Relatório

Carlos Martins, 18836

João Azevedo, 18845

Barcelos, Fevereiro 2023

Contents

# Introduction

Uma empresa que comercializa bicicletas numa cadeia de lojas de mobilidade sustentável, está a mudar o seu sistema de dados de base de dados relacionais on-premise, para Big Data. Além dos dados novos pretende integrar no seu novo sistema os dados históricos de vendas, que foram exportados dos sistemas atuais em formato CSV. O armazenamento de dados será feito em HDFS, e será utilizado o Hive para gestão de dados.

## Project objectives

Dada a introdução feita no capítulo anterior (1) pretendemos que a elaboração deste projeto tenha em conta os seguintes objetivos:

* Deve ser realizada a melhor modelação dos dados de acordo com o ficheiro de dados disponibilizado.
* Garantir o melhor formato para armazenamento dos dados em termos de performance e armazenamento.
* Garantir a melhor performance das queries realizadas em Hive.
* Implementar algumas análises de dados em Zepplin.

# Modelação de Dados

Nesta secção será apresentado o modelo de dados escolhido após a análise dos dados.

O primeiro passo passou pela análise do CSV disponibilizado para a realização do trabalho. Este CSV continha um conjunto de dados que estava agrupado e, inicialmente, foi realizada a separação do CSV original em múltiplos CSVs, onde cada um correspondia a uma tabela como mostra a figura 1.

Com base na figura 1, podemos verificar que o “produto” final consiste em quatro tabelas, sendo elas as seguintes:

* **SalesOrder –** Esta tabela é a tabela que armazena todas as “faturas” do cliente, ou seja, cada linha desta tabela representa uma encomenda realizada por um certo cliente. O seu conteúdo consiste nas informações temporais, financeiras e de cliente para a respetiva fatura.
* **SalesOrderDetail –** Esta tabela armazena todas as linhas de uma “fatura”, ou seja, o seu conteúdo é, basicamente, uma linha da fatura que contém que produto foi comprado, a respetiva quantidade e ao valor total da linha.
* **Product –** Esta tabela armazena informações sobre os produtos, tais como, o nome, o preço e peso.
* **Customer –** Esta tabela armazena as informações sobre clientes, tais como, o nome, nome da empresa e contactos.

Figura - Modelo de dados

De modo a garantir o melhor desempenho do sistema, foram escolhidos certos tipos de dados que após investigação e conhecimento obtido através de aula, funcionam melhor.

Para guardar texto, foi escolhido o tipo de dados *string* porque em Hive, ao contrário do tipo de dados *varchar*, as *strings* utilizam um processo chamado vectorização\* quando são utilizados ficheiros ORC (que é o nosso caso).

Para guardar números, foram utilizados diferentes tipos de dados:

* Quanto a números inteiros dentro do intervalo -128 a 127 foi utilizado o tipo de dados *tinyint*. Este tipo de dados foi utilizado para as colunas “RevisionNumberId” e “OnlineOrderFlag” da tabela SalesOrder, por exemplo.
* Relativamente a números inteiros dentro do intervalo -32768 a 32767 foi utilizado o tipo de dados *smallint*. Este tipo de dados foi utilizado para as colunas “ShipToAddressId” e “BillToAddressId” da tabela SalesOrder, por exemplo.
* Por fim, para armazenar números flutuantes com precisão até 7 casas decimais, foi utilizado o tipo de dados *float*. Este tipo de dados foi utilizado nas colunas “UnitPrice” e “UnitPriceDiscount” da tabela Product, por exemplo.

Para armazenar datas com horas, foi utilizado o tipo de dados *timestamp* uma vez que é o único definido para esse efeito.

\*A vectorização é um processo que permite ao Hive processar linhas em *batch* em vez de uma de cada vez, o que aumenta a velocidade de processamento.

# Armazenamento de dados em Hive

O Hive suporta vários formatos de arquivos, cada um com suas próprias características, vantagens e usos adequados.

Um desses formatos é o ***Text File Format***. Este formato é, talvez, o mais simples que o Hive suporta. Neste formato, os ficheiros de texto são armazenados como ficheiros de texto simples no HDFS. O maior proveito deste formato pode ser tirado em projetos de pequena dimensão ou em dados não estruturados.

Além disso, existe também o ***SequenceFile Format***que é utilizado para armazenar conjuntos *key-value*. Este formato é mais adequado para ficheiros com uma grande quantidade de dados, devido à sua eficiência de armazenamento e performance na leitura dos mesmos.

O **RC *File*** (*Record Columnar File*) é semelhante ao Sequence File Format, mas, ao contrário do *SequenceFile Format*, este trabalha ao nível da coluna e não ao nível da linha. Este formato é mais adequado para dados estruturados com muitas colunas.

Existe ainda, o **ORC *File*** (*Optimized Row Columnar*) que é um tipo de ficheiro otimizado para de colunas e tem muitas outras melhorias de desempenho, como compressão de dados e suporte para dados nulos. Este formato é mais adequado para projetos de grande escala com dados estruturados. Por fim, temos o **Parquet File** é outro formato de arquivo avançado suportado por Hive. É semelhante ao ORC, mas é projetado para ser utilizado com vários sistemas de base de dados, incluindo Hive, Impala, Spark, etc. Este formato é mais adequado para projetos que exigem colaboração entre sistemas de base de dados.

Após a análise dos formatos indicados acima foi escolhido como tipo de formato final o ORC por garantir um bom desempenho do sistema.

# Implementação do Modelo

# Upload dos Dados

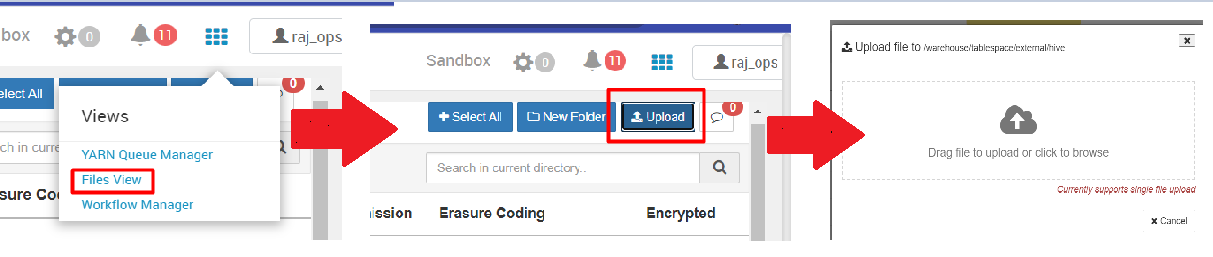
Como foi referido no [capítulo 2](#_Modelação_de_Dados), o ficheiro original foi dividido em quatro onde cada um correspondia a uma tabela.

Relativamente a esta divisão dos ficheiros, tivemos de fazer um conjunto de alterações aos dados iniciais, das quais:

* A remoção de todas as vírgulas presentes nos ficheiros. Isto deveu-se ao delimitador escolhido para o carregamento dos dados para as tabelas ser a vírgula. Tentámos descartar esta necessidade ao máximo, escolhendo outros delimitadores como, por exemplo, o cardinal, mas os dados não eram inseridos com sucesso.
* A alteração do delimitador das colunas dos ficheiros de ponto e vírgula para vírgula.
* A remoção de *primary keys* duplicadas das tabelas SalesOrderDetails, Product e Customer. Como referido anteriormente, no modelo de base de dados implementado foi definido que existe uma ligação fictícia de um para muitos entre as tabelas SalesOrder e SalesOrderDetails.

Por fim, para efetuar o carregamento para o Hive dos ficheiros obtidos, foram realizados os seguintes passos:

1. Foi criada uma base de dados, sendo a sua localização na diretoria:   
   ‘/warehouse/tablespace/external/hive/bda\_g01\_files'. Esta base de dados teve como objetivo servir como área de preparação de dados e foi criada através do respetivo comando: 
2. De seguida, foram carregados os ficheiros resultantes para a diretoria indicada no ponto anterior. Isto foi realizado através da UI do Ambari.



1. Foram criadas as tabelas com um conjunto de condições. Para título de exemplo, será apenas apresentado o código da criação de uma e as restantes vão estar presentes nos anexos no fim do documento.



1. Após a execução do comando anterior, foi executado um comando cujo objetivo é carregar os dados do ficheiro para dentro da tabela. O comando foi o seguinte:



1. Depois dos dados estarem carregados nas tabelas temporárias, foram criadas as tabelas finais. Estas tabelas finais foram criadas com o parâmetro “STORED AS ORC” de modo a definir que a tabela deve ser guardada em formato ORC. O comando utilizado foi o seguinte:  
   
2. Por fim, foram carregados os dados presentes na tabela temporária para a tabela final com o comando: 

# Implementação e análise das querys

# Implementação e análise das querys em Zeppelin