



# Data Logic

Esta apresentação demonstrará o projeto de database application, destacando seus objetivos, metodologias e potencial impacto.



[github.com/TulioMendesDev/DataLogic](https://github.com/TulioMendesDev/DataLogic)



# Our Team



**ÂNGELO SANTOS**  
01589358



**JOSÉ MIGUEL**  
01665230



**LIVYA CARVALHO**  
01645272



**THAIS MELO**  
01068175



**TÚLIO MENDES**  
01633581

# Project Overview

Este projeto tem como objetivo **coletar e manipular** o dataset NYC Airbnb Open Data, disponível na plataforma Kaggle. Fazendo gerenciamento em contêiner pelo docker, utilizando bancos SQL e NoSQL. Nossa equipe selecionou a tecnologia **MongoDB** orientado a **documentos**.

New York City Airbnb Open Data  
Airbnb listings and metrics in NYC, NY, USA (2019)

 <https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data>



O dataset contém informações sobre os anfitriões, atividades de listagem, disponibilidade geográfica e métricas do Airbnb em NYC 2019



# Metodologia



## Data Collection

Definição do Dataset e Reunir dados relevantes



## Preprocessing

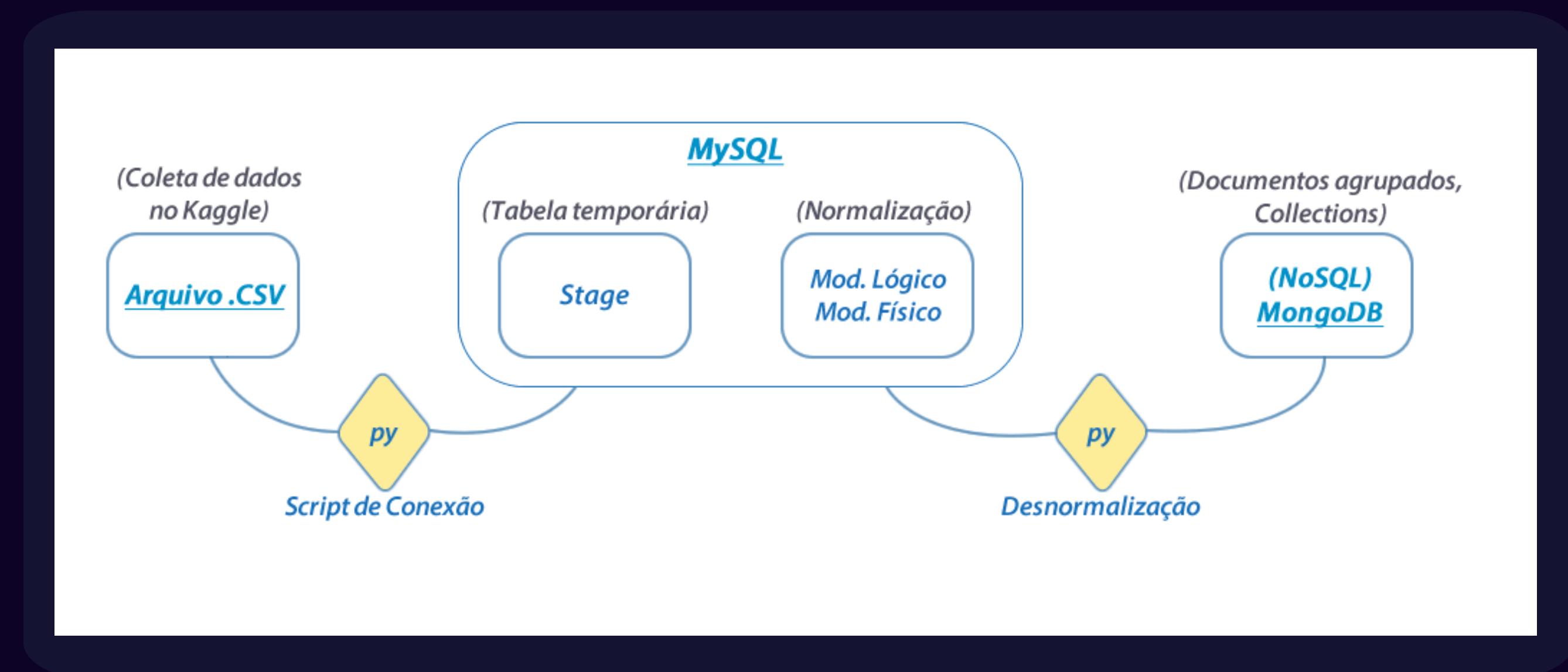
Limpar, normalizar e estruturar os dados



## Model Selection

Fazer passagem de dados  
(SQL para NoSQL)  
MongoDB

# Panorama do Projeto





# Coletando os dados

Após coletar o conjunto de dados (.CSV) na plataforma kaggle, foi realizado uma conversão de texto (" , ") para colunas no Excel, para entender a estrutura e obter uma noção para a normalização.

M15	A	B	C	D	E	F	G	H	I	J	K
1	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights
2	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	4.064.749	-7.397.237	Private room	149	1
3	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	4.075.362	-7.398.377	Entire home	225	1
4	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	4.080.902	-739.419	Private room	150	3
5	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	4.068.514	-7.395.976	Entire home	89	1
6	5022	Entire Apt: Spacious Studio/Loft by centr	7192	Laura	Manhattan	East Harlem	4.079.851	-7.394.399	Entire home	80	10
7	5099	Large Cozy 1 BR Apartment In Midtown E	7322	Chris	Manhattan	Murray Hill	4.074.767	-73.975	Entire home	200	3
8	5121	BlissArtsSpace!	7356	Garon	Brooklyn	Bedford-Stuyvesant	4.068.688	-7.395.596	Private room	60	45
9	5178	Large Furnished Room Near B'way	8967	Shunichi	Manhattan	Hell's Kitchen	4.076.489	-7.398.493	Private room	79	2
10	5203	Cozy Clean Guest Room - Family Apt	7490	MaryEllen	Manhattan	Upper West Side	4.080.178	-7.396.723	Private room	79	2
11	5238	Cute & Cozy Lower East Side 1 bdrm	7549	Ben	Manhattan	Chinatown	4.071.344	-7.399.037	Entire home	150	1
12	5295	Beautiful 1br on Upper West Side	7702	Lena	Manhattan	Upper West Side	4.080.316	-7.396.545	Entire home	135	5
13	5441	Central Manhattan/near Broadway	7989	Kate	Manhattan	Hell's Kitchen	4.076.076	-7.398.867	Private room	85	2
14	5803	Lovely Room 1, Garden, Best Area, Legal	9744	Laurie	Brooklyn	South Slope	4.066.829	-7.398.779	Private room	89	4
15	6021	Wonderful Guest Bedroom in Manhattan	11528	Claudio	Manhattan	Upper West Side	4.079.826	-7.396.113	Private room	85	2

# Docker Compose

Preciamos:

- 1.Criar o Docker Compose no VS Code, para Conectar Dataset com o MySQL;
- 2.Importar ferramentas de conexão, como o mysql-connector-python;
- 3.Importar a biblioteca pandas;
- 4.Import do SQLAlchemy para a criação da Engine;
- 5.Com a Engine criada, utilizar uma string de conexão como parâmetro;
- 6.Por fim, utilizamos o comando ".to\_sql" com parâmetros do nome da DB a ser criada.



# Docker Compose

O Docker compose permite uma execução multi-container de forma simplificada. No arquivo YML, podemos especificar as dependências entre os serviços, as configurações de rede e volumes, facilitando a configuração e garantindo consistência entre diferentes ambientes de desenvolvimento, teste e produção.

```
docker-compose-mysql-mongodb.yml
1 version: '3'
2
3 services:
4   mysql-01:
5     image: mysql:8.0
6     container_name: mysql-01
7     restart: always
8     environment:
9       MYSQL_ROOT_PASSWORD: root
10      MYSQL_DATABASE: datalogic
11      MYSQL_USER: dataloc
12      MYSQL_PASSWORD: datalogic
13     ports:
14       - "3306:3306"
15     volumes:
16       - ./mysql-01:/var/lib/mysql
17   mongodb:
18     image: mongo
19     container_name: mongodb-01
20     restart: always
21     environment:
22       MONGO_INITDB_ROOT_USERNAME: root
23       MONGO_INITDB_ROOT_PASSWORD: rootpw
24     ports:
25       - "27017:27017"
26     volumes:
27       - ./data:/data/db
28
```



# Import de Algumas

```
#Instalando e importando ferramentas de conexão do arquivo Python para o MySQL.
```

```
!pip install mysql-connector-python  
import mysql.connector
```

```
!pip install sqlalchemy  
from sqlalchemy import create_engine
```

```
#Instalando e importando biblioteca Pandas para manipulação dos dados do arquivo .CSV através do Python
```

```
!pip install pandas  
import pandas as pd
```



# Atribuição e Teste de Leitura

```
#Atribuindo leitura do arquivo do dataset csv em um DataFrame, através do pandas, para uma variável
```

```
df_airbnb = pd.read_csv('.\AB_NYC_2019.csv')
```

```
#Teste da leitura
```

```
df_airbnb.head()
```



# Engine e Conexão Com

```
# Início do processo de conexão com o MySQL. Começamos atribuindo a string de conexão a uma variável.  
  
params = "mysql+mysqlconnector://root:root@localhost:3306/datalogic"  
  
# Criamos nossa engine com nossa string de conexão como parâmetro  
  
engine = create_engine(params)  
  
# Finalmente, utilizamos o comando ".to_sql" com parametros do nome da DB a ser criada, nossa engine de conexão,  
# Ação para recorrência e booleano de existência do valor "index"  
  
df_airbnb.to_sql("airbnb_stage",con=engine, if_exists='replace', index=False)
```



# Modelo Relacional



O modelo relacional é utilizado em **ambientes transacionais (OLTP)**, pois são ambientes na qual os principais comandos utilizados são de **inserção e atualizações**, desta forma o modelo é **normalizado** afim de evitar redundâncias de dados; essas relações se dão através de chaves que criam restrições e também gerem o relacionamento entre os atributos (campos) nas entidades (tabelas).



# Contextualizaçā

Como abordado, para iniciar o processo de modelagem para ambientes transacionais (OLTP), **é recomendado** a utilização de três tipos de modelagem que ajuda na abstração do modelo de dados até a entrega do modelo pronto e preparado para implantação em um sistema de Banco de Dados, conforme diagrama a seguir e os passos.

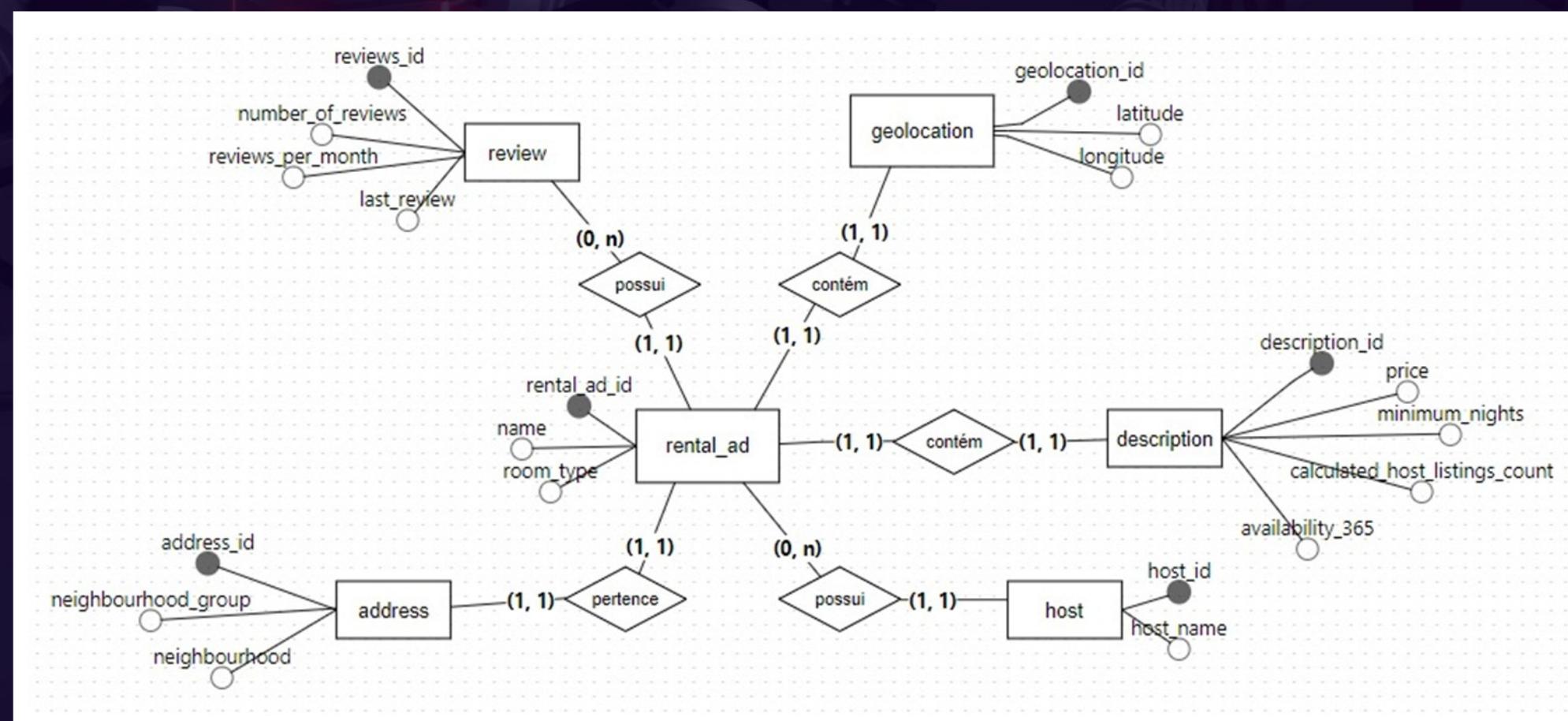


Processo de Modelagem de Dados — Joe Reis & Matt Housley (1<sup>a</sup> Ed.)



# Modelo Conceitual

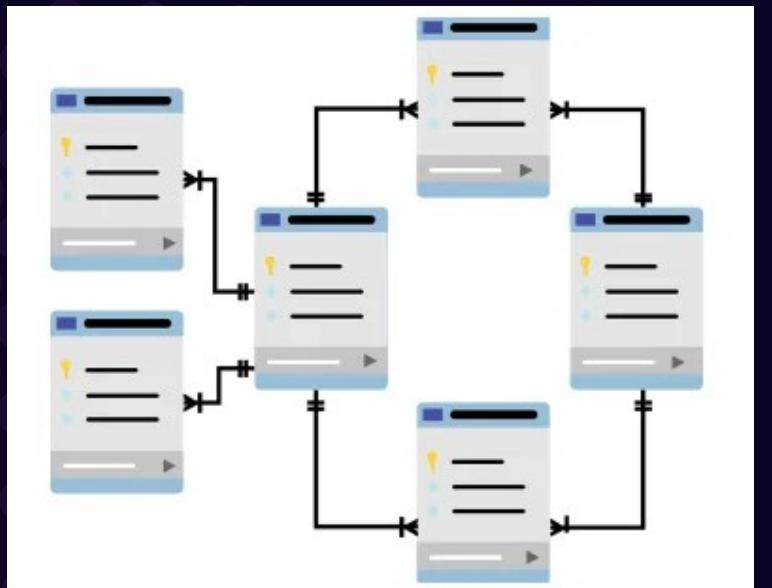
Representa o domínio do negócio em um nível de abstração mais alto e objetivo é descrever as **principais entidades, relacionamentos** e regras de negócio que compõem o domínio do problema.





# Algumas Regras

Para a construção de uma modelagem relacional é necessário seguir os passos de **normalização** que tem por finalidade reduzir as **redundâncias e inconsistências**, bem como garantir a integridade deles. O processo de normalização envolve formas normais (FN) que foi proposto por Boyce-Codd em 1972.



## Normalização

### #1FN

- Cada coluna em uma tabela deve conter apenas valores atômicos (indivisíveis).
- Cada célula da tabela deve conter um único valor, não listas de valores separados por vírgula ou outro delimitador.

### #2FN

- A tabela deve estar na 1NF.
- Todos os atributos não-chave devem depender completamente da chave primária. Em outras palavras, não deve haver dependência parcial da chave primária.

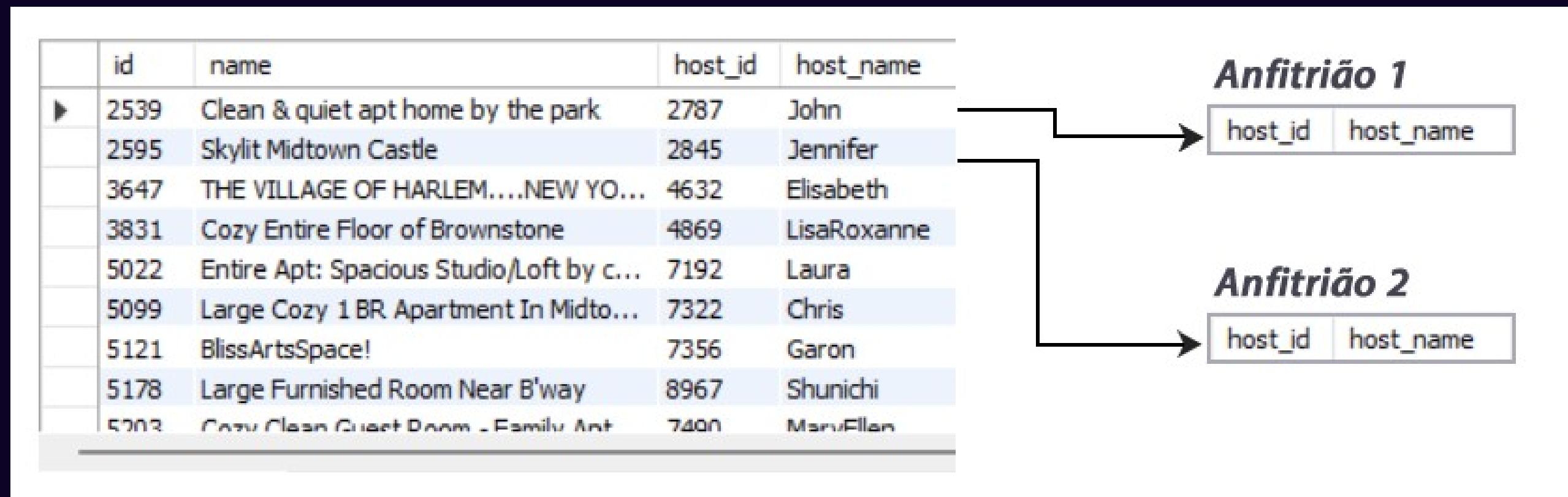
### #3FN

- A tabela deve estar na 2NF.
- Baseia-se em dependência transitiva: Identificar cada grupo de atributos não-chave que dependam de outros atributos não-chave;
- Criar uma tabela para armazenar os atributos (ou conjunto de atributos) não-chave que não estão relacionados à chave primária da tabela original.



## Primeira Forma Normal (1FN):

- Todos os valores são atômicos (indivisíveis);
- Há apenas um dado por coluna;
- Existe uma chave primária (pelo menos);
- Não há relações aninhadas (tabelas dentro de tabelas).

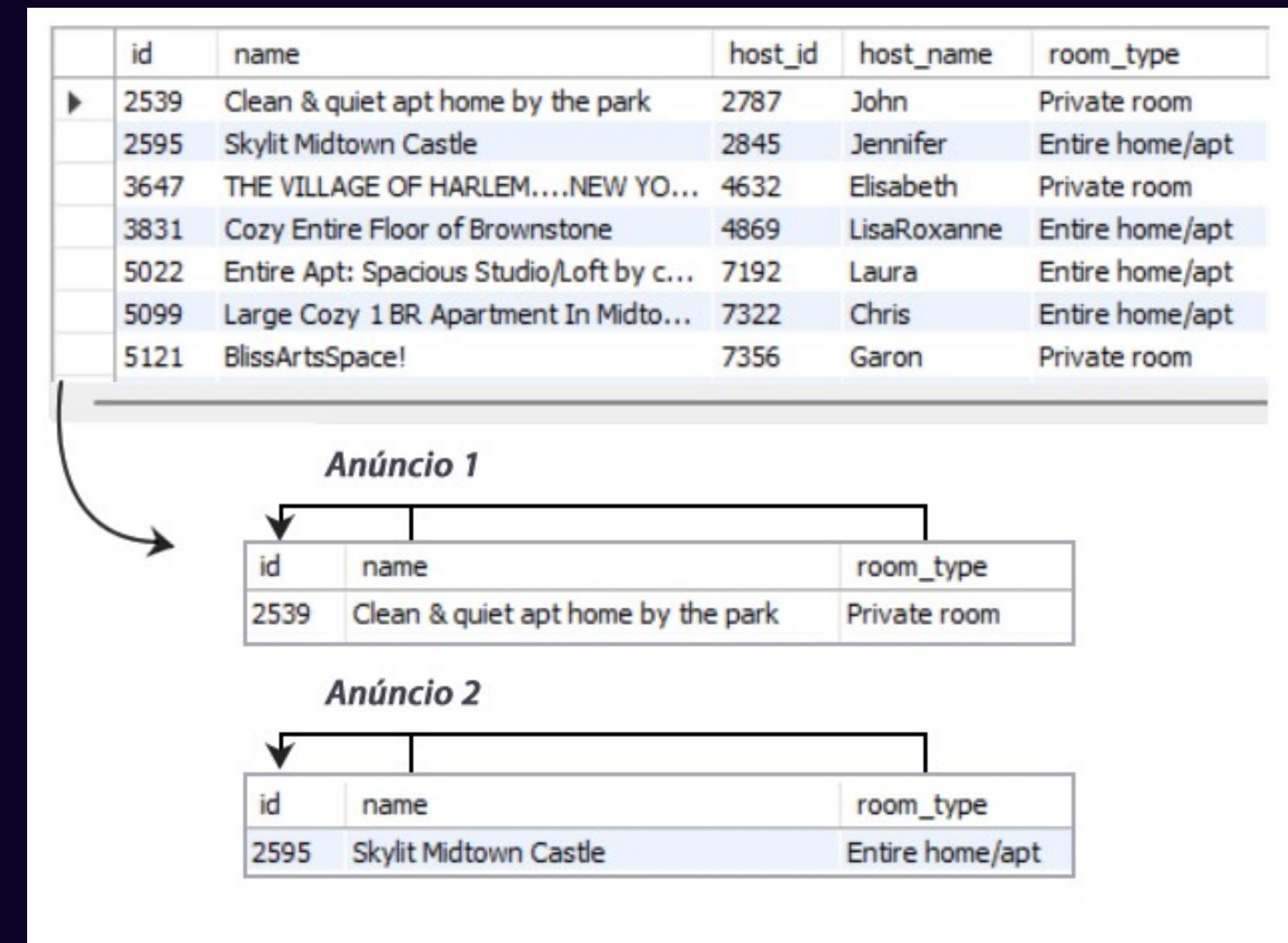


Processo de Normalização em 1FN. Adaptado de Sistema de Bancos de Dados – Navathe (6<sup>a</sup> Ed)



## Segunda Forma Normal (2FN):

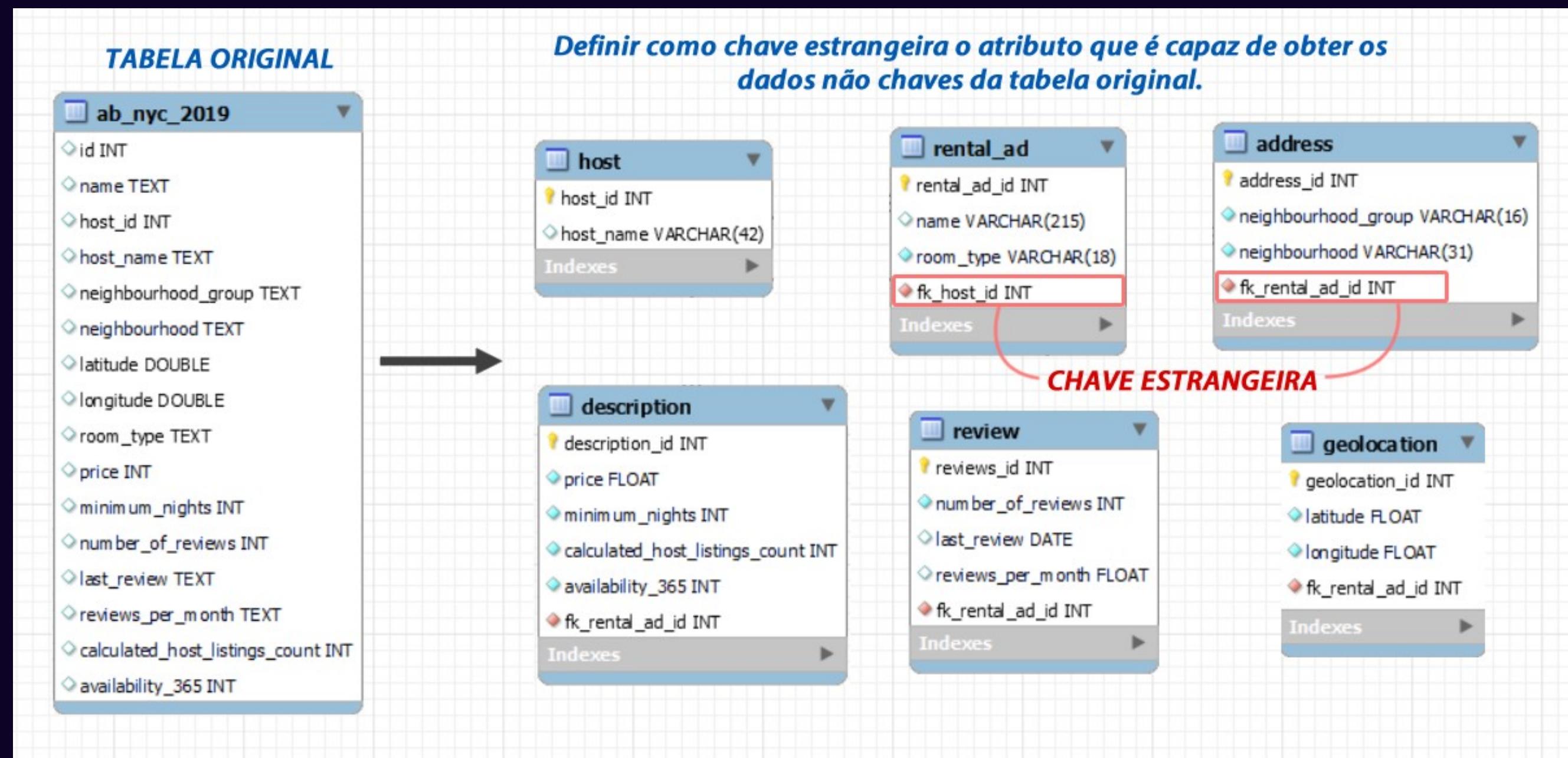
- Está na 1FN;
- Não contém dependências parciais;
- Os atributos (campos) de uma entidade (tabela) tenham campos dependentes a sua chave primária.





## Terceira Forma Normal (3FN): - Está na 2FN;

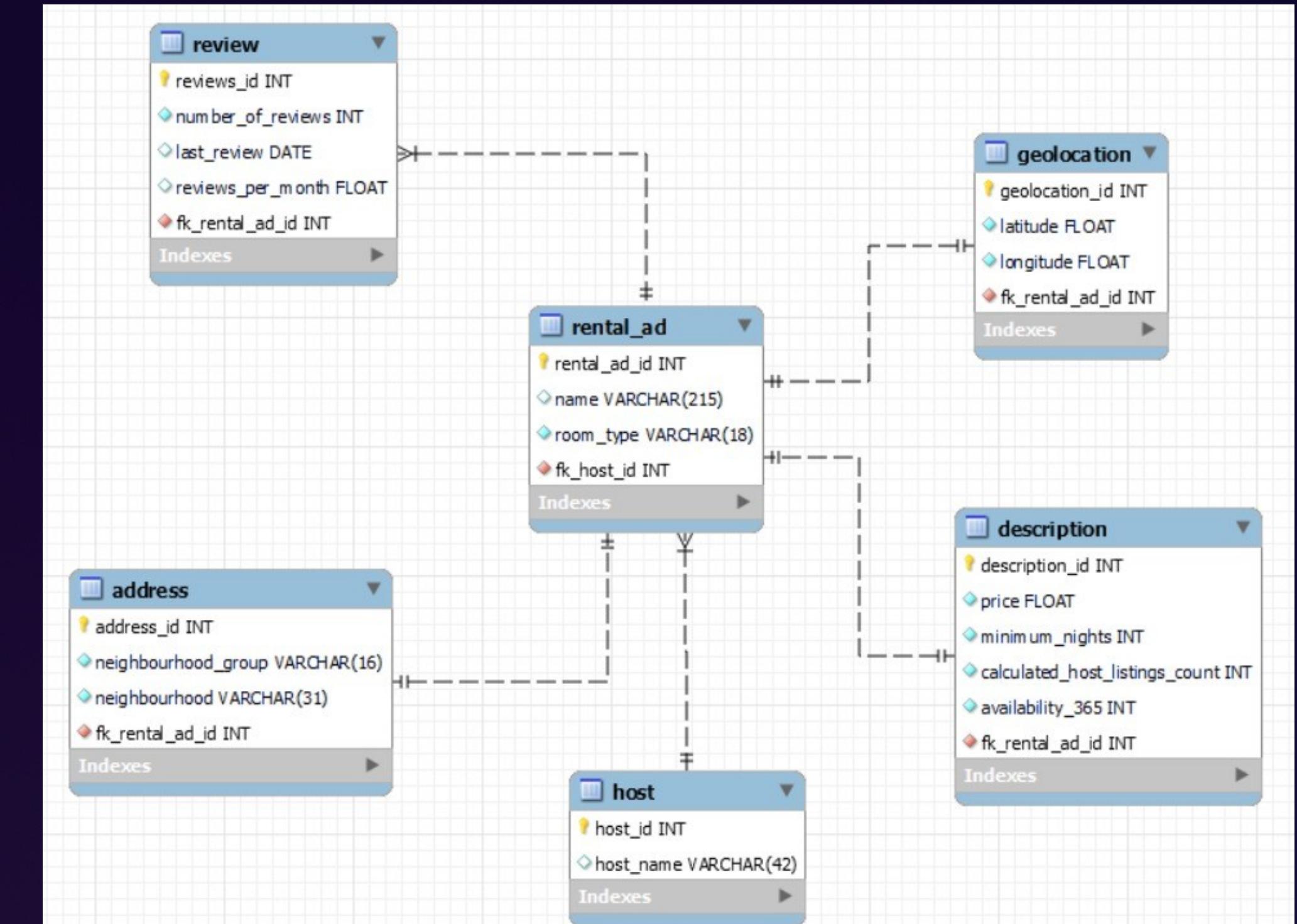
- Baseia-se em dependência transitiva: Identificar cada grupo de atributos não-chave que dependam de outros atributos não-chave;
- Criar uma tabela para armazenar os atributos (ou conjunto de atributos) não-chave que não estão relacionados à chave primária da tabela original.





# Modelo Lógico

Representação de dados mais detalhadas do que na modelagem conceitual, com suas entidades, relacionamentos e **atributos definidos** e como estes dados serão armazenados em um banco de dados.





# Modelo Físico

Representa um modelo definido para um banco de dados específico e na construção dos códigos na sua linguagem específica para criação das entidades, atributos, dependências e restrições.

```
#-----> TABELA "host" <-----#
• CREATE TABLE IF NOT EXISTS `datalogic`.`host` (
  `host_id` INT NOT NULL,
  `host_name` VARCHAR(42) NULL,
  PRIMARY KEY (`host_id`)
)
DEFAULT CHARACTER SET = utf8mb4;
```

Exemplo Modelagem Física no MySQL

Em resumo podemos dizer que:

	CONCEITUAL	LÓGICO	FÍSICO
NOME DE ENTIDADE	✓	✓	✗
RELACIONAMENTO DE ENTIDADES	✓	✓	✗
ATRIBUTOS	✓	✓	✗
CHAVE PRIMÁRIA (PK)	✗	✓	✓
CHAVE ESTRANGEIRA (FK)	✗	✓	✓
NOME DE TABELA	✗	✗	✓
NOME DE COLUNA	✗	✗	✓
TIPO DA COLUNA	✗	✗	✓

Resumo de Conceitual, Lógico e Físico



# Modelo Físico

**DDL's** (Data Definition Language): Responsáveis pela **definição da estrutura** e organização dos objetos no banco de dados, como tabelas, índices, visões e procedimentos armazenados. Exemplos: CREATE, ALTER e DROP.

```
#-----> TABELA "host" <-----#
• CREATE TABLE IF NOT EXISTS `datalogic`.`host` (
  `host_id` INT NOT NULL,
  `host_name` VARCHAR(42) NULL,
  PRIMARY KEY (`host_id`))
  DEFAULT CHARACTER SET = utf8mb4;

#-----> TABELA "rental_ad" <-----#
• CREATE TABLE IF NOT EXISTS `rental_ad` (
  `rental_ad_id` INT NOT NULL,
  `name` VARCHAR(215),
  `room_type` VARCHAR(18) NOT NULL,
  `fk_host_id` INT NOT NULL,
  PRIMARY KEY (`rental_ad_id`),
  CONSTRAINT `fk_host_rental_ad`
    FOREIGN KEY (`fk_host_id`)
    REFERENCES `host` (`host_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
  DEFAULT CHARACTER SET = utf8mb4;
```

```
#-----> TABELA "address" <-----#
• CREATE TABLE IF NOT EXISTS `datalogic`.`address` (
  `address_id` INT NOT NULL AUTO_INCREMENT,
  `neighbourhood_group` VARCHAR(16) NOT NULL,
  `neighbourhood` VARCHAR(31) NOT NULL,
  `fk_rental_ad_id` INT NOT NULL,
  PRIMARY KEY (`address_id`),
  CONSTRAINT `fk_rental_ad_address`
    FOREIGN KEY (`fk_rental_ad_id`)
    REFERENCES `datalogic`.`rental_ad` (`rental_ad_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
  DEFAULT CHARACTER SET = utf8mb4;
```



```
#-----> TABELA "review" <-----#
• CREATE TABLE IF NOT EXISTS `datalogic`.`review` (
  `reviews_id` INT NOT NULL AUTO_INCREMENT,
  `number_of_reviews` INT NOT NULL,
  `last_review` DATE NULL,
  `reviews_per_month` FLOAT NULL,
  `fk_rental_ad_id` INT NOT NULL,
  PRIMARY KEY (`reviews_id`),
  CONSTRAINT `fk_rental_review`
    FOREIGN KEY (`fk_rental_ad_id`)
    REFERENCES `datalogic`.`rental_ad` (`rental_ad_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
  DEFAULT CHARACTER SET = utf8mb4;

#-----> TABELA "description" <-----#
• CREATE TABLE IF NOT EXISTS `datalogic`.`description` (
  `description_id` INT NOT NULL AUTO_INCREMENT,
  `price` FLOAT NOT NULL,
  `minimum_nights` INT NOT NULL,
  `calculated_host_listings_count` INT NOT NULL,
  `availability_365` INT NOT NULL,
  `fk_rental_ad_id` INT NOT NULL,
  PRIMARY KEY (`description_id`),
  CONSTRAINT `fk_rental_ad_description`
    FOREIGN KEY (`fk_rental_ad_id`)
    REFERENCES `datalogic`.`rental_ad` (`rental_ad_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
  DEFAULT CHARACTER SET = utf8mb4;
```

```
#-----> TABELA "geolocation" <-----#
• CREATE TABLE IF NOT EXISTS `datalogic`.`geolocation` (
  `geolocation_id` INT NOT NULL AUTO_INCREMENT,
  `latitude` FLOAT NOT NULL,
  `longitude` FLOAT NOT NULL,
  `fk_rental_ad_id` INT NOT NULL,
  PRIMARY KEY (`geolocation_id`),
  CONSTRAINT `fk_rental_ad_geolocation`
    FOREIGN KEY (`fk_rental_ad_id`)
    REFERENCES `datalogic`.`rental_ad` (`rental_ad_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
  DEFAULT CHARACTER SET = utf8mb4;
```



**DML's (Data Manipulation Language):** Usadas para manipular os dados dentro das tabelas, realizando operações como inserção, atualização, exclusão e recuperação de dados. Exemplos: INSERT, UPDATE, DELETE e SELECT INTO.

```
#-----> TABELA "host" <-----#
#Relizando a inserção dos dados da tabela airbnb_stage para a tabela "host"
• INSERT INTO host (host_id, host_name)
  SELECT DISTINCT host_id, host_name
  FROM airbnb_stage;

#Output da inserção
• select * from host;

# conferindo quantos nomes se repetem
• SELECT host_name, COUNT(host_name) AS repeticoes
  FROM host
  GROUP BY host_name;

# Confirmando que os nomes repetidos sao pessoas com id únicos
• SELECT host_id, COUNT(host_id) AS repeticoes
  FROM host
  GROUP BY host_id
  HAVING COUNT(host_id) > 1;
```

```
#-----> TABELA "rental_ad" <-----#
#Relizando a inserção dos dados da tabela airbnb_stage para a tabela "rental_ad"
• INSERT INTO rental_ad (rental_ad_id, name, fk_host_id, room_type)
  SELECT id, name, host_id, room_type
  FROM airbnb_stage;

# Output da inserção
• select * from rental_ad;

#Confirmando número total de linhas inseridas
• select count(*) from rental_ad;

#Realizando teste de SELECT entre duas tabelas relacionadas.
• SELECT rental_ad_id, name, room_type, fk_host_id, host_id, host_name
  FROM rental_ad
  JOIN host ON fk_host_id = host_id;
```

```
#-----> TABELA "address" <-----#
#Realizando a inserção dos dados da tabela airbnb_stage para a tabela "address"
• INSERT INTO address (neighbourhood, neighbourhood_group, fk_rental_ad_id)
  SELECT neighbourhood, neighbourhood_group, id
  FROM airbnb_stage;

# Output da inserção
• select * from address;

#Confirmando número total de linhas inseridas
• select count(*) from address;

#Realizando teste de SELECT entre duas tabelas relacionadas.
• SELECT rental_ad_id, name, room_type, fk_host_id, host_id, host_name, neighbourhood, neighbourhood_group
  FROM rental_ad
  JOIN host ON fk_host_id = host_id
  JOIN address ON fk_rental_ad_id = rental_ad_id;
```

```
#-----> TABELA "description" <-----#
#Realizando a inserção dos dados da tabela airbnb_stage para a tabela "description"
• INSERT INTO description (price, minimum_nights, calculated_host_listings_count, availability_365, fk_rental_ad_id)
  SELECT price, minimum_nights, calculated_host_listings_count, availability_365, id
  FROM airbnb_stage;

# Output da inserção
• select * from description;

#Confirmando número total de linhas inseridas
• select count(*) from description;

#Realizando teste de SELECT entre duas tabelas relacionadas.
• SELECT rental_ad_id, name, room_type, fk_host_id, host_id, host_name, neighbourhood,
neighbourhood_group, price, minimum_nights, calculated_host_listings_count, availability_365
  FROM rental_ad
  JOIN host ON fk_host_id = host_id
  JOIN address ON fk_rental_ad_id = rental_ad_id
  JOIN description ON description.fk_rental_ad_id = rental_ad.rental_ad_id;
```

```

-----> TABELA "geolocation" <-----
#Realizando a inserção dos dados da tabela airbnb_stage para a tabela "geolocation"
• INSERT INTO geolocation (latitude, longitude, fk_rental_ad_id)
  SELECT latitude, longitude, id
  FROM airbnb_stage;

  # Output da inserção
• select * from geolocation;

#Confirmando número total de linhas inseridas
• select count(*) from geolocation;

#Realizando teste de SELECT entre duas tabelas relacionadas.
• SELECT rental_ad_id, name, room_type, fk_host_id, host_id, host_name, neighbourhood,
neighbourhood_group, price, minimum_nights, calculated_host_listings_count, availability_365,
longitude, latitude
FROM rental_ad
JOIN host ON fk_host_id = host_id
JOIN address ON fk_rental_ad_id = rental_ad_id
JOIN description ON description.fk_rental_ad_id = rental_ad.rental_ad_id
JOIN geolocation ON geolocation.fk_rental_ad_id = rental_ad.rental_ad_id;

```

```

-----> TABELA "review" <-----
#Realizando a inserção dos dados da tabela airbnb_stage para a tabela "review"
• INSERT INTO review ( number_of_reviews, last_review, reviews_per_month, fk_rental_ad_id)
  SELECT number_of_reviews, last_review, reviews_per_month, id
  FROM airbnb_stage;

  # Output da inserção
• select * from review;

#Confirmando número total de linhas inseridas
• select count(*) from review;

#Realizando teste de SELECT entre duas tabelas relacionadas.
• SELECT rental_ad_id, name, room_type, host_id, host_name, neighbourhood, neighbourhood_group, price,
minimum_nights, calculated_host_listings_count, availability_365, latitude, longitude,
number_of_reviews, last_review, reviews_per_month
FROM rental_ad
JOIN host ON fk_host_id = host_id
JOIN address ON fk_rental_ad_id = rental_ad_id
JOIN description ON description.fk_rental_ad_id = rental_ad.rental_ad_id
JOIN geolocation ON geolocation.fk_rental_ad_id = rental_ad.rental_ad_id
JOIN review ON review.fk_rental_ad_id = rental_ad.rental_ad_id;

```



**DQL's (Data Query Language):** Responsáveis por recuperar dados específicos do banco de dados. O comando principal da DQL é o **SELECT**, que **permite realizar consultas para recuperar informações** de uma ou mais tabelas.

- `select * from airbnb_stage order by id ASC;`
- `select max(id) from airbnb_stage;`
- `SELECT id, COUNT(id) AS repeticoes  
FROM airbnb_stage  
GROUP BY id  
HAVING COUNT(id) <> 1;`
- `SELECT host_id, COUNT(host_id) AS repeticoes  
FROM airbnb_stage  
GROUP BY host_id  
HAVING COUNT(host_id) <> 1;`
- `select * from airbnb_stage where host_id="2787";`

- `SELECT * FROM host;`
- `SELECT host_id, COUNT(host_id) AS repeticoes  
FROM airbnb_stage  
GROUP BY host_id;`
- `SELECT host_name, COUNT(host_name) AS repeticoes  
FROM airbnb_stage  
GROUP BY host_name;`
- `select * from airbnb_stage where host_name="John";`



# Comprimento Máximo de dados e Validação de Nulos:

```
# N U L O S

#CONTAGEM DE NULO POR ATRIBUTO: (name, host_name, last_review, reviews_per_month - Possuem nulos!)

• SELECT COUNT(*) FROM airbnb_stage WHERE id IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE name IS NULL;                                #tem 16 nulos!
• SELECT COUNT(*) FROM airbnb_stage WHERE host_id IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE host_name IS NULL;                            #tem 21 nulos!
• SELECT COUNT(*) FROM airbnb_stage WHERE neighbourhood IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE neighbourhood_group IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE latitude IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE longitude IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE room_type IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE price IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE minimum_nights IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE number_of_reviews IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE last_review IS NULL;                         #TEM 10052 NULOS!!
• SELECT COUNT(*) FROM airbnb_stage WHERE reviews_per_month IS NULL;                  #TEM 10052 NULOS!!
• SELECT COUNT(*) FROM airbnb_stage WHERE calculated_host_listings_count IS NULL;
• SELECT COUNT(*) FROM airbnb_stage WHERE availability_365 IS NULL;

#Puxar as linhas COMPLETAS com todas as colunas onde o atributo X foi apresentado como nulo:

• SELECT * FROM airbnb_stage WHERE name IS NULL;                                     #TEM 16 NULOS!
• SELECT * FROM airbnb_stage WHERE host_name IS NULL;                               #TEM 21 NULOS!
• SELECT * FROM airbnb_stage WHERE last_review IS NULL;                            #TEM 10052 NULOS!!
• SELECT * FROM airbnb_stage WHERE reviews_per_month IS NULL;                      #TEM 10052 NULOS!!
```

# Início da Montagem do Dicionário de Dados:

```
# (Airbnb.csv)Keagle --> (*Stage* --> Mod. Log. Relacional )MYSQL --> (Collection_airbnb)Servidor_MONGO_DB  
# Esta Ultima,esta no Git, além disso, procurar por plugins python para isso  
# ESTE É O PASSO "STAGE", QUE PRECEDE O PASSO "MOD. LOG. RELACIONAL"  
  
• show tables;  
  
#Para Montagem do "DICONARIO DE DADOS" com Nome Coluna, Tipo Dado, Nullable, Descrição do Campo(Este tem no Keaggle)  
• describe airbnb_stage;  
  
#Imprimindo a tabela  
• select * from airbnb_stage;  
  
#total de dados em nosso banco:  
• select count(*)from airbnb_stage;  
• select * from airbnb_stage where host_id="22486";
```



# Dicionário de Dados:

Origem dos dados				
O conjunto de dados descreve as atividades de listagem e métricas no Airbnb, fornecendo informações sobre os anfitriões, disponibilidade geográfica e avaliações dos usuários. Tais dados foram obtidos através de um dataset c				
Dicionário de Dados				
1. Tabela: host (anfitrião)				
Campo/Atributo	Tipo	Definições	Significado	Uso
host_id	INT	Inteiro	Identificador único para cada anfitrião (chave primária).	Esta tabela armazena informações sobre o Anfitrião. O campo fornece detalhe como o nome.
host_name	VARCHAR(42)	Cadeia de caracteres de 42 caracteres	Nome do anfitrião.	
2. Tabela: rental_ad (anúncio do aluguel)				
Campo/Atributo	Tipo	Definições	Significado	Uso
rental_ad_id	INT	Inteiro	Identificador único para cada anúncio de aluguel (chave primária).	Esta tabela armazena informações sobre os anúncios de alugueis. Os campos fornecem detalhes como nome e tipos de acomodações.
name	VARCHAR(215)	Cadeia de caracteres de 215 caracteres	Nome do aluguel.	
room_type	VARCHAR(18)	Cadeia de caracteres de 18 caracteres	Tipos de acomodações.	
fk_host_id	INT	Inteiro	Chave estrangeira referenciando o "host_id" na tabela "host".	
3. Tabela: address (endereço)				
Campo/Atributo	Tipo	Definições	Significado	Uso
address_id	INT	Inteiro	Identificador único para cada endereço (chave primária).	Esta tabela armazena informações sobre os endereços. Os campos fornecem detalhes como bairro e ponto de referência.
neighbourhood_group	VARCHAR(16)	Cadeia de caracteres de 16 caracteres	Bairro.	
neighbourhood	VARCHAR(31)	Cadeia de caracteres de 31 caracteres	Ponto de referência.	
fk_rental_ad_id	INT	Inteiro	Chave estrangeira referenciando o "rental_ad_id" na tabela "rental_ad".	
4. Tabela: reviews (avaliações)				
Campo/Atributo	Tipo	Definições	Significado	Uso
reviews_id	INT	Inteiro	Identificador único para cada avaliação (chave primária).	Esta tabela armazena informações sobre as avaliações. Os campos fornecem detalhes como o número de avaliações, as avaliações anteriores e as avaliações por mês.
number_of_reviews	INT	Inteiro	Número das avaliações.	
last_review	DATE	Data	Número avaliações anteriores.	
reviews_per_month	FLOAT	Decimal	Número de avaliações por mês.	
fk_rental_ad_id	INT	Inteiro	Chave estrangeira referenciando "rental_ad_id" na tabela "rental_ad".	

# Dicionário de Dados:

Tabela 05	5. Tabela: description (descrição)					
	Campo/Atributo	Tipo	Definições	Significado	Uso	
Tabela 05	description_id	INT	Inteiro	Identificador único para cada descrição (chave primária).	Essa tabela armazena a descrição com informações sobre o anúncio de aluguel. Os campos fornecem detalhes como o preço, o mínimo de noites, quantidade de hóspedes e quantidade de dias disponíveis.	
Tabela 05	price	FLOAT	Decimal	Armazena o preço do anúncio de aluguel.		
Tabela 05	minimum_nights	INT	Inteiro	Mínimo de noites.		
Tabela 05	calculated_host_listings_count	INT	Inteiro	Quantidade de hóspedes suportada.		
Tabela 05	availability_365	INT	Inteiro	Quantidade de dias disponíveis.		
Tabela 05	fk_rental_ad_id	INT	Inteiro	Chave estrangeira referenciando o 'rental_ad_id' na tabela 'rental_ad'.		
Tabela 06	6. Tabela: geolocation (geolocalização)					
	Campo/Atributo	Tipo	Definições	Significado	Uso	
Tabela 06	geolocation_id	INT	Inteiro	Identificador único para cada geolocalização (chave primária).	Esta tabela armazena informações sobre geolocalização do imóvel a ser alugado no anúncio. Os campos fornecem detalhes como a latitude e longitude da localização.	
Tabela 06	latitude	FLOAT	Decimal	Armazena a latitude da localização.		
Tabela 06	longitude	FLOAT	Decimal	Armazena a longitude da localização.		
Tabela 06	fk_rental_ad_id	INT	Inteiro	Chave estrangeira referenciando " rental_ad_id " na tabela " rental_ad "		
Relacionamento das tabelas						
	Tabela	Tabela FK	Tabela PK	Relacionamento	Nome do relacionamento	Descrição
Tabela 02	rental_ad	fk_host_id	host_id	host	Possui	Relacionamento que descreve para qual host (anfitrião) possui aquele rental_ad (anúncio de aluguel).
Tabela 03	address	fk_rental_ad_id	rental_ad_id	rental_ad	Pertence	Relacionamento que descreve para qual address (endereço) a
Tabela 04	reviews	fk_rental_ad_id	rental_ad_id	rental_ad	Possui	Relacionamento que descreve as reviews (avaliações) que o
Tabela 05	description	fk_rental_ad_id	rental_ad_id	rental_ad	Contém	Relacionamento que contém a description (descrição) do
Tabela 06	geolocation	fk_rental_ad_id	rental_ad_id	rental_ad	Contém	Relacionamento que contém a geolocation (geolocalização) do

Relacionamento das  
Tabelas

# Dicionário de Dados:

Regras de Negócio
<b>Restrições de integridade referencial</b>
Garantir que as chaves estrangeiras (FK) nas tabelas estejam sempre referenciando chaves primárias válidas nas tabelas relacionadas.
Exemplo: Garantir que cada entrada na tabela address (endereço) esteja associada a um anúncio de aluguel existente na tabela rental_ad.
<b>Consistência de Dados</b>
Assegurar que os dados inseridos nas tabelas, como por exemplo, host_name, name, room_type, neighbourhood_group, neighbourhood estejam corretos e consistentes.
Exemplo: Garantir que o nome de um anfitrião seja no formato aceitável do tipo de dado VARCHAR (por exemplo, "John", "Elisabeth" etc.)
<b>Restrições de Validação de Dados</b>
Validar os dados inseridos nas tabelas para garantir que atendam aos critérios definidos para cada campo.
Exemplo: Garantir que o nome de um anfitrião seja no formato aceitável do tipo de dado VARCHAR (por exemplo, "John", "Elisabeth" etc.)
<b>Gerenciamento de Relacionamentos</b>
Garantir que os relacionamentos entre os diferentes elementos (host, rental_ad, address, review, description, geolocation) sejam corretamente mantidos e atualizados quando necessário.
Exemplo: Garantir que quando um anúncio de aluguéis for removido do rental_ad , suas entradas associadas na tabela address, review, description e geolocation, como FKs sejam também removidas ou atualizadas adequadamente.
<b>Privacidade e Segurança dos Dados</b>
Garantir que os dados dos usuários e os dados confidenciais da Airbnb sejam devidamente protegidos e que apenas pessoas autorizadas tenham acesso a eles em conformidade com a Lei Geral de Proteção de Dados Pessoais (LGPD),

## Regras de Negócio

# Diferenças Fundamentais na Gestão de Dados

É importante ressaltar a diferença fundamental entre SQL (Structured Query Language) e NoSQL (Not Only SQL). Enquanto **SQL** é um modelo de banco de dados **relacional** que utiliza um esquema fixo e tabelas com linhas e **colunas predefinidas**, o **NoSQL** permite o armazenamento de dados **semi-estruturados** ou **não estruturados**, facilitando a manipulação de conjuntos de dados heterogêneos, como é o caso do formato documento que estamos utilizando neste projeto.



# NoSQL: Documentos e o Uso de CSV como Formato de Dados Flexível

A **tecnologia NoSQL** baseada em documento, é uma categoria ampla que inclui várias implementações, o **MongoDB** é uma implementação específica dessa tecnologia, e outros bancos de dados NoSQL, como Couchbase, CouchDB e Amazon DynamoDB, também podem ser classificados como tecnologias NoSQL baseadas em documento.

O formato escolhido **CSV** (Comma-Separated Values), em tradicionalmente associado a bancos de dados relacionais, pode ser considerado uma forma de NoSQL devido à sua estrutura flexível e sem esquema.

Document 1	Document 2	Document 3
{ "id": "1", "name": "John Smith", <del>"isActive": true,</del> "dob": "1964-30-08" }	{ "id": "2", <del>"fullName": "Sarah Jones",</del> <del>"isActive": false,</del> "dob": "2002-02-18" }	{ "id": "3", <del>"fullName":</del> { "first": "Adam", "last": "Stark" }, <del>"isActive": true,</del> "dob": "2015-04-19" }





# MongoDB



“O MongoDB é um banco de dados NoSQL, orientado a documentos, que oferece **flexibilidade** e **escalabilidade**. Ele armazena dados em documentos JSON, permitindo esquemas dinâmicos e consultas poderosas. O MongoDB é **altamente escalável** e distribuído, adequado para aplicativos que precisam lidar com grandes volumes de dados não estruturados e que exigem agilidade no desenvolvimento.”

- Filipe Nascimento

```
from pymongo import MongoClient
import mysql.connector

#Função de conexão com MongoDB. Criação do client, da DataBase e da Collection.
def conectar_mongodb():
    connection_string = "mongodb://root:rootpw@localhost:27017/?authSource=admin"
    client = MongoClient(connection_string)
    db_connection = client["DB_AirBnb"]
    collection = db_connection.get_collection("Collection_AirBnb")
    return collection

def importar_desnormalizar_mysql():
    # Valores com as configurações necessárias de conexão com o MySQL
    config_mysql = {
        'user': 'root',
        'password': 'root',
        'host': 'localhost',
        'port': 3306,
        'database': 'datalogic'
    }

    # Estabelecer conexão
    connection_mysql = mysql.connector.connect(**config_mysql)
    cursor_mysql = connection_mysql.cursor()

    # Query para selecionar os dados já normalizados do modelo relacional
    query = """
        SELECT rental_ad_id, name, room_type, host_id, host_name, neighbourhood,
        neighbourhood_group, price, minimum_nights, calculated_host_listings_count,
        availability_365, latitude, longitude, number_of_reviews, last_review, reviews_per_month
        FROM rental_ad
        JOIN host ON fk_host_id = host_id
        JOIN address ON fk_rental_ad_id = rental_ad_id
        JOIN description ON description.fk_rental_ad_id = rental_ad.rental_ad_id
        JOIN geolocation ON geolocation.fk_rental_ad_id = rental_ad.rental_ad_id
        JOIN review ON review.fk_rental_ad_id = rental_ad.rental_ad_id;
    """
    ....
```

# Importação e Desnormalização de Dados para o MongoDB Com Python



```
# Executar a query no MySQL
cursor_mysql.execute(query)

# Coleção para armazenar os dados de variedades do airbnb
# Conexão feita antes do looping
collection_criada_airbnb = conectar_mongodb()

# Loop através das linhas retornadas pelo cursor da query do MySQL
for row in cursor_mysql.fetchall():

    # Convertendo o valor da coluna "last_review" de DATE para string formatada
    def date_string():
        if row[14] is not None:
            return row[14].strftime('%Y-%m-%d')
        else:
            return row[14]

    # Criando documento para inserção no MongoDB.
    # Em cada repetição, os é assumido os valores da nova linha
    documento = {
        "rental_ad_id":row[0],
        "name":row[1],
        "room_type":row[2],
        "host_id":row[3],
        "host_name":row[4],
        "neighbourhood":row[5],
        "neighbourhood_group":row[6],
        "price":row[7],
        "minimum_nights":row[8],
        "calculated_host_listings_count":row[9],
        "availability_365":row[10],
        "latitude":row[11],
        "longitude":row[12],
        "number_of_reviews":row[13],
        "last_review":date_string(),
        "reviews_per_month":row[15]
    }
```

# Automatizando a Migração de Dados: MySQL para MongoDB



```
# Já conectado ao banco, em cada repetição será inserindo o documento na coleção
| collection_criada_airbnb.insert_one(documento)

# Fechar conexão com o MySQL
| cursor_mysql.close()
| connection_mysql.close()

# Conectar ao MongoDB
#db = conectar_mongodb()

importar_desnormalizar_mysql()

"""
print(db_connection)
print()
collection = db_connection.get_collection("minhaCollection")

print(collection)
print()

search_filter = { "estou": "aqui" }
response = collection.find(search_filter)

for registry in response: print(registry)

collection.insert_one({
    "Estou": "Inserindo",
    "Numeros": [123, 456, 789]
})
"""


```

# Finalização da Migração de Dados para o MongoDB



MongoDB Compass - Conexao banco docker/DB\_AirBnb.Collection\_AirBnb

Connect Edit View Collection Help

Conexao banco ...

My Queries DB\_AirBnb Collection\_AirBnb

Conexao banco docker > DB\_AirBnb > Collection\_AirBnb

Documents 48.9K Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

ADD DATA EXPORT DATA UPDATE DELETE

`_id: ObjectId('66401e2444d48c6e4d60fe45')  
rental_ad_id: 2539  
name: "Clean & quiet apt home by the park"  
room_type: "Private room"  
host_id: 2787  
host_name: "John"  
neighbourhood: "Kensington"  
neighbourhood_group: "Brooklyn"  
price: 149  
minimum_nights: 1  
calculated_host_listings_count: 6  
availability_365: 365  
latitude: 40.6475  
longitude: -73.9724  
number_of_reviews: 9  
last_review: "2018-10-19"  
reviews_per_month: 0.21`

`_id: ObjectId('66401e2444d48c6e4d60fe46')  
rental_ad_id: 2595  
name: "Skylit Midtown Castle"  
room_type: "Entire home/apt"  
host_id: 2845  
host_name: "Jennifer"  
neighbourhood: "Midtown"  
neighbourhood_group: "Manhattan"  
price: 225`

My Queries Performance Databases Search DB\_AirBnb Collection\_AirBnb admin config local meuBanco

# MongoDB: Banco de Dados Integrado com os Dados do MySQL

- ✓ Flexibilidade de Esquema
- ✓ Escalabilidade Horizontal
- ✓ Consultas Avançadas
- ✓ Alta Disponibilidade
- ✓ Tolerância a Falhas





DataLogic

# Thank You!

Get in touch with  
us



[github.com/TulioMendesDev/DataLogic](https://github.com/TulioMendesDev/DataLogic)

