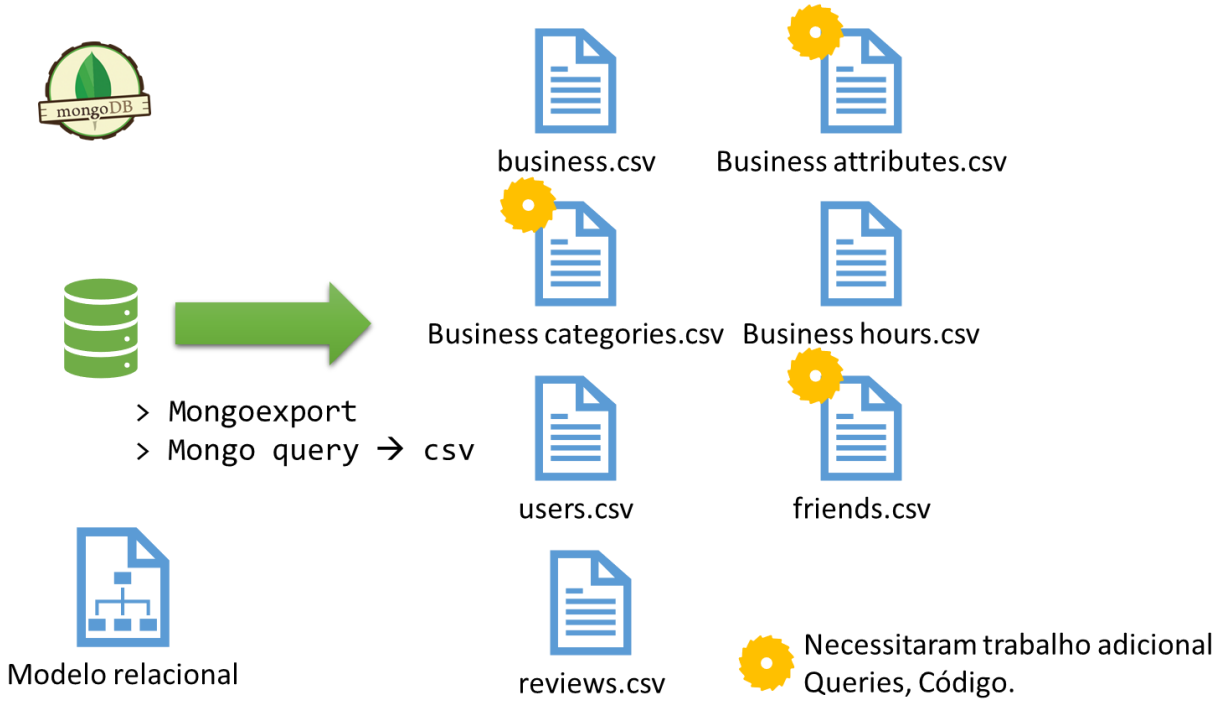

Figure 2 - Etapa 2

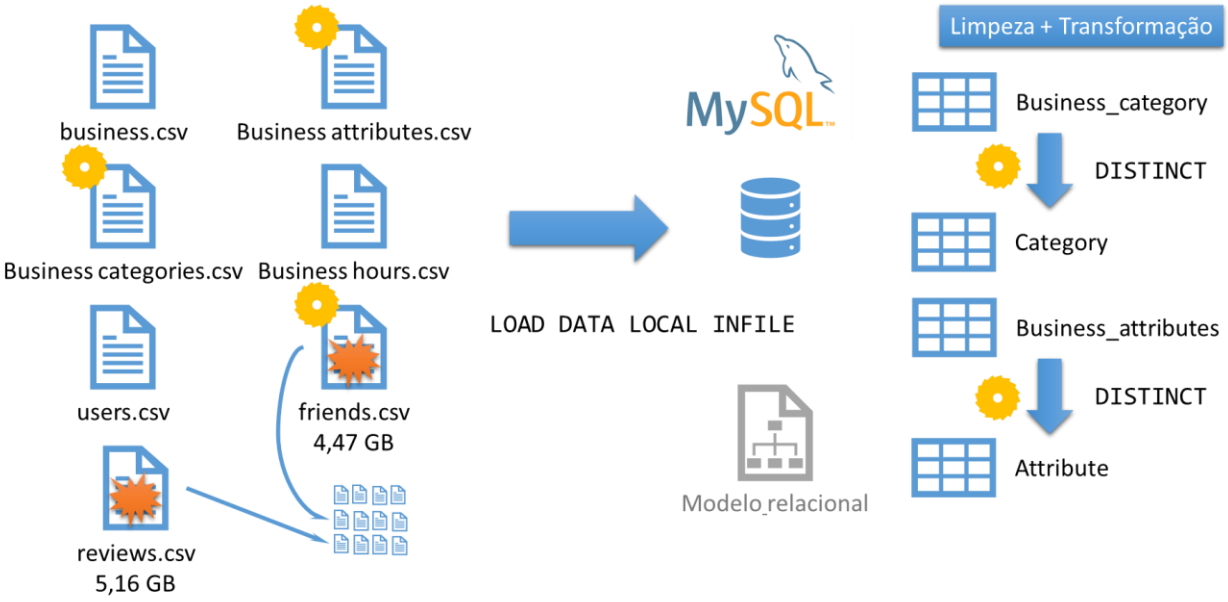

Figure 3 - Etapa 3

Pesquisa do *dataset*

O primeiro passo diz respeito à procura e escolha de um *dataset* interessante para o trabalho em questão, cumprindo os requisitos de ser em formato JSON e ter mais do que 1 milhão de registos. Assim,

foi escolhido um *dataset* sobre negócios, *reviews* e dados de utilizadores do Yelp na plataforma Kaggle (Yelp, Inc., 2020).

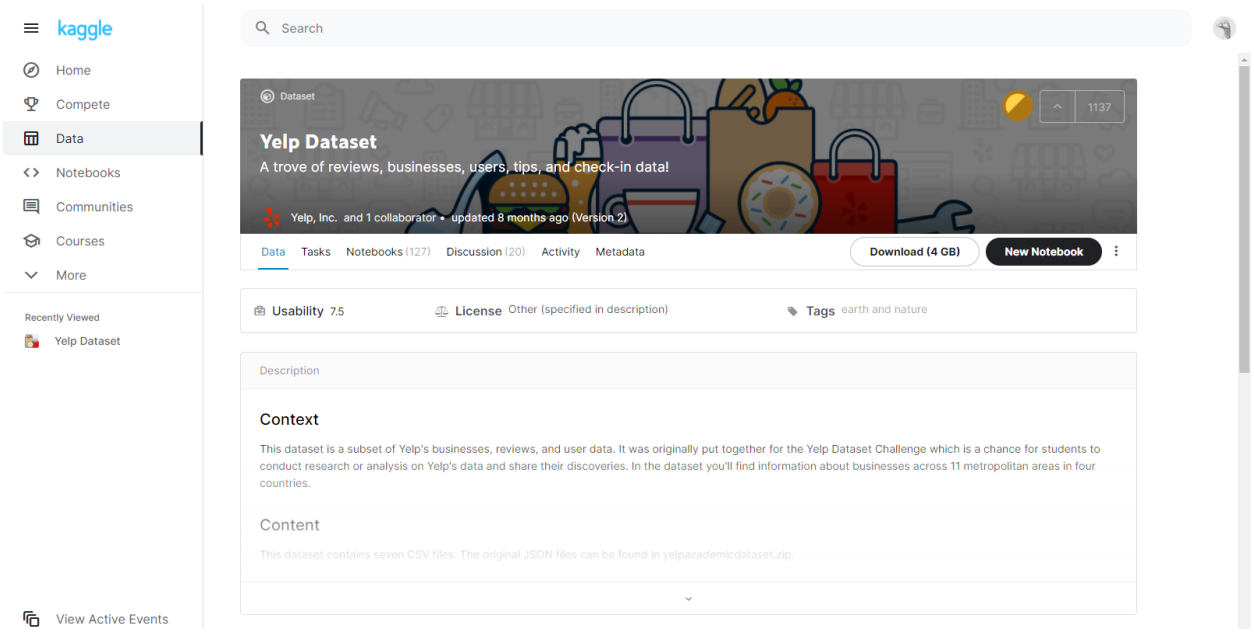

Figure 4- Dataset Yelp no Kaggle

Análise e Seleção

O segundo passo consistiu em analisar o *dataset* e selecionar os dados que seriam relevantes para o trabalho.

O *dataset* é composto por 5 ficheiros, dos quais apenas usámos 3:

Ficheiro	Dimensão	Registos	Comentário
yelp_academic_dataset_business.json	145.82 MB	209 393	Listagem de lojas/negócios
yelp_academic_dataset_checkin.json	428.83 MB	Não apurado	Listagem de "check-ins" de utilizadores em negócios. Não utilizámos este ficheiro, por não ser interessante.
yelp_academic_dataset_review.json	5.89 GB	8 021 122	Avaliações que utilizadores fizeram sobre negócios.
yelp_academic_dataset_tip.json	251.28 MB	Não apurado	Dicas de utilizadores sobre negócios. Não utilizámos este *dataset*, por não ser interessante.
yelp_academic_dataset_user.json	3.04 GB	1 968 703	Listagem anonimizada de utilizadores do Yelp.

Etapa 1: Importar e Explorar Dados Em MongoDB

Nesta etapa, interessa-nos importar os ficheiros do *dataset* que escolhemos para uma base de dados MongoDB e explorá-los para entender o formato dos dados, tamanhos, coleções e outras informações relevantes que nos permitirão construir o modelo relacional mais tarde.

Importação da coleção *business*

Importámos a colecção *yelp_business*, recorrendo ao mongoimport:

```
mongoimport --db yelpdb --collection yelp_business --drop --file
yelp_academic_dataset_business.json
```

Desta forma, com os dados disponíveis no MongoDB, é-nos possível explorá-los, ficando a compreender melhor os tipos de informação que temos e também as formas como se interligam.

Esta colecção específica, a *yelp_business*, contém informação relativa a 209 393 negócios, caracterizados com os campos: nome, morada, cidade, código postal, latitude, longitude, "estrelas" (avaliação média dos utilizadores), número de avaliações, se está aberto ou encerrado, e os horários de funcionamento. Contém também, como sub-colecções, tanto as categorias de negócio em que se enquadra (por exemplo: "Active Life, Gun/Rifle Ranges, Guns & Ammo, Shopping") como a caracterização do negócio de acordo com 39 atributos específicos (por exemplo: `BusinessAcceptsCreditCards[True, False], BikeParking[True, False], GoodForKids[True, False]`, etc.). Como exemplo, aplicando o comando `db.yelp_business.find().limit(1).pretty()` obtemos a informação sobre o primeiro documento:

```
> db.yelp_business.find().limit(1).pretty()
{
        "_id" : ObjectId("5fc379c0a2f7c1519362d029"),
        "business_id" : "f9NumwFMBDn751xgFiRbNA",
        "name" : "The Range At Lake Norman",
        "address" : "10913 Bailey Rd",
        "city" : "Cornelius",
        "state" : "NC",
        "postal_code" : "28031",
        "latitude" : 35.4627242,
        "longitude" : -80.8526119,
        "stars" : 3.5,
        "review_count" : 36,
        "is_open" : 1,
        "attributes" : {
                "BusinessAcceptsCreditCards" : "True",
                "BikeParking" : "True",
                "GoodForKids" : "False",
                "BusinessParking" : "{'garage': False, 'street': False, 'validated': False,
'lot': True, 'valet': False}",
                "ByAppointmentOnly" : "False",
                "RestaurantsPriceRange2" : "3"
        },
        "categories" : "Active Life, Gun/Rifle Ranges, Guns & Ammo, Shopping",
        "hours" : {
                "Monday" : "10:0-18:0",
                "Tuesday" : "11:0-20:0",
                "Wednesday" : "10:0-18:0",
                "Thursday" : "11:0-20:0",
                "Friday" : "11:0-20:0",
                "Saturday" : "11:0-20:0",
```

```
                "Sunday" : "13:0-18:0"
        }
}
```

Exploração da coleção *business*

Com o auxílio de comandos como `find()`, `disctinct()` complementados com `length` e `count()`, por exemplo (**Error! Reference source not found.**), conseguimos perceber que os campos *business_id*, *state*, *latitude*, *longitude*, *stars*, e *is_open* não admitem valores `Null`. Todos os restantes admitem: para o campo name temos 1 documento sem informação, para address temos 8679 documentos sem informação, para *city* temos 2, e para *postal_code* temos 509. Destes últimos, o campo com "*name*" e o "*city*", pelos pouquíssimos casos de ausência de informação poderiam em SQL ser de carácter obrigatório. Contudo, para podermos considerar à mesma toda a restante informação relativa aos negócios em questão, determinámos não fazer esta restrição no SQL – pese embora num contexto real pudesse fazer sentido não permitir a inserção de informação sobre um dado negócio sem no mínimo introduzir o nome do mesmo.

É também importante determinar nesta fase quais os tipos de dados com que estamos a lidar, para podermos mais tarde fazer a sua migração para SQL da forma mais adequada. Depois desta análise, é possível perceber que os tipos que podem ser considerados para cada campo na criação da tabela *business* no SQL são os seguintes:

- *business_id*: varchar(24);
- *name*: varchar(100);
- *address*: varchar(200);
- *city*: varchar(100);
- *state*: varchar(5);
- *postal_code*: varchar(15);
- *latitude*: float;
- *longitude*: float;
- *stars*: float;
- *is_open*: boolean;

Neste ponto realçamos também o facto de os campos "name", "address" e "city" não serem sujeitos pela natureza do próprio MongoDB a uma estandardização, ou seja, temos de admitir a possibilidade de existirem os seguintes casos:

- Negócios com ids diferentes, mas com designações iguais ou semelhantes (sugerindo a possibilidade de serem afinal um mesmo negócio);
- Moradas relativas à mesma localização específica, mas escritas de forma diferente;
- Nomes de cidades relativas a uma mesma cidade, mas escritos de forma diferente.

Relativamente a estes casos, torna-se muito complexa a tarefa de uniformização dos campos, uma vez que seria necessário fazer uma análise em todos os documentos e determinar, por exemplo, se o negócio em questão é na realidade um outro, mas que se encontra escrito de outra forma. Além disto, não temos informação detalhada sobre os negócios únicos que existem na base de dados, não temos informação sobre o país do negócio e os dados disponíveis são apenas uma amostra da base de dados total. Devido

a esta complexidade, decidimos não levar em diante a uniformização destes dados nem no MongoDB nem no MySQL.

Exploração da coleção *business attributes*

No caso do objeto *attributes* não temos nenhum campo cujo valor seja o nome do atributo, de tal forma que nos permita mais diretamente obter um conjunto de todos os campos únicos que podem ser considerados em *attributes*. Por isso, recorremos a uma ferramenta que nos permite explorar rapidamente "*nested objetcs*" (Maypop Inc, 2020). Depois de correr esta ferramenta (ver **Error! R eference source not found.**) obtemos então uma listagem dos 39 objetos que podemos encontrar em *yelp_business.atributes*. São eles: *BusinessAcceptsCreditCards, RestaurantsPriceRange2, BikeParking, GoodForKids, RestaurantsTakeOut, WiFi, ByAppointmentOnly, OutdoorSeating, RestaurantsDelivery, RestaurantsGoodForGroups, RestaurantsReservations, HasTV, Alcohol, RestaurantsAttire, NoiseLevel, Caters, WheelchairAccessible, RestaurantsTableService, DogsAllowed, BusinessAcceptsBitcoin, HappyHour, AcceptsInsurance, GoodForDancing, CoatCheck, DriveThru, Smoking, BYOBCorkage, Corkage, BYOB, AgesAllowed, Open24Hours, RestaurantsCounterService*. Tínhamos ainda também outros 7 campos (*BusinessParking, Ambience, GoodForMeal, Music, BestNights, HairSpecializesIn, DietaryRestrictions*), mas que o grupo decidiu não incluir neste projeto de migração para SQL devido à complexidade acrescida inerente (acrescentar as 7 tabelas necessárias) e à reduzida informação em comparação.

Com a lista de atributos, foi possível explorar de forma mais incisiva o tipo de informação possível em cada um deles. Em anexo (**Error! Reference source not found.**) encontramos os comandos usados para este efeito e os respectivos outputs. A maioria dos atributos, além da ausência de valor (ie, *Null* ou "*None*"), podem assumir `True`/`False` e em alguns casos strings específicas (exemplo: `attributes.NoiseLevel` pode assumir `["'average'", "'loud'", "'quiet'", "'very_loud'", "None", "u'average'", "u'loud'", "u'quiet'", "u'very_loud'"]`. Aquando a importação de dados para a tabela business_attribute em SQL teremos de considerar tanto a ausência de valor para um dado atributo como a própria existência de um valor designado "`None`" que iremos assumir ter o mesmo significado teórico que `Null`. Veja-se o exemplo:

```
> db.yelp_business.distinct("attributes.Corkage");
[ "False", "None", "True" ]
> db.yelp_business.find({"attributes.Corkage":{$exists:false}}).count();
208303
> db.yelp_business.find({"attributes.Corkage":{$exists:true}}).count();
1090
> db.yelp_business.find({"attributes.Corkage":"None"}).count();
4
> db.yelp_business.find({"attributes.Corkage":"True"}).count();
253
> db.yelp_business.find({"attributes.Corkage":"False"}).count();
833
```

Ou seja, somando os totais de de *True*, *False*, e *None* obtemos 1090, que corresponde exactamente ao número de documentos para os quais existe algum valor associado (ou seja, `attributes.Corkage":{$exists:true}}`).

Outro pormenor que encontramos nestes dados é que os campos `attribute.WiFi`, `attribute.Alcohol, attribute.NoiseLevel, attribute.Smoking` e

`attribute.BYOBCorkage` admitem valores precedidos da letra `'u'`, com a tipologia: palavra1, palavra2, palavra3, None, u'palavra1', u'palavra2', u'palavra3'. Temos como exemplo:

```
> db.yelp_business.distinct("attributes.WiFi");
[ "'free'", "'no'", "'paid'", "None", "u'free'", "u'no'", "u'paid'"  ]
```

De acordo com a documentação disponível do PyMongo (MongoDB, Inc, 2020) estas situações têm que ver com a necessidade de transformar do formato Unicode para UTF-8, deixando assim as *strings* em Unicode precedidas do *char* 'u'.

Para estes casos, tomamos a decisão de não fazer a sua transformação já e deixá-la para quando tivermos os dados em SQL.

Depois deste tipo de análise feita para todos os campos, ficamos assim com a informação necessária sobre os tipos de dados para a tabela business_attribute mais à frente no SQL.

Exploração de *business categories*

No campo *categories* encontramos uma lista de categorias separadas por vírgulas e é nossa intenção criar uma tabela dedicada para formalizar a relação dos negócios com as categorias mais adiante.

Posto isto, decidimos processá-lo no momento da exportação para CSV.

Conseguimos explorar as categorias individualmente usando o comando *aggregate*:

```
> db.yelp_business.aggregate([{$project: {_id: 0, business_id: 1, categories: {$split:
["$categories", ", "]}}}, {$unwind: "$categories"},{$out:"categoriesdist"}]);

> db.categoriesdist.distinct("categories");
```

Com isto verificámos que existem 1292 categorias únicas no nosso *dataset* e são efetivamente usadas de forma sistemática para caracterizar os negócios, pelo que faz sentido autonomizar estas categorias na sua própria tabela.

Exploração de *business hours*

A estrutura de *business hours* é simples, com um horário por cada dia da semana:

```
"hours" : {
     "Monday" : "10:0-18:0",
     "Tuesday" : "11:0-20:0",
     "Wednesday" : "10:0-18:0",
     "Thursday" : "11:0-20:0",
     "Friday" : "11:0-20:0",
     "Saturday" : "11:0-20:0",
     "Sunday" : "13:0-18:0"
     }
```

Fomos verificar quantos negócios têm horas de funcionamento disponíveis para cada dia da semana:

```
> db.yelp_business.find({"hours.Monday":{$exists:true}},{}).count()
151872
> db.yelp_business.find({"hours.Monday":{$exists:false}},{}).count()
57521
>
> db.yelp_business.find({"hours.Tuesday":{$exists:true}},{}).count()
159636
> db.yelp_business.find({"hours.Tuesday":{$exists:false}},{}).count()
```

```
49757
>
> db.yelp_business.find({"hours.Wednesday":{$exists:true}},{}).count()
161491
> db.yelp_business.find({"hours.Wednesday":{$exists:false}},{}).count()
47902
>
> db.yelp_business.find({"hours.Thursday":{$exists:true}},{}).count()
162281
> db.yelp_business.find({"hours.Thursday":{$exists:false}},{}).count()
47112
>
> db.yelp_business.find({"hours.Friday":{$exists:true}},{}).count()
161515
> db.yelp_business.find({"hours.Friday":{$exists:false}},{}).count()
47878
>
> db.yelp_business.find({"hours.Saturday":{$exists:true}},{}).count()
139579
> db.yelp_business.find({"hours.Saturday":{$exists:false}},{}).count()
69814
>
> db.yelp_business.find({"hours.Sunday":{$exists:true}},{}).count()
102117
> db.yelp_business.find({"hours.Sunday":{$exists:false}},{}).count()
107276
```

É possível concluir que este campo não é obrigatório e que os campos dos dias individuais também não são obrigatórios – isto é, um negócio pode ter horário definido num dia e não ter noutro, podendo isso indicar que nesse dia está fechado.

Verificamos mesmo que alguns negócios não têm qualquer informação de horário de funcionamento:

```
> db.yelp_business.find({"hours.Monday":{$exists:false},
"hours.Tuesday":{$exists:false},
"hours.Wednesday":{$exists:false},
"hours.Thursday":{$exists:false},
"hours.Friday":{$exists:false},
"hours.Saturday":{$exists:false},
"hours.Sunday":{$exists:false}},{}).count()
44843
```

Quase metade dos negócios (93 230) estão abertos todos os dias da semana:

```
> db.yelp_business.find({"hours.Monday":{$exists:true},
... "hours.Tuesday":{$exists:true},
... "hours.Wednesday":{$exists:true},
... "hours.Thursday":{$exists:true},
... "hours.Friday":{$exists:true},
... "hours.Saturday":{$exists:true},
... "hours.Sunday":{$exists:true}},{}).count()
93230
```

Importação e Exploração da Coleção *Users*

Importámos a coleção yelp_dataset_user, recorrendo ao mongoimport:

```
❯ mongoimport --db yelpdb --collection users --drop --file yelp_academic_dataset_user.json
```

```
1968703 document(s) imported successfully. 0 document(s) failed to import.
```

Desta forma, com os dados disponíveis no MongoDB, é-nos possível explorá-los, ficando a compreender melhor os tipos de informação que temos e também as formas como se interligam.

Esta coleção específica, a *yelp_users*, contém informação relativa a 1 968 703 *users*, com os campos: id, nome, número de *reviews*, data em que se inscreveram no yelp, avaliações de outros utilizadores (votos úteis, "engraçado" e outros para as suas *reviews*), uma lista de amigos e se são ou não "elite".

Verificando a estrutura de cada registo:

```
> db.users.findOne()
{
        "_id" : ObjectId("5fca7f842a9571e151e73ea1"),
        "user_id" : "ntlvfPzc8eglqvk92iDIAw",
        "name" : "Rafael",
        "review_count" : 553,
        "yelping_since" : "2007-07-06 03:27:11",
        "useful" : 628,
        "funny" : 225,
        "cool" : 227,
        "elite" : "",
        "friends" : "oeMvJh94PiGQnx_6GlndPQ, wm1z1PaJKvHgSDRKfwhfDg,
IkRib6Xs91PPW7pon7VVig, A8Aq8f0-XvLBcyMk2GJdJQ, eEZM1kogR7eL4GOBZyPvBA,
e1o1LN7ez5ckCpQeAab4iw, _HrJVzFaRFUhPva8cwBjpQ, pZeGZGzX-ROT_D5lam5uNg, 0S6EI51ej5J7dgYz3-
O0lA, woDt8raW-AorxQM_tIE2eA, hWUnSE5gKXNe7bDc8uAG9A, c_3LDSO2RHwZ94_Q6j_O7w, -
uv1wDiaplY6eXXS0VwQiA, QFjqxXn3acDC7hckFGUKMg, ErOqapICmHPTN8YobZIcfQ,
mJLRvqLOKhqEdkgt9iEaCQ, VKX7jlScJSA-ja5hYRw12Q, ijIC9w5PRcj3dWVlanjZeg, CIZGlEw-
Bp0rmkP8M6yQ9Q, OC6fT5WZ8EU7tEVJ3bzPBQ, UZSDGTDpycDzrlfUlyw2dQ, deL6e_z9xqZTIODKqnvRXQ,
5mG2ENw2PylIWElqHSMGqg, Uh5Kug2fvDd51RYmsNZkGg, 4dI4uoShugD9z84fYupelQ,
EQpFHqGT9Tk6YSwORTtwpg, o4EGL2-ICGmRJzJ3GxB-vw, s8gK7sdVzJcYKcPv2dkZXw, vOYVZgb_GVe-
kdtjQwSUHw, wBbjgHsrKr7BsPBrQwJf2w, p59u2EC_qcmCmLeX1jCi5Q, VSAZI1eHDrOPRWMK4Q2DIQ,
efMfeI_dkhpeGykaRJqxfQ, x6qYcQ8_i0mMDzSLsFCbZg, K_zSmtNGw1fu-vmxyTVfCQ, 5IM6YPQCK-
NABkXmHhlRGQ, U_w8ZMD26vnkeeS1sD7s4Q, AbfS_oXF8H6HJb5jFqhrLw, hbcjX4_D4KIfonNnwrH-cg,
UKf66_MPz0zHCP70mF6p1g, hK2gYbxZRTqcqlSiQQcrtQ, 2Q45w_Twx_T9dXqlE16xtQ,
BwRn8qcKSeA77HLaOTbfiQ, jouOn4VS_DtFPtMR2w8VDA, ESteyJabbfvqas6CEDs3pQ",
        "fans" : 14,
        "average_stars" : 3.57,
        "compliment_hot" : 3,
        "compliment_more" : 2,
        "compliment_profile" : 1,
        "compliment_cute" : 0,
        "compliment_list" : 1,
        "compliment_note" : 11,
        "compliment_plain" : 15,
        "compliment_cool" : 22,
        "compliment_funny" : 22,
        "compliment_writer" : 10,
        "compliment_photos" : 0
}
```

Explorando o campo "elite", verificamos que tem uma lista de anos em que o utilizador é elite:

```
    > db.users.distinct("elite")
    [
            "",
            "2006",
```

```
            "2006,2007",
            "2006,2007,2008",
            "2006,2007,2008,2009",
            "2006,2007,2008,2009,2010",
            "2006,2007,2008,2009,2010,2011",
            "2006,2007,2008,2009,2010,2011,2012",
            "2006,2007,2008,2009,2010,2011,2012,2013",
            "2006,2007,2008,2009,2010,2011,2012,2013,2014",
            "2006,2007,2008,2009,2010,2011,2012,2013,2014,2015",
            "2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016",
            "2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016,2017",
            …
```

Este campo nunca está a *null* na coleção.

Importação e exploração da coleção *Reviews*

Importámos para uma coleção do MongoDB, as *reviews*:

```
❯ mongoimport --db yelpdb --collection reviews --drop --file C:\Users\user
\Yelp\yelp_academic_dataset_review.json
8021122 document(s) imported successfully. 0 document(s) failed to import.
```

8 milhões de *reviews*.

Verificando a estrutura típica de uma *review*:

```
> db.reviews.findOne({},{_id:0});
{
  "review_id": "UmFMZ8PyXZTY2QcwzsfQYA",
  "user_id": "nIJD_7ZXHq-FX8byPMOkMQ",
  "business_id": "lbrU8StCq3yDfr-QMnGrmQ",
  "stars": 1,
  "useful": 1,
  "funny": 1,
  "cool": 0,
  "text": "I am actually horrified this place is still in business…",
  "date": "2013-12-07 03:16:52"
}
```

Existe uma referência ao utilizador que fez a *review* (*user_id*), bem como ao negócio a que se refere (*business_id*).

Para além disso, podemos verificar que o campo *stars* indica o número de estrelas, de 1 a 5, e que todas as *reviews* têm estrelas:

```
> db.reviews.distinct("stars");
[
  1,
  2,
  3,
  4,
  5
]
> db.reviews.find({"stars":{$exists:false}},{}).count();
0
```

Também verificamos que os campos *useful*, *funny* e *cool* estão sempre preenchidos com o número de votos que foi atribuído à *review* por cada categoria.

Para verificar o tamanho máximo do texto da *review* usamos o *aggregate*, com a função *$strLenCP*:

```
> db.reviews.aggregate([{ $match:{}  },
{$addFields: {"length": { $strLenCP: "$text" }}},
{"$sort": { "length": -1 }},
{$limit:1}
]);
[
  {
    "_id": {
      "$oid": "5fd346868a1e68d90f0fe358"
    },
    "review_id": "qJPOzmEWywnLDZXOFkRyxg",
    "user_id": "MgqEi4Qvx9Z70Z-qfeF3Gg",
    "business_id": "tKi4OvAlckcVzeP92InsOQ",
    "stars": 1,
    "useful": 2,
    "funny": 1,
    "cool": 2,
    "text": "This is not hyperbole…",
    "date": "2017-03-05 14:41:33",
    "length": 5000
  }
]
```

E com isto ficamos a saber que tipo de campo de texto precisamos criar na tabela de dados relacional.

Modelo relacional

Após a exploração inicial dos dados em MongoDB é possível obter uma visão geral da relação das diversas coleções e dos diversos campos de dados.

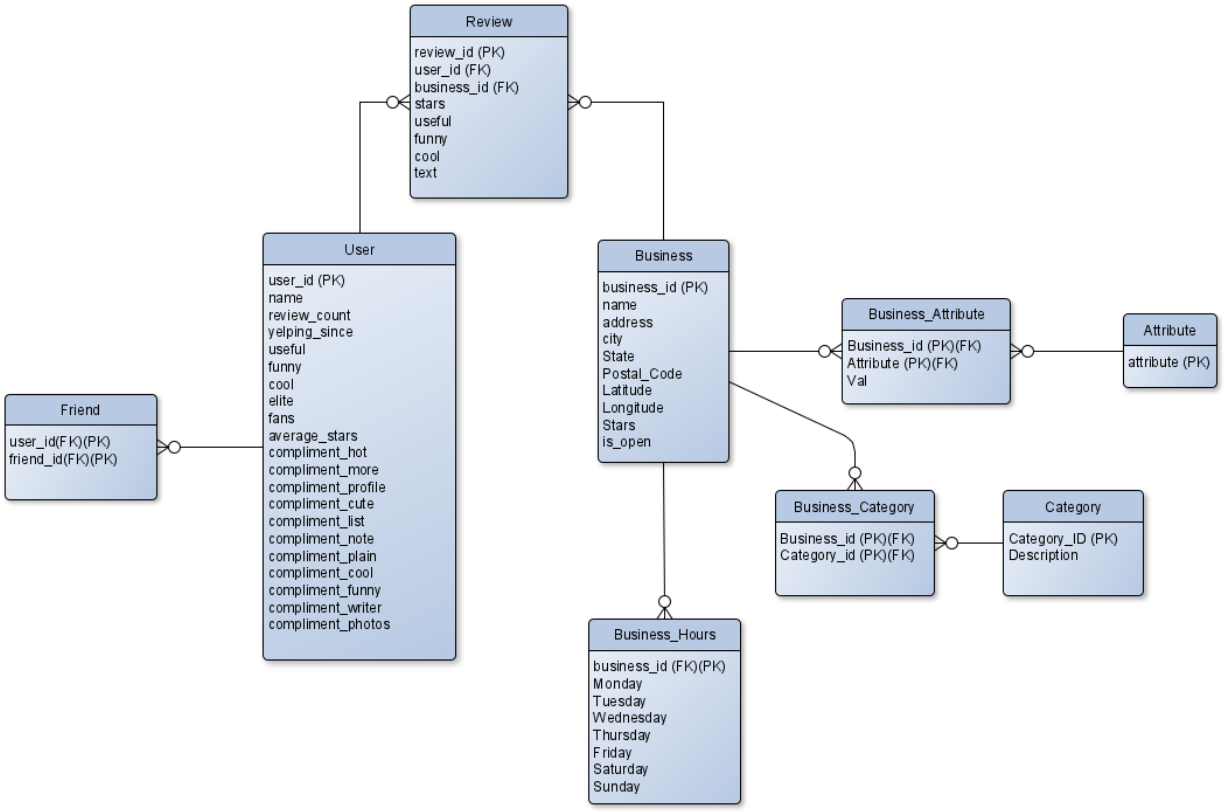

Figure 5- Modelo Relacional

Sobre estas relações decidiu-se criar um esquema relacional de acordo com a imagem acima.

As tabelas principais são a *Business*, *User* e *Review,* que acabam por corresponder aos três ficheiros importados. A partir destas três tabelas, derivam as restantes.

Etapa 2: Exportar Dados Para Ficheiros CSV

Nesta etapa, após a exploração dos dados e a construção do modelo relacional, exportamos os dados em ficheiros CSV o mais aproximado possível às tabelas do nosso modelo. Se para as tabelas principais a exportação é praticamente direta, para algumas das sub-tabelas como *attributes*, *categories* e *friends* a exportação inclui uma *query* complexa do lado do MongoDB.

Exportação da coleção *business*

Com recurso à ferramenta mongoexport, iremos criar ficheiros CSV distintos: um que irá conter os campos *business_id*, *name*, *address*, *city*, *state*, *postal_code*, *latitude*, *longitude*, *stars*, e *is_open*; e outro que irá conter a informação de todos os campos dentro de *attributes*.

```
mongoexport --db yelpdb --collection yelp_business --out exportBusiness_Business.csv --
type=csv –fields
business_id,name,address,city,state,postal_code,latitude,longitude,stars,is_open

mongoexport --db yelpdb --collection yelp_business --out
exportBusiness_BusinessAttributes.csv --type=csv --fields
business_id,attributes.BusinessAcceptsCreditCards,attributes.RestaurantsPriceRange2,attribut
es.BikeParking,attributes.GoodForKids,attributes.RestaurantsTakeOut,attributes.WiFi,attribut
```

```
es.ByAppointmentOnly,attributes.OutdoorSeating,attributes.RestaurantsDelivery,attributes.Res
taurantsGoodForGroups,attributes.RestaurantsReservations,attributes.HasTV,attributes.Alcohol
,attributes.RestaurantsAttire,attributes.NoiseLevel,attributes.Caters,attributes.WheelchairA
ccessible,attributes.RestaurantsTableService,attributes.DogsAllowed,attributes.BusinessAccep
tsBitcoin,attributes.HappyHour,attributes.AcceptsInsurance,attributes.GoodForDancing,attribu
tes.CoatCheck,attributes.DriveThru,attributes.Smoking,attributes.BYOBCorkage,attributes.Cork
age,attributes.BYOB,attributes.AgesAllowed,attributes.Open24Hours,attributes.RestaurantsCoun
terService
```

exportBusiness_Business.csv 22,4 MB.

exportBusiness_BusinessAttributes 16,3 MB.

Exportação da coleção *business categories*

Iremos criar um ficheiro CSV que irá conter apenas as variáveis *business_id* e *categories* (para criar as tabelas *business_category* e *category*) com recurso à operação *aggregate* para executar os seguintes passos: *project* (que guarda os campos "pretendidos" para a próxima fase do comando), *split* (que separa uma expressão do tipo *string* e remove delimitadores) e *unwind* (que desconstrói o *array* das categorias e gera um documento para cada elemento) e a função *.forEach*() que é função *JavaScript* que é aplicada a cada elemento do documento obtido no passo anterior.

```
db.yelp_business.aggregate([{$project:{_id:0, business_id:1, categories: {$split:
["$categories", ", "]}}},
{$unwind:"$categories"}]).forEach(function(pr){print("\""+pr.business_id+"\",\""+pr.categori
es+"\"")});
```

Esta query devolve um resultado que já relaciona cada negócio com cada uma das categorias, que mais tarde irá facilitar o tratamento dos dados:

```
"fuukQ0bggEEBKVYn_e3d7Q","Cafes"
"fuukQ0bggEEBKVYn_e3d7Q","Restaurants"
"fuukQ0bggEEBKVYn_e3d7Q","Sandwiches"
"fuukQ0bggEEBKVYn_e3d7Q","Breakfast & Brunch"
"o9kh2LJDlNdgz8pjF0olxA","Nail Salons"
"o9kh2LJDlNdgz8pjF0olxA","Beauty & Spas"
"CYmhTXL29bc1UCj4mhx-6w","Oil Change Stations"
"CYmhTXL29bc1UCj4mhx-6w","Tires"
"CYmhTXL29bc1UCj4mhx-6w","Automotive"
"CYmhTXL29bc1UCj4mhx-6w","Auto Repair"
"yA9dC3IvoPPex9RB6-G2MA","Chinese"
```

Neste caso não usamos o *MongoExport*, uma vez que queremos correr essa *query* para transformação dos dados. Usamos diretamente o *Mongo* cliente, redirecionando o resultado da *query* para o ficheiro csv:

```
> mongo –quiet yelpdb ExportYelpReviews.mongodb > yelp_business_categories.csv
```

Yelp_business_categories.csv 14.3 MB.

Exportação da coleção *business hours*

É uma exportação direta a partir da subestrutura *hours* da coleção business:

```
mongoexport --db yelpdb --collection yelp_business --out export_business_hours.csv --
type=csv --fields business_id, hours.Monday, hours.Tuesday, hours.Wednesday, hours.Thursday,
hours.Friday, hours.Saturday, hours.Sunday
```

export_business_hours.csv 14,3 MB

Exportação Coleção *Users* e *Friends*

A exportação é simples, deixando de fora a lista de amigos (*friends*), pois serão exportados para a sua própria tabela:

```
❯ mongoexport --db yelpdb --collection users --type=csv --fields
"user_id,name,review_count,yelping_since,useful,funny,cool,elite,fans,average_stars,complime
nt_hot,compliment_more,compliment_profile,compliment_cute,compliment_list,compliment_note,co
mpliment_plain,compliment_cool,compliment_funny,compliment_writer,compliment_photos" --out
yelp_users.csv
exported 1968703 records
```

O resultado é um ficheiro com 168 MB.

Para exportar os amigos, criamos o ficheiro <ExportYelpFriends.mongodb>, com o código JSON que constrói uma tabela de relações entre utilizadores:

```
print("user_id,friend_id");
db.users
  .aggregate([
    {$project: { _id: 0,
      user_id: 1, friends: {$split: ["$friends", ", "]}
    } },
    {$unwind: "$friends"}
  ]).forEach(function(pr){
                    print(pr.user_id+","+pr.friends)
                });
```

E, para exportar o ficheiro fazemos o comando:

```
❯ mongo --quiet yelpdb ExportYelpFriends.mongodb > yelp_friends.csv
```

O resultado é um ficheiro com 4.7 GB.

Exportação Coleção *Review*

A exportação é simples, pois serão exportados para a sua própria tabela:

```
❯ mongoexport --db yelp --collection reviews --type=csv --fields
"review_id,user_id,business_id,stars,useful,funny,cool,text,date" --out yelp_reviews.csv
exported 8021122 records
```

O resultado é um ficheiro com 5,16 GB.

Etapa 3: Importação de Dados de CSV para MySQL

Depois de criado o esquema de base de dados *yelp*, procedemos à criação das tabelas relacionais necessárias, de acordo com o anexo (Anexo IV – DDL Criação Estrutura BD relacional). A criação das chaves estrangeiras foi postergada para uma fase seguinte à importação dos dados nas tabelas, pois torna a importação mais rápida uma vez que as chaves estrangeiras não têm que ser verificadas para cada um dos registos importados.

Tabelas *business*, *business_hours*, *users*, *friends* e *reviews*

Para estas tabelas basta importar diretamente o CSV, não se tendo revelado necessário nenhum tipo de tratamento adicional dos dados. Sendo assim, todas estas importações usam o método LOAD DATA INFILE:

```
LOAD DATA LOCAL INFILE yelp_ficheiro.csv' INTO TABLE `Nome_Tabela`
FIELDS OPTIONALLY ENCLOSED BY '"' TERMINATED BY ','    IGNORE 1 LINES;
```

Business atributtes

Criámos uma tabela temporária para albergar uma primeira estrutura dos dados contidos naquilo que era o campo yelp_business.attributes do ficheiro JSON original. Designamos esta tabela de *business_attribute_inicial* (ver estrutura abaixo). Temos de ter aqui em atenção considerar como VARCHAR todos aqueles campos que tinham admitido no ficheiro JSON os resultados ["False", "None", "True"]. Numa primeira tentativa tínhamos considerado estes campos como BOOLEAN aquando a sua definição na estrutura da tabela em SQL, o que teve como consequência a caracterização de todos os valores "None" como True. Contudo, o nosso objectivo é considerar os "None" como ausência de valor. Assim, iremos importar para esta tabela os valores originais como VARCHAR (5) e depois desconsiderar tanto os "None" como os *Null*. No caso do *attribute* Alcohol iremos também considerar "'none'" e "u'none'" como ausência de valor.

```
    CREATE TABLE business_attribute_inicial
    (
      business_id varchar(24) NOT NULL,
      business_accepts_credit_cards VARCHAR(5) NULL,
      restaurants_price_range2 VARCHAR(5) NULL,
      bike_parking VARCHAR(5) NULL,
      good_for_kids VARCHAR(5) NULL,
      restaurants_take_out VARCHAR(5) NULL,
      wifi VARCHAR(10) NULL,
      by_appointment_only VARCHAR(5) NULL,
      outdoor_seating VARCHAR(5) NULL,
      restaurants_delivery VARCHAR(5) NULL,
      restaurants_good_for_groups VARCHAR(5) NULL,
      restaurants_reservations VARCHAR(5) NULL,
      has_tv VARCHAR(5) NULL,
      alcohol VARCHAR(20) NULL,
      restaurants_attire VARCHAR(10) NULL,
      noise_level VARCHAR(20) NULL,
      caters VARCHAR(5) NULL,
      wheelchair_accessible VARCHAR(5) NULL,
      restaurants_table_service VARCHAR(5) NULL,
      dogs_allowed VARCHAR(5) NULL,
      business_accepts_bitcoin VARCHAR(5) NULL,
      happy_hour VARCHAR(5) NULL,
      accepts_insurance VARCHAR(5) NULL,
      good_for_dancing VARCHAR(5) NULL,
      coat_check VARCHAR(5) NULL,
      drive_thru VARCHAR(5) NULL,
      smoking VARCHAR(10) NULL,
      byob_corkage VARCHAR(20) NULL,
      corkage VARCHAR(5) NULL,
      byob VARCHAR(5) NULL,
      ages_allowed VARCHAR(10) NULL,
      open_24hours VARCHAR(5) NULL,
      restaurants_counter_service VARCHAR(5) NULL,
```

```
    CONSTRAINT PK_business_attribute_inicial PRIMARY KEY (business_id)
);
```

Criámos também uma tabela *attribute* (Anexo IV – DDL Criação Estrutura BD relacional), que vai albergar o conjunto de elementos específicos de "attributes" que cada negócio poderá assumir; no fundo estamos a criar restrições na adição de futuros negócios, obrigando-os a assumir um dos valores aqui disponíveis.

Finalmente, temos a tabela *business_attribute* (Anexo IV – DDL Criação Estrutura BD relacional), aquela que irá albergar de forma definitiva a informação que começámos por encontrar na tabela temporária com os *attributes*. Cada linha irá conter o identificador do negócio, o nome de um atributo (*attribute*) específico, e o valor referente ao atributo (*Val*). A tabela não permitirá a inserção de combinações dos campos da chave primária *business_id, attribute* sem também existir um valor *val* associado. Nesta fase não definimos ainda as chaves estrangeiras nesta tabela

Para os ficheiros `export_Business.csv` e `exportBusiness_Attributes.csv`, executamos um comando para carregar os dados. Primeiro carregamos dados para a tabela *business* e depois para a tabela temporária *business_attribute_inicial*.

```
LOAD DATA LOCAL INFILE "export_Business.csv" INTO TABLE business FIELDS TERMINATED BY ','
ENCLOSED BY '"' IGNORE 1 LINES;

LOAD DATA LOCAL INFILE "exportBusiness_Attributes.csv" INTO TABLE business_attribute_inicial
FIELDS TERMINATED BY ',' ENCLOSED BY '"' IGNORE 1 LINES;
```

A tabela *business_attribute_inicial* é pouco prática, armazenando os dados de forma muito pouco eficaz, já que associa para cada *business_id* um atributo de negócio, mesmo nos casos em que originalmente não existiam atributos associados ao negócio. Arriscamo-nos desta forma a ter uma tabela repleta de "informação vazia". Apesar de ser simples a extração de dados desta forma para o SQL a partir de JSON, com o MySQL conseguimos criar relações entre os diversos dados com critérios mais rígidos e "limpando" a ausência de dados.

O nosso objetivo é ter a tabela *attribute* preenchida com os 32 atributos de negócio. Por este motivo criámos um conjunto de instruções específicas para cada atributo conforme o exemplo abaixo (ver detalhe das instruções em Anexo V - Inserir os dados na tabela attribute):

```
INSERT INTO attribute (attribute) VALUES ("business_accepts_credit_cards");
INSERT INTO attribute (attribute) VALUES ("restaurants_price_range2");
-- (…)
INSERT INTO attribute (attribute) VALUES ("restaurants_counter_service");
```

De seguida conseguimos então carregar os dados para a *tabela business_attribute* (ver Anexo VI - Transformar e migrar os dados da tabela business_attribute_inicial para a business_attribute, desconsiderando *Nulls* e campos vazios para script completo):

```
-- attribute 1
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "business_accepts_credit_cards", business_accepts_credit_cards
   FROM business_attribute_inicial
   WHERE length(business_accepts_credit_cards) > 0
   AND business_accepts_credit_cards <> "None"
```

```
;
-- (…)
-- attribute 32
INSERT INTO business_attribute (business_id, attribute, val)
   SELECT business_id, "restaurants_counter_service", restaurants_counter_service
   FROM business_attribute_inicial
   WHERE length(restaurants_counter_service) > 0
   AND restaurants_counter_service <> "None"
;
```

Após esta inserção dos dados procedemos então à planeada criação das chaves estrangeiras *business_id* e *attribute* (Anexo VII - Definição de Chaves Estrangeiras na tabela Business_attribute). Uma vez que o nosso objetivo é permitir que a atualização dos campos *business_id* e *attribute* seja refletida nesta tabela, o critério `ON UPDATE` ficou definido como `CASCADE`. Relativamente à eliminação de registos da tabela business, definimos como `CASCADE` uma vez que também queremos propagar a alteração automaticamente sem deixar registos órfãos. Contudo em relação à tabela *attribute* não queremos que uma eliminação seja permitida no caso de ainda existirem negócios que façam uso de um dado atributo – por isso usámos `ON DELETE RESTRICT`.

Limpeza dos restantes dados em *business_attribute*

Conforme tinha sido planeado na altura da análise dos dados em JSON, executámos a limpeza dos dados com instruções SQL nesta fase. O objetivo aqui é unir valores de atributos que apesar de terem significado teórico igual têm estado representados de forma diferente. Queremos aqui estandardizar os dados.

```
-- business_attribute.wifi
UPDATE business_attribute SET val = "paid"
WHERE (attribute = "wifi" AND val IN("u'paid'", "'paid'"));
UPDATE business_attribute SET val = "no"
WHERE (attribute = "wifi" AND val IN("u'no'", "'no'"));
UPDATE business_attribute SET val = "free"
WHERE (attribute = "wifi" AND val IN("u'free'", "'free'"));

-- business_attribute.alcohol
UPDATE business_attribute SET val = "beer_and_wine"
WHERE (attribute = "alcohol" AND val IN("u'beer_and_wine'", "'beer_and_wine'"));
UPDATE business_attribute SET val = "full_bar"
WHERE (attribute = "alcohol" AND val IN("u'full_bar'", "'full_bar'"));

-- business_attribute.restaurants_attire
UPDATE business_attribute SET val = "casual"
WHERE (attribute = "restaurants_attire" AND val IN("u'casual'", "'casual'"));
UPDATE business_attribute SET val = "dressy"
WHERE (attribute = "restaurants_attire" AND val IN("u'dressy'", "'dressy'"));
UPDATE business_attribute SET val = "formal"
WHERE (attribute = "restaurants_attire" AND val IN("u'formal'", "'formal'"));

-- business_attribute.noise_level
UPDATE business_attribute SET val = "average"
WHERE (attribute = "noise_level" AND val IN("u'average'", "'average'"));
UPDATE business_attribute SET val = "loud"
WHERE (attribute = "noise_level" AND val IN("u'loud'", "'loud'"));
UPDATE business_attribute SET val = "quiet"
WHERE (attribute = "noise_level" AND val IN("u'quiet'", "'quiet'"));
UPDATE business_attribute SET val = "very_loud" WHERE (attribute = "noise_level" AND val
IN("u'very_loud'", "'very_loud'"));
```

```
-- business_attribute.smoking
UPDATE business_attribute SET val = "no"
WHERE (attribute = "smoking" AND val IN("u'no'", "'no'"));
UPDATE business_attribute SET val = "outdoor"
WHERE (attribute = "smoking" AND val IN("u'outdoor'", "'outdoor'"));
UPDATE business_attribute SET val = "yes"
WHERE (attribute = "smoking" AND val IN("u'yes'", "'yes'"));

-- business_attribute.byob_corkage
UPDATE business_attribute SET val = "no"
WHERE (attribute = "byob_corkage" AND val IN("u'no'", "'no'"));
UPDATE business_attribute SET val = "yes_corkage"
WHERE (attribute = "byob_corkage" AND val IN("u'yes_corkage'", "'yes_corkage'"));
UPDATE business_attribute SET val = "yes_free"
WHERE (attribute = "byob_corkage" AND val IN("u'yes_free'", "'yes_free'"));

-- business_attribute.ages_allowed
UPDATE business_attribute SET val = "18plus"
WHERE (attribute = "ages_allowed" AND val IN("u'18plus'", "'18plus'"));
UPDATE business_attribute SET val = "19plus"
WHERE (attribute = "ages_allowed" AND val IN("19plus'", "'19plus'"));
UPDATE business_attribute SET val = "21plus"
WHERE (attribute = "ages_allowed" AND val IN("u'21plus'", "'21plus'"));
UPDATE business_attribute SET val = "allages"
WHERE (attribute = "ages_allowed" AND val IN("u'allages'", "'allages'"));
```

Finalmente, uma vez que já não precisamos da tabela business_attribute_inicial, eliminamos a mesma:

```
drop table if exists business_attribute_inicial;
```

Business categories

Inicialmente vamos importar para a tabela *business_categories* (ver Anexo IV – DDL Criação Estrutura BD relacional) o *business_id* e a descrição da categoria. Posteriormente vamos substituir a descrição da categoria pelo código da categoria.

Após termos criado as tabelas, importámos o ficheiro *yelp_business.csv* para a tabela *business_category* com o seguinte comando:

```
> LOAD DATA INFILE yelp_business.csv' INTO TABLE ‘business_category’
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\r\n';
```

Para obtermos as categorias existentes, com a intenção de posteriormente preenchermos a tabela category, criámos uma *view* com os valores distintos das categorias usando *distinct*.

```
CREATE VIEW category_view (Category_temp) AS
     SELECT DISTINCT Category_temp
     FROM business_category;
```

De seguida introduzimos a coluna C*ategory_temp* da *view* na tabela *category*:

```
INSERT INTO category.Description
SELECT Category_temp FROM category_view;
```

Adicionámos uma coluna *category_id* na tabela *business_category* onde vão estar os ids das categorias e preenchemo-la de acordo com os ids das categorias de negócio correspondentes da tabela *category*. Aproveitámos também para apagar a coluna temporária Category_temp.

```
ALTER TABLE ‘Business_Category’ ADD ‘category_id’ INT NOT NULL;

UPDATE business_category, category
SET business_category.category_id=category.Category_id
WHERE business_category.Category_temp=category.Description;

ALTER TABLE business_category
DROP COLUMN Category_temp;
```

O próximo passo foi criar a chave estrangeira *category_id* da tabela *business_category*:

```
ALTER TABLE business_category
ADD FOREIGN KEY (category_id) REFERENCES category(category_id)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Ao tentar tornar as colunas *business_id* e *category_id* Primary Key, ocorreu um erro por causa da existência de chaves duplicadas. Para perceber a dimensão dos mesmos realizamos o seguinte comando (ver em Anexo VIII – Problemas e tratamento de duplicação da categoria “Gas Station” o resultado):

```
SELECT business_id, category_id, COUNT(*)
FROM business_category
GROUP BY business_id, category_id
HAVING COUNT(*) > 1
```

Observámos que os casos duplicados pertenciam apenas a negócios com `category_id = 329` que corresponde à categoria “Gas Stations”. É possível verificar que este erro tinha origem na duplicação da categoria nos dados originais, conforme o exemplo abaixo:

```
"business_id": "PSrITKG8En8mvwZ_YhywTg",
"categories": "Automotive, Gas Stations, Coffee & Tea, Service Stations, Gas Stations,
Convenience Stores, Food",…
```

Dado que os casos em questão não eram muitos optámos por apagar as entradas uma a uma com o seguinte comando para cada *business_id*:

```
DELETE FROM business_category
WHERE Business_id = "PSrITKG8En8mvwZ_YhywTg" AND category_id = 329
LIMIT 1;
```

Depois da eliminação destas entradas foi possível criar a *Primary Key* da tabela *business_category* com as colunas *business_id* e *category_id*:

```
ALTER TABLE business_category
ADD PRIMARY KEY (business_id, category_id)
```

Registos órfãos

Terminada a fase de importação e ajuste, fomos verificar se existiam alguns registos órfão na tabela de *friends* e *reviews*.

Para a tabela *friends* os amigos que não existem na tabela de *users* recorrendo à instrução seguinte:

```
SELECT COUNT(*)
FROM friend
WHERE NOT EXISTS(
    SELECT *
    FROM `user`
    WHERE user_id=friend.friend_id
    );
```

O número de registos órfãos corresponde ao número total de registos na tabela, pelo que concluímos que na base de dados não está nenhum utilizador relacionado, seja por uma questão de privacidade, para dificultar a re-identificação de utilizadores, ou porque trazer todos os amigos implicaria um número muito maior de registos de utilizadores no *dataset*.

No caso das *reviews*, elas estão todas relacionadas com business e *user* que estão na base de dados, não existindo por isso nenhuma órfã.

Desafios

Alguns desafios com que nos deparamos no decorrer do trabalho.

Mapeamento de dados não relacionais para modelo relacional

Como referido anteriormente, no caso dos atributos, a dificuldade foi em mapear um modelo não relacional para um modelo relacional, uma vez que não existe explicitamente a designação dos próprios campos, apenas os valores a que teoricamente correspondem. Desta forma, tivemos de encontrar uma solução alternativa que nos permitisse obter uma listagem de todos os atributos implicitamente usados.

Dados locais vs base de dados na nuvem

Para carregar dados de um ficheiro CSV para o MySQL é normal usar o comando `LOAD DATA INFILE`. No entanto, isso pressupõe que o próprio servidor MySQL tem acesso a esses ficheiros. No caso de servidores na nuvem, para ficheiros que estão no computador local, torna-se necessário correr o comando localmente, com indicação que o ficheiro CSV se encontra no cliente e não no servidor. Para isso basta acrescentar a palavra-chave "LOCAL" à instrução habitual: `LOAD DATA **LOCAL** INFILE`.

Grande volume do ficheiro CSV

No caso das tabelas *friends* (4,47 GB) e *reviews* (5,16 GB), temos um volume muito grande de informação (muitos registos) e as ferramentas que dispomos acabam por não conseguir importar um ficheiro completo para a tabela, pois a conexão ao servidor na *cloud* acaba por se perder `(ERROR 2013 (HY000): Lost connection to MySQL server during query)`. Para mitigar este problema esses ficheiros CSV foram partidos em ficheiros mais pequenos (ver Anexo IX – Script de Powershell usado para dividir ficheiros CSV muito extensos) que foram importados individualmente. No nosso caso o ponto em que conseguimos que funcionasse foi cada ficheiro ter cerca de 2 500 000 registos/linhas.

Problemas na Utilização do Powershell

Um outro problema que enfrentámos foi que, ao usar a linha de comando de powershell, em vez da nativa do Windows (por omissão é essa a usada no Windows Terminal), os ficheiros de texto/CSV criados por redirecionar o resultado de um comando `mongo` com uma *query* complexa para um ficheiro, gerava ficheiros em formato UTF16 ou Unicode, que não são suportados pela instrução do MySQL `LOAD`

`DATA LOCAL INFILE` (Oracle Corporation, 2020). Para mitigar o problema usámos a linha de comando normal do Windows (*Command Prompt*).

Referências Bibliográficas

Maypop Inc. (2020). *Variety, a Schema Analyzer for MongoDB*. Obtido de GitHub: https://github.com/variety/variety

MongoDB, Inc. (2020). *A Note On Unicode Strings*. Obtido de PyMongo 3.11.2 documentation: https://pymongo.readthedocs.io/en/stable/tutorial.html#a-note-on-unicode-strings

Oracle Corporation. (2020). *13.2.7 LOAD DATA Statement*. Obtido de MySQL 8.0 Reference Manual: https://dev.mysql.com/doc/refman/8.0/en/load-data.html

Smet, V. D. (10 de Feb de 2015). *How can I split a text file using PowerShell?* Obtido de Stack Overflow: https://stackoverflow.com/a/28432606/2160637

Yelp, Inc. (26 de March de 2020). *Yelp Dataset*. Obtido de Kaggle: https://www.kaggle.com/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_business.json

ANEXO I – Script Exploração de dados

```
// Exploração básica dos dados:

use yelpdb
show collections
db.yelp_business.count()
db.yelp_business.find().limit(1).pretty()


db.yelp_business.distinct("business_id").length
db.yelp_business.count()

db.yelp_business.distinct("name") // nome dos negócios não estão uniformizados
db.yelp_business.find({"name":''},{}).pretty() // exemplo de negócio sem nome

db.yelp_business.distinct("city") // nomes das cidades não estão todos uniformizados
db.yelp_business.find({city:""},{city:1}).sort({city:1}).limit(10) // city tem valores em
branco

db.yelp_business.distinct("state") // verifica-se que não existem estados com mais que 3
caracteres
// ["AB", "AK", "AL", "AR", "AZ", "BC", "CA", "CO", "CT", "DOW", "DUR", "FL", "GA", "HI",
"HPL", "IL", "MB", "MI", "MO", "NC", "NE", "NV", "NY", "OH", "ON", "OR", "PA", "QC", "SC",
"TX", "UT", "VA", "VT", "WA", "WI", "XWY", "YT"]

db.yelp_business.find({},{postal_code:1}).sort({postal_code:-1}).limit(5)
db.yelp_business.find({},{postal_code:1}).sort({postal_code:1}).limit(10) // códigos postais
têm valores em branco

db.yelp_business.find({},{address:1}).sort({address:1}).limit(10) // existem addresses em
branco
db.yelp_business.find({"address":''},{}).limit(2).pretty() // exemplo de address em branco

db.yelp_business.aggregate([{$group:{_id:{latitude:"$latitude",longitude:"$longitude"},count:{
$sum:1}}},{$match:{count:{$gt:1}}}]) // negócios com a mesma localização (lat,long)
db.yelp_business.find({"latitude":36.1319001, "longitude":-115.1864148},{}).pretty() //
exemplo de negócios numa lat,long específica
db.yelp_business.find({$or:[{"latitude":{$exists:false}},{"longitude":{$exists:false}}]},{}).c
ount() // não existem null em lat,long

db.yelp_business.find({"stars":{$exists:false}},{}).count() // não existem valores em branco
no campo "stars"
db.yelp_business.distinct("stars") // [ 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5 ]

db.yelp_business.find({"is_open":{$exists:false}},{}).count() // não existem valores em branco
neste campo
db.yelp_business.distinct("is_open") // [ 0, 1 ]

// dos campos que a tabela original business tem, não consideramos para importação o campo
"review_count" que não é mais que uma contagem do número de avaliações que cada negócio tem;
não consideramos importante dedicar um campo para uma contagem que no modelo relacional é
obtida facilmente com um comando SELECT simples à tabela review, eg:
SELECT COUNT(*) FROM Review WHERE business_id="/*Introduzir business_id*/"; // para obter o
review_count relativo a um business_id específico
//ou
```

```
SELECT business_id, COUNT(*) as Review_count FROM Review GROUP BY business_id; // para obter
uma listagem de todos os review_count para todos os business_id
//entre outros

// Levantamento de valores em branco por campo:

db.yelp_business.find({$or:[
    {"business_id":"None"},
    {"business_id":"none"},
    {"business_id":" "},
    {"business_id":""},
    {"business_id":"'none"},
    {"business_id":{$exists:false}}
]},{}).count()  // 0

db.yelp_business.find({$or:[
    {"name":"None"},
    {"name":"none"},
    {"name":" "},
    {"name":""},
    {"name":"'none"},
    {"name":{$exists:false}}
]},{}).count()  // 1

db.yelp_business.find({$or:[
    {"address":"None"},
    {"address":"none"},
    {"address":" "},
    {"address":""},
    {"address":"'none"},
    {"address":{$exists:false}}
]},{}).count() // 8679

db.yelp_business.find({$or:[
    {"city":"None"},
    {"city":"none"},
    {"city":" "},
    {"city":""},
    {"city":"'none"},
    {"city":{$exists:false}}
]},{}).count() // 2

db.yelp_business.find({$or:[
    {"state":"None"},
    {"state":"none"},
    {"state":" "},
    {"state":""},
    {"state":"'none"},
    {"state":{$exists:false}}
]},{}).count() // 0

db.yelp_business.find({$or:[
    {"postal_code":"None"},
    {"postal_code":"none"},
    {"postal_code":" "},
    {"postal_code":""},
    {"postal_code":"'none"},
```

```
    {"postal_code":{$exists:false}}
]},{}).count() // 509

db.yelp_business.find({$or:[
    {"latitude":"None"},
    {"latitude":"none"},
    {"latitude":" "},
    {"latitude":""},
    {"latitude":"'none"},
    {"latitude":{$exists:false}}
]},{}).count() // 0

db.yelp_business.find({$or:[
    {"longitude":"None"},
    {"longitude":"none"},
    {"longitude":" "},
    {"longitude":""},
    {"longitude":"'none"},
    {"longitude":{$exists:false}}
]},{}).count() // 0

db.yelp_business.find({$or:[
    {"stars":"None"},
    {"stars":"none"},
    {"stars":" "},
    {"stars":""},
    {"stars":"'none"},
    {"stars":{$exists:false}}
]},{}).count() // 0

db.yelp_business.find({$or:[
    {"is_open":"None"},
    {"is_open":"none"},
    {"is_open":" "},
    {"is_open":""},
    {"is_open":"'none"},
    {"is_open":{$exists:false}}
]},{}).count() // 0
```

Anexo II - Comando para correr ferramenta "variety"

```
:: ferramenta que oferece uma listagem dos objectos que podem ser encontrados em
yelp_business.attributes
mongo yelpdb --eval "var collection = 'yelp_business'" variety.js
```

Anexo III – Levantamento dos outputs possíveis para cada "attribute"

```
db.yelp_business.distinct("attributes.BusinessAcceptsCreditCards").length
db.yelp_business.distinct("attributes.BusinessAcceptsCreditCards")

db.yelp_business.distinct("attributes.BusinessParking").length //89
db.yelp_business.distinct("attributes.BusinessParking")
```

```
db.yelp_business.distinct("attributes.RestaurantsPriceRange2") //[ '1', '2', '3', '4', 'None'
]

db.yelp_business.distinct("attributes.BikeParking") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.GoodForKids") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.RestaurantsTakeOut") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.WiFi") // ["'free'",  "'no'", "'paid'",  'None',
"u'free'", "u'no'", "u'paid'"]

db.yelp_business.distinct("attributes.ByAppointmentOnly") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.OutdoorSeating") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.RestaurantsDelivery") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.RestaurantsGoodForGroups") // [ 'False', 'None', 'True'
]

db.yelp_business.distinct("attributes.RestaurantsReservations") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.Ambience").length // 1158

db.yelp_business.distinct("attributes.HasTV") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.Alcohol") // ["'beer_and_wine'", "'full_bar'", "'none'",
'None', "u'beer_and_wine'", "u'full_bar'", "u'none'"]

db.yelp_business.distinct("attributes.RestaurantsAttire") // ["'casual'" "'dressy'",
"'formal'", 'None', "u'casual'", "u'dressy'", "u'formal'"]

db.yelp_business.distinct("attributes.NoiseLevel") // ["'average'", "'loud'","'quiet'",
"'very_loud'", 'None', "u'average'", "u'loud'", "u'quiet'","u'very_loud'"]

db.yelp_business.distinct("attributes.Caters") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.GoodForMeal").length // 282

db.yelp_business.distinct("attributes.WheelchairAccessible") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.RestaurantsTableService") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.DogsAllowed") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.BusinessAcceptsBitcoin") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.HappyHour") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.AcceptsInsurance") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.Music").length // 108

db.yelp_business.distinct("attributes.BestNights").length // 65

db.yelp_business.distinct("attributes.GoodForDancing") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.CoatCheck") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.DriveThru") // [ 'False', 'None', 'True' ]
```

```
db.yelp_business.distinct("attributes.Smoking") // ["'no'", "'outdoor'", "'yes'",'None',
"u'no'", "u'outdoor'", "u'yes'"]

db.yelp_business.distinct("attributes.BYOBCorkage") // ["'no'", "'yes_corkage'",
"'yes_free'",'None',"u'no'","u'yes_corkage'","u'yes_free'"]

db.yelp_business.distinct("attributes.HairSpecializesIn").length // 135

db.yelp_business.distinct("attributes.Corkage") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.BYOB") // [ 'False', 'None', 'True' ]

db.yelp_business.distinct("attributes.AgesAllowed") // [ 'None', "u'18plus'", "u'19plus'",
"u'21plus'", "u'allages'" ]

db.yelp_business.distinct("attributes.DietaryRestrictions").length // 12

db.yelp_business.distinct("attributes.Open24Hours") // [ 'False', 'True' ]

db.yelp_business.distinct("attributes.RestaurantsCounterService") // [ 'False', 'True' ]

//destes atributos, decidiu-se não considerar para importação em SQL os seguintes:
//attributes.BusinessParking
//attributes.Ambience
//attributes.GoodForMeal
//attributes.Music
//attributes.BestNights
//attributes.HairSpecializesIn
//attributes.DietaryRestrictions
```

Anexo IV – DDL Criação Estrutura BD relacional

```
CREATE TABLE business
(
   business_id VARCHAR(24) NOT NULL,
   name VARCHAR(100) NULL,
   address VARCHAR(200) NULL,
   city VARCHAR(100) NULL,
   state VARCHAR(5) NULL,
   postal_code VARCHAR(15) NULL,
   latitude FLOAT NOT NULL,
   longitude FLOAT NOT NULL,
   stars FLOAT NOT NULL,
   is_open BOOLEAN NOT NULL,

   CONSTRAINT PK_business PRIMARY KEY (business_id)
);

CREATE TABLE attribute
(
   attribute VARCHAR(40) NOT NULL,


   CONSTRAINT PK_attribute PRIMARY KEY (attribute)
);

CREATE TABLE business_attribute
(
   business_id VARCHAR(24) NOT NULL,
```

```
   attribute VARCHAR(40) NOT NULL,
   val VARCHAR(30) NOT NULL,

   CONSTRAINT PK_business_attribute PRIMARY KEY (business_id, attribute)
);


CREATE TABLE Category
(category_id INT NOT NULL AUTO_INCREMENT,
  Description varchar(50)  NOT NULL,
  PRIMARY KEY (category_id)
);

CREATE TABLE IF NOT EXISTS `Business_Category` (
    `business_id` VARCHAR(24) NOT NULL,
    `Category_temp` VARCHAR(50) NOT NULL,
 );

CREATE TABLE IF NOT EXISTS `Business_Hours` (
    `business_id` VARCHAR(24) NOT NULL,
    `Monday` VARCHAR(11) NULL,
    `Tuesday` VARCHAR(11) NULL,
    `Wednesday` VARCHAR(11) NULL,
    `Thursday` VARCHAR(11) NULL,
    `Friday` VARCHAR(11) NULL,
    `Saturday` VARCHAR(11) NULL,
    `Sunday` VARCHAR(11) NULL,
    CONSTRAINT PK_Business_Hours PRIMARY KEY (business_id)
 );

CREATE TABLE IF NOT EXISTS `user` (
    user_id VARCHAR(24) PRIMARY KEY,
    `name` VARCHAR(50) NULL,
    review_count INT NULL,
    yelping_since DATETIME NULL,
    useful INT NULL,
    funny INT NULL,
    cool INT NULL,
    elite VARCHAR(100) NOT NULL DEFAULT '',
    fans INT NULL,
    average_stars DECIMAL(5,2) NULL,
    compliment_hot INT NULL,
    compliment_more INT NULL,
    compliment_profile INT NULL,
    compliment_cute INT NULL,
    compliment_list INT NULL,
    compliment_note INT NULL,
    compliment_plain INT NULL,
    compliment_cool INT NULL,
    compliment_funny INT NULL,
    compliment_writer INT NULL,
    compliment_photos INT NULL
);

CREATE TABLE IF NOT EXISTS friend (
    user_id VARCHAR(24) NOT NULL,
    friend_id VARCHAR(24) NOT NULL,
    PRIMARY KEY (user_id, friend_id)
);
```

```
CREATE TABLE IF NOT EXISTS review (
    review_id VARCHAR(24) NOT NULL,
    user_id VARCHAR(24) NOT NULL,
    business_id VARCHAR(24) NOT NULL,
    stars TINYINT NOT NULL,
    useful INT NOT NULL,
    funny INT NOT NULL,
    cool INT NOT NULL,
    `text` TEXT NULL,
    `date` DATETIME NOT NULL,
    PRIMARY KEY (review_id)
);

-- Definir ligações entre tabela business e tabela business_attribute
-- A realizar apenas após a importação de dados estra terminada
ALTER TABLE business_attribute
    ADD CONSTRAINT FK_business_attribute_business FOREIGN KEY (business_id)
    REFERENCES business(business_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
;

ALTER TABLE business_attribute
    ADD CONSTRAINT FK_business_attribute_business_attribute_set FOREIGN KEY (attribute)
    REFERENCES business_attribute_set(attribute)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
;

ALTER TABLE business_category
    ADD FOREIGN KEY (category_id) REFERENCES category(category_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
;

ALTER TABLE friend
    ADD FOREIGN KEY (user_id) REFERENCES user(user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
;

/* Não é possível correr, uma vez que os amigos não foram incluidos no dataset
ALTER TABLE friend
    ADD CONSTRAINT FK_friend_friend_user_id FOREIGN KEY (friend_id) REFERENCES user(user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE;
*/

ALTER TABLE review
    ADD FOREIGN KEY (user_id) REFERENCES user(user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
;

ALTER TABLE review
    ADD FOREIGN KEY (business_id) REFERENCES business(business_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
;
```

Anexo V - Inserir os dados na tabela *attribute*

```
INSERT INTO attribute (attribute) VALUES ("business_accepts_credit_cards");
INSERT INTO attribute (attribute) VALUES ("restaurants_price_range2");
INSERT INTO attribute (attribute) VALUES ("bike_parking");
INSERT INTO attribute (attribute) VALUES ("good_for_kids");
INSERT INTO attribute (attribute) VALUES ("restaurants_take_out");
INSERT INTO attribute (attribute) VALUES ("wifi");
INSERT INTO attribute (attribute) VALUES ("by_appointment_only");
INSERT INTO attribute (attribute) VALUES ("outdoor_seating");
INSERT INTO attribute (attribute) VALUES ("restaurants_delivery");
INSERT INTO attribute (attribute) VALUES ("restaurants_good_for_groups");
INSERT INTO attribute (attribute) VALUES ("restaurants_reservations");
INSERT INTO attribute (attribute) VALUES ("has_tv");
INSERT INTO attribute (attribute) VALUES ("alcohol");
INSERT INTO attribute (attribute) VALUES ("restaurants_attire");
INSERT INTO attribute (attribute) VALUES ("noise_level");
INSERT INTO attribute (attribute) VALUES ("caters");
INSERT INTO attribute (attribute) VALUES ("wheelchair_accessible");
INSERT INTO attribute (attribute) VALUES ("restaurants_table_service");
INSERT INTO attribute (attribute) VALUES ("dogs_allowed");
INSERT INTO attribute (attribute) VALUES ("business_accepts_bitcoin");
INSERT INTO attribute (attribute) VALUES ("happy_hour");
INSERT INTO attribute (attribute) VALUES ("accepts_insurance");
INSERT INTO attribute (attribute) VALUES ("good_for_dancing");
INSERT INTO attribute (attribute) VALUES ("coat_check");
INSERT INTO attribute (attribute) VALUES ("drive_thru");
INSERT INTO attribute (attribute) VALUES ("smoking");
INSERT INTO attribute (attribute) VALUES ("byob_corkage");
INSERT INTO attribute (attribute) VALUES ("corkage");
INSERT INTO attribute (attribute) VALUES ("byob");
INSERT INTO attribute (attribute) VALUES ("ages_allowed");
INSERT INTO attribute (attribute) VALUES ("open_24hours");
INSERT INTO attribute (attribute) VALUES ("restaurants_counter_service");
```

Anexo VI - Transformar e migrar os dados da tabela business_attribute_inicial para a business_attribute, desconsiderando Nulls e campos vazios

```
-- attribute 1
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "business_accepts_credit_cards", business_accepts_credit_cards
FROM business_attribute_inicial
WHERE length(business_accepts_credit_cards) > 0
AND business_accepts_credit_cards != "None"
;

-- attribute 2
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "restaurants_price_range2", restaurants_price_range2
FROM business_attribute_inicial
WHERE length(restaurants_price_range2) > 0
AND restaurants_price_range2 != "None"
;

-- attribute 3
INSERT INTO business_attribute (business_id, attribute, val)
```

```
SELECT business_id, "bike_parking", bike_parking
FROM business_attribute_inicial
WHERE length(bike_parking) > 0
AND bike_parking != "None"
;

-- attribute 4
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "good_for_kids", good_for_kids
FROM business_attribute_inicial
WHERE length(good_for_kids) > 0
AND good_for_kids != "None"
;

-- attribute 5
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "restaurants_take_out", restaurants_take_out
FROM business_attribute_inicial
WHERE length(restaurants_take_out) > 0
AND restaurants_take_out != "None"
;

-- attribute 6
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "wifi", wifi
FROM business_attribute_inicial
WHERE length(wifi) > 0
AND wifi != "None"
;

-- attribute 7
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "by_appointment_only", by_appointment_only
FROM business_attribute_inicial
WHERE length(by_appointment_only) > 0
AND by_appointment_only != "None"
;

-- attribute 8
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "outdoor_seating", outdoor_seating
FROM business_attribute_inicial
WHERE length(outdoor_seating) > 0
AND outdoor_seating != "None"
;

-- attribute 9
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "restaurants_delivery", restaurants_delivery
FROM business_attribute_inicial
WHERE length(restaurants_delivery) > 0
AND restaurants_delivery != "None"
;

-- attribute 10
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "restaurants_good_for_groups", restaurants_good_for_groups
FROM business_attribute_inicial
WHERE length(restaurants_good_for_groups) > 0
AND restaurants_good_for_groups != "None"
;
```

```
-- attribute 11
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "restaurants_reservations", restaurants_reservations
FROM business_attribute_inicial
WHERE length(restaurants_reservations) > 0
AND restaurants_reservations != "None"
;

-- attribute 12
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "has_tv", has_tv
FROM business_attribute_inicial
WHERE length(has_tv) > 0
AND has_tv != "None"
;

-- attribute 13
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "alcohol", alcohol
FROM business_attribute_inicial
WHERE length(alcohol) > 0
AND alcohol != "None"
    AND alcohol != "'none'"
AND alcohol != "u'none'"
;

-- attribute 14
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "restaurants_attire", restaurants_attire
FROM business_attribute_inicial
WHERE length(restaurants_attire) > 0
AND restaurants_attire != "None"
;

-- attribute 15
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "noise_level", noise_level
FROM business_attribute_inicial
WHERE length(noise_level) > 0
AND noise_level != "None"
;

-- attribute 16
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "caters", caters
FROM business_attribute_inicial
WHERE length(caters) > 0
AND caters != "None"
;

-- attribute 17
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "wheelchair_accessible", wheelchair_accessible
FROM business_attribute_inicial
WHERE length(wheelchair_accessible) > 0
AND wheelchair_accessible != "None"
;

-- attribute 18
INSERT INTO business_attribute (business_id, attribute, val)
```

```
SELECT business_id, "restaurants_table_service", restaurants_table_service
FROM business_attribute_inicial
WHERE length(restaurants_table_service) > 0
AND restaurants_table_service != "None"
;

-- attribute 19
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "dogs_allowed", dogs_allowed
FROM business_attribute_inicial
WHERE length(dogs_allowed) > 0
AND dogs_allowed != "None"
;

-- attribute 20
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "business_accepts_bitcoin", business_accepts_bitcoin
FROM business_attribute_inicial
WHERE length(business_accepts_bitcoin) > 0
AND business_accepts_bitcoin != "None"
;

-- attribute 21
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "happy_hour", happy_hour
FROM business_attribute_inicial
WHERE length(happy_hour) > 0
AND happy_hour != "None"
;

-- attribute 22
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "accepts_insurance", accepts_insurance
FROM business_attribute_inicial
WHERE length(accepts_insurance) > 0
AND accepts_insurance != "None"
;

-- attribute 23
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "good_for_dancing", good_for_dancing
FROM business_attribute_inicial
WHERE length(good_for_dancing) > 0
AND good_for_dancing != "None"
;


-- attribute 24
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "coat_check", coat_check
FROM business_attribute_inicial
WHERE length(coat_check) > 0
AND coat_check != "None"
;

-- attribute 25
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "drive_thru", drive_thru
FROM business_attribute_inicial
WHERE length(drive_thru) > 0
AND drive_thru != "None"
```

```
;

-- attribute 26
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "smoking", smoking
FROM business_attribute_inicial
WHERE length(smoking) > 0
AND smoking != "None"
;

-- attribute 27
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "byob_corkage", byob_corkage
FROM business_attribute_inicial
WHERE length(byob_corkage) > 0
AND byob_corkage != "None"
;

-- attribute 28
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "corkage", corkage
FROM business_attribute_inicial
WHERE length(corkage) > 0
AND corkage != "None"
;

-- attribute 29
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "byob", byob
FROM business_attribute_inicial
WHERE length(byob) > 0
AND byob != "None"
;

-- attribute 30
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "ages_allowed", ages_allowed
FROM business_attribute_inicial
WHERE length(ages_allowed) > 0
AND ages_allowed != "None"
;

-- attribute 31
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "open_24hours", open_24hours
FROM business_attribute_inicial
WHERE length(open_24hours) > 0
;

-- attribute 32
INSERT INTO business_attribute (business_id, attribute, val)
SELECT business_id, "restaurants_counter_service", restaurants_counter_service
FROM business_attribute_inicial
WHERE length(restaurants_counter_service) > 0
;
```

Anexo VII - Definição de Chaves Estrangeiras na tabela Business_attribute

```
alter table business_attribute
    add constraint FK_business_attribute_business foreign key (business_id)
    references business(business_id)
    on delete cascade
    on update cascade
;

alter table business_attribute
    add constraint FK_business_attribute_attribute foreign key (attribute)
    references attribute(attribute)
    on delete restrict
    on update cascade
;
```

Anexo VIII – Problemas e tratamento de duplicação da categoria "Gas Station"

1. Descobrir quais registos têm categorias duplicadas:

```
> SELECT business_id, category_id, COUNT(*)
    FROM business_category
    GROUP BY business_id, category_id
    HAVING COUNT(*) > 1
    ;
+------------------------+-------------+----------+
| business_id            | category_id | COUNT(*) |
+------------------------+-------------+----------+
| HEGy1__jKyMhkhXRW3O1ZQ |         329 |        2 |
| DppMDrrAE3eWkD3JE7gl7g |         329 |        2 |
| PSrITKG8En8mvwZ_YhywTg |         329 |        2 |
| nY9ZcdrkncyVh3ZJ4WvtwA |         329 |        2 |
| KoOqNG0ysV8yChCf5-XE0Q |         329 |        2 |
| 7bpqrYxc8kJb-zkR376RbA |         329 |        2 |
| ApDQuqBm37kISQ0mulaQoQ |         329 |        2 |
| albXOYDgIU1wYqpOrZO7_A |         329 |        2 |
| QwK3g4oA6QIQxAdphgLHGg |         329 |        2 |
| bRMLe--K17Hvvx0Xc7hTPQ |         329 |        2 |
| gOrFC-2-msNhkyhHXH41eA |         329 |        2 |
| sqUsqgzKZPRvc4qdgKNJiQ |         329 |        2 |
| j9UX-w0YQKXY6YuERT6BHA |         329 |        2 |
| CeWu2SuUQBYnS68sgg04wA |         329 |        2 |
| DB7IH44YyZviWx3v_YGeGQ |         329 |        2 |
| q8D-tlj8pQPXAtdUC3409w |         329 |        2 |
| XBlUgpFOvm05WSCgErQ_Ow |         329 |        2 |
| BRySK7Ck7EHZ6YwOFnUI2g |         329 |        2 |
| EmKn83E_Fs73KJ2x7d55UQ |         329 |        2 |
| BzAqjV3mqablP3Bls-rEgQ |         329 |        2 |
| BZEBhwEBY_sLANuWgS2w_Q |         329 |        2 |
| dX7v8PS03axhJfkPdMPEBw |         329 |        2 |
| vNSaUpD6LSqxA93WjXMsKQ |         329 |        2 |
| 1vnnEE4WlCoTuToas5dnUw |         329 |        2 |
| ikQr8ma9SnHIKu-8xZyXWw |         329 |        2 |
| TxYq9LvtvYgi-zfOT6h_rQ |         329 |        2 |
| pohAeCMsGb8p4dv6I_m-Kg |         329 |        2 |
```

```
| bAuqQfgFl7kMqhftvCJcTQ |         329 |         2 |
| 6sBjtav4xHQJQKBuqxQqsQ |         329 |         2 |
| bfMUEkieoeH34V3u0BHrJA |         329 |         2 |
| FszzbwebTzkO8fxgq6H7qA |         329 |         2 |
| m6UyR6yD-GVRkZUOJ6ohxA |         329 |         2 |
| MHFoJxTgeQ7WsJda1Ky_Cg |         329 |         2 |
| y5u2_46X2AucRjr15Ym8cA |         329 |         2 |
| HkmT3UNSG8_xhdbT548PYg |         329 |         2 |
| hwR00bYYMCcrkyqB-tN2Ow |         329 |         2 |
| Shfg3OSN0LuexhugwaVdaA |         329 |         2 |
| VEkVzZhziHZdoUBTTfpb0A |         329 |         2 |
+------------------------+-------------+-----------+
38 rows in set (1.201 sec)
```

2. Apagando os registos duplicados:

```
MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "HEGy1__jKyMhkhXRW3O1ZQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.060 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "DppMDrrAE3eWkD3JE7gl7g" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.074 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "PSrITKG8En8mvwZ_YhywTg" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.071 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "nY9ZcdrkncyVh3ZJ4WvtwA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.050 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "KoOqNG0ysV8yChCf5-XE0Q" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.052 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "7bpqrYxc8kJb-zkR376RbA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.054 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "ApDQuqBm37kISQ0mulaQoQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.054 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "albXOYDgIU1wYqpOrZO7_A" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.076 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "QwK3g4oA6QIQxAdphgLHGg" AND category_id = 329
    -> LIMIT 1;
```

```
Query OK, 1 row affected (0.060 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "bRMLe--K17Hvvx0Xc7hTPQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.055 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "gOrFC-2-msNhkyhHXH41eA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.057 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "sqUsqgzKZPRvc4qdgKNJiQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.053 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "j9UX-w0YQKXY6YuERT6BHA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.057 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "CeWu2SuUQBYnS68sgg04wA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.062 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "DB7IH44YyZviWx3v_YGeGQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.083 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "q8D-tlj8pQPXAtdUC3409w" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.054 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "XBlUgpFOvm05WSCgErQ_Ow" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.055 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "BRySK7Ck7EHZ6YwOFnUI2g" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.069 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "EmKn83E_Fs73KJ2x7d55UQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.059 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "BzAqjV3mqablP3Bls-rEgQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.057 sec)

MySQL [yelp]> DELETE FROM business_category
```

```
    -> WHERE Business_id = "BZEBhwEBY_sLANuWgS2w_Q" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.054 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "dX7v8PS03axhJfkPdMPEBw" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.056 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "vNSaUpD6LSqxA93WjXMsKQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.055 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "1vnnEE4WlCoTuToas5dnUw" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.066 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "ikQr8ma9SnHIKu-8xZyXWw" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.127 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "TxYq9LvtvYgi-zfOT6h_rQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.072 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "pohAeCMsGb8p4dv6I_m-Kg" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.057 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "bAuqQfgFl7kMqhftvCJcTQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.057 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "6sBjtav4xHQJQKBuqxQqsQ" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.078 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "bfMUEkieoeH34V3u0BHrJA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.056 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "FszzbwebTzkO8fxgq6H7qA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.054 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "m6UyR6yD-GVRkZUOJ6ohxA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.072 sec)
```

```
MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "MHFoJxTgeQ7WsJda1Ky_Cg" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.063 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "y5u2_46X2AucRjr15Ym8cA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.055 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "HkmT3UNSG8_xhdbT548PYg" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.060 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "hwR00bYYMCcrkyqB-tN2Ow" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.060 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "Shfg3OSN0LuexhugwaVdaA" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.057 sec)

MySQL [yelp]> DELETE FROM business_category
    -> WHERE Business_id = "VEkVzZhziHZdoUBTTfpb0A" AND category_id = 329
    -> LIMIT 1;
Query OK, 1 row affected (0.054 sec)
```

Anexo IX – Script de Powershell usado para dividir ficheiros CSV muito extensos

```
#split files
$sw = new-object System.Diagnostics.Stopwatch
$sw.Start()
$filename = ".\yelp_reviews.csv"
$rootName = ".\yelp_reviews_"
$ext = "csv"

$linesperFile = 2500000#2.5G
$filecount = 1
$reader = $null
try{
  $reader = [io.file]::OpenText($filename)
  try{
    "Creating file number $filecount"
    $writer = [io.file]::CreateText("{0}{1}.{2}" -f ($rootName,$filecount.ToString("000"),$ext))
    $filecount++
    $linecount = 0

    while($reader.EndOfStream -ne $true) {
      "Reading $linesperFile"
      while( ($linecount -lt $linesperFile) -and ($reader.EndOfStream -ne $true)){
```

```
            $writer.WriteLine($reader.ReadLine());
            $linecount++
        }

        if($reader.EndOfStream -ne $true) {
            "Closing file"
            $writer.Dispose();

            "Creating file number $filecount"
            $writer = [io.file]::CreateText("{0}{1}.{2}" -f ($rootName,$filecount.ToString("000"),$ext))
            $filecount++
            $linecount = 0
        }
      }
    } finally {
      $writer.Dispose();
    }
} finally {
    $reader.Dispose();
}
$sw.Stop()

Write-Host "Split complete in " $sw.Elapsed.TotalSeconds "seconds"
```

Com base em https://stackoverflow.com/questions/1001776/how-can-i-split-a-text-file-using-powershell (Smet, 2015).