

Talend Open Studio e Python

Data Mart
DMOIs

SUMÁRIO

1.0. Ferramenta Talend	3
2.0. DMOs	4
2.1. DMOs Python e Talend Estrutura	6
2.2. DMOs Stagings	9
2.3. DMOs Schemas	14
2.4. DMOs Execução do ETL	20
2.5. DMOs Docs	25

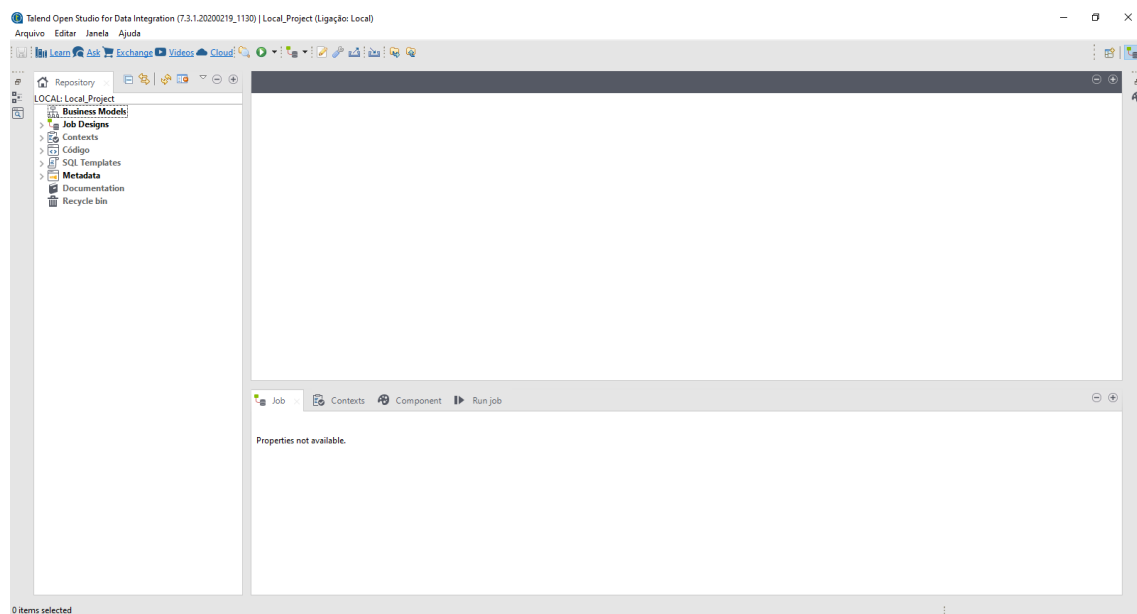
1.0. FERRAMENTA TALEND

Talend, mais precisamente, Talend Cloud é uma plataforma (SaaS), mas o Talend Open Studio, é gratuita, desenvolvida em Java que fornece ferramentas para integração e gerenciamento de dados, a chave principal é para o processo ETL, com esta solução é possível conectar dados de várias fontes em diferentes sistemas. Ele suporta integração de dados em lote e em tempo real.

Esta ferramenta é muito fácil de implementar e começar a desenvolver, Talend fornece um ambiente de desenvolvimento gráfico que permite aos usuários projetar e implementar integração de dados e/ou ETL.

A ferramenta é dividida em “componentes” e estes componentes possuem inúmeras propriedades, como componentes de conexão via jdbc e clouds, transformações de dados, mapeamentos, notas, etc,

O Talend possui componentes de conexão com as clouds, é possível usar esta ferramenta para integração com as três principais provedoras, isso permite uma ampla gama de outras atividades que a ferramenta possibilita a ser desenvolvida, como migração de dados para a nuvem.



Interface do Talend no Windows 10.

2.0. DMOls

A DMOls é um simples projeto de modelagem de dados de conceitos Star Schema de um Data Mart, a ideia é apenas explorar a ferramenta e suas funcionalidades com a sua analogia desenvolvida em puramente em Python.

Os dados utilizados foram do marketplace olist, disponível neste link:

<https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce>

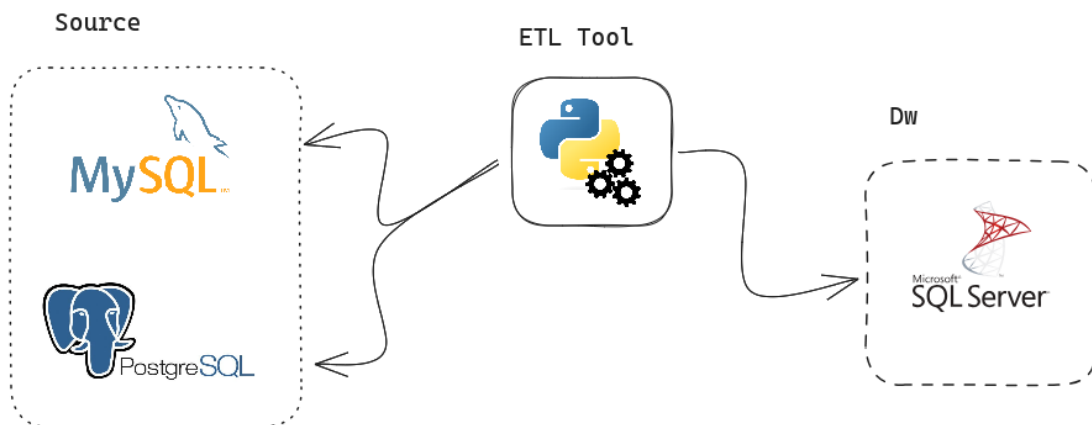
Após isso, estes csv do Kaggle da plataforma “Olist” foram migrados a dois bancos de dados diferentes, um banco MySQL e um banco PostgreSQL para simular dois bancos OLTP, ou duas fontes diferentes, os respectivos foram levantados com a ajuda do docker. E para simular a “Dw” ou em outras palavras o destino das integrações e do processo de ETL foi utilizado o banco de dados SQL Server que foi levantado via docker também.

Foi utilizado dois bancos porque nem sempre existem apenas uma fonte de dados no dia a dia, assim os dados dos bancos se complementam para as análises.

Repositório do projeto no github:

<https://github.com/xGabrielR/Traditional-Dw-Etl/tree/main>

A ideia geral é fazer um simples pipeline que irá coletar dados do PostgreSQL e do MySQL e salvar as informações na “Dw” no banco SQL Server como esta no simples diagrama abaixo.

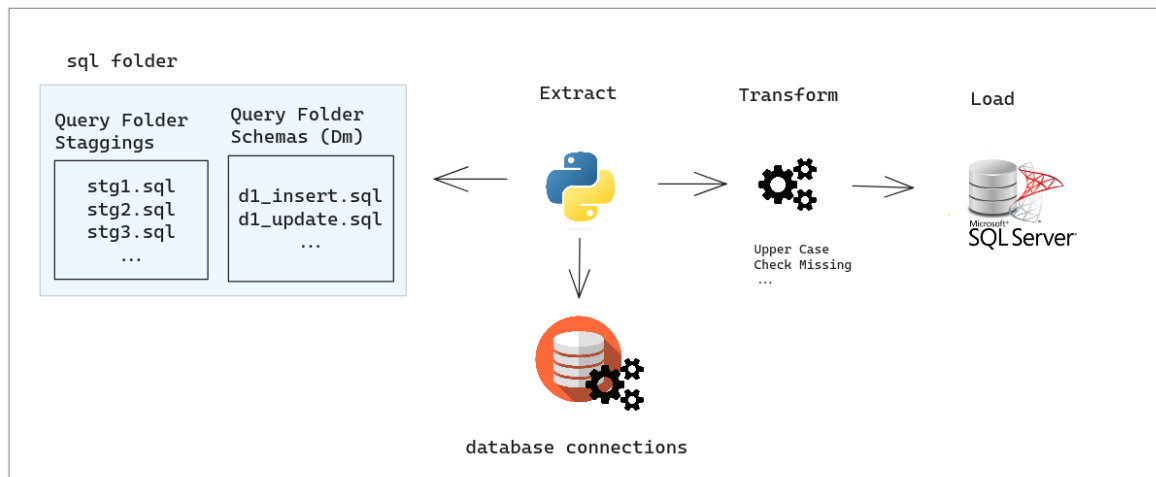


Dando um zoom-in, a arquitetura em pastas do projeto em Python, foi ligeiramente diferente da solução em Talend do mesmo projeto.

No Talend as consultas são armazenadas em seus respectivos “componentes”, na versão do Python, queria de alguma forma que essas consultas também pudessem ser acessadas e modificadas facilmente.

Mas, em regras gerais, a arquitetura que montem para o projeto em python foi a representada na imagem abaixo:

Geral Pipeline



Existem as respectivas pastas e dentro delas as consultas a serem executadas (extract), quem irá fazer isso vai ser o Python, após isso as conexões com os bancos, pois é necessário uma conexão com o banco para executar as consultas.

Após isso, esses dados vão ser transformados de diversas formas, podem ser transformados na consulta SQL ou no próprio Python / Talend.

E por fim, esses dados são armazenados nos respectivos schemas a fim de atender algum problema ou montar uma visualização.

Para acompanhar toda a implementação, considere acompanhar os scripts em python e em Talend.

2.1. DMOs TALEND E PYTHON ESTRUTURA

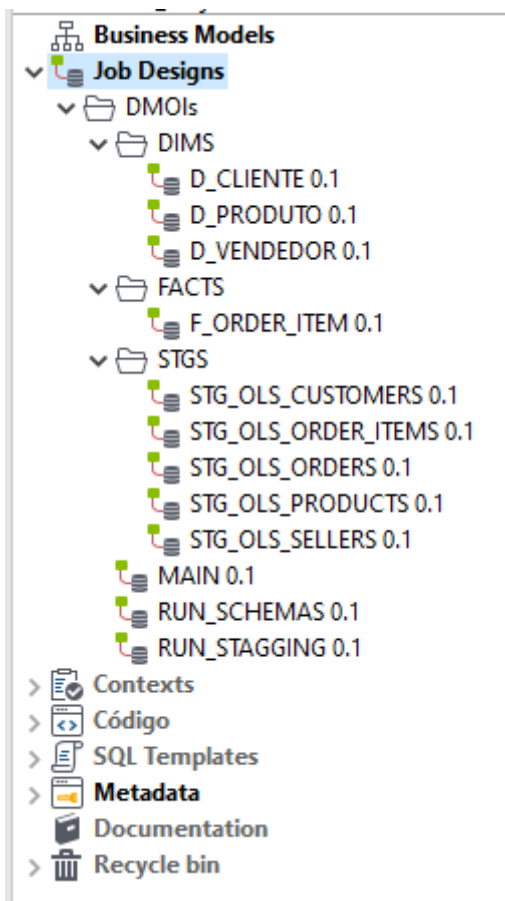
Para a versão do python foi desenvolvido um ambiente com poetry e as respectivas dependências que estão na descrição do repositório, a partir disso foi planejado um simples sistema onde existem consultas em arquivos SQL que precisam ser lidos e executadas no processo, similar ao Talend, mas no Talend escreve-se as consultas dentro da própria ferramenta em seus componentes de execuções.

Para a primeira etapa da versão do Python quanto da versão do Talend, foi definir a arquitetura inicial de como iria se comportar as pastas e os arquivos do projeto, para o Talend e o Python utilizei a mesma lógica, existe um script python chamado main.py e um job em Talend chamado MAIN, que é responsável por executar os respectivos subprocessos: run_staggings.py em Python e (RUN_STAGGING job em Talend) e também run_schemas.py em Python e (RUN_SCHEMAS job em Talend), esta é a arquitetura inicial dos processos.

Cada uma das duas funções / métodos ou Jobs representam a carga das stages e das dimensões e fatos para montar o Data Mart dentro da DW.

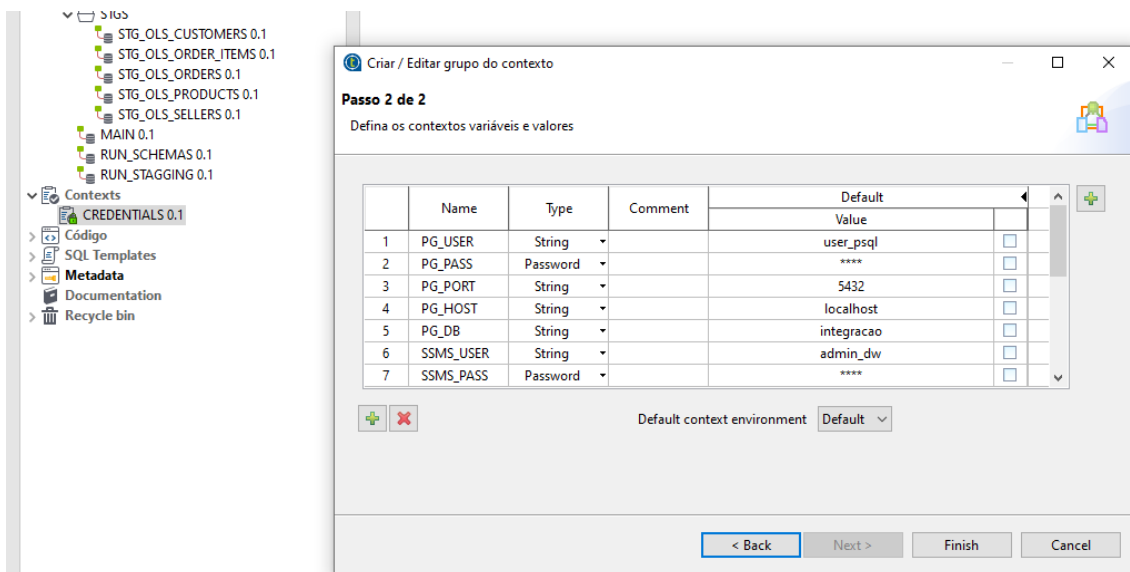
As Stages nada mais são que o dado da fonte já carregado na DW em um formato tabular (pois podem ser outras fontes como API) já adequados para serem modelados nas dimensões e fatos.

As Dimensões são as características modeladas em algum Fato, por exemplo fato “venda”, a venda que ocorreu possui uma determinada característica que pode ser o cliente, e este cliente tem seus atributos como nome, cpf, portanto F_Venda e D_Cliente comportar uma modelagem.



Para o Talend, como a consulta em sql a ser executada pode ser armazenada dentro dos componentes, então cada pasta possui o nome do processo em seus respectivos Jobs que serão executados nos (RUN_STAGGING e RUN_SCHEMAS) e cada Job deste possui uma consulta a ser realizada nos bancos de dados MySQL, PostgreSQL e SQL Server, esta é a estrutura de Jobs do Talend que escolhi.

Para a versão do Python é quase a mesma coisa, exceto que existe mais processos que irão coletar as query em formato “.sql” de alguma determinada pasta, e todo esse processamento está abstraído e é realizado dentro das classes (RunStagging e RunSchemas respectivamente), basta instanciar a classe e rodar os métodos, essa analogia final é a mesma do Talend, basta chamar um Job então é executado tudo que tem dentro dele.



Já para as credenciais no Python fica dentro da pasta utils e no Talend no bloco “Contexts”.

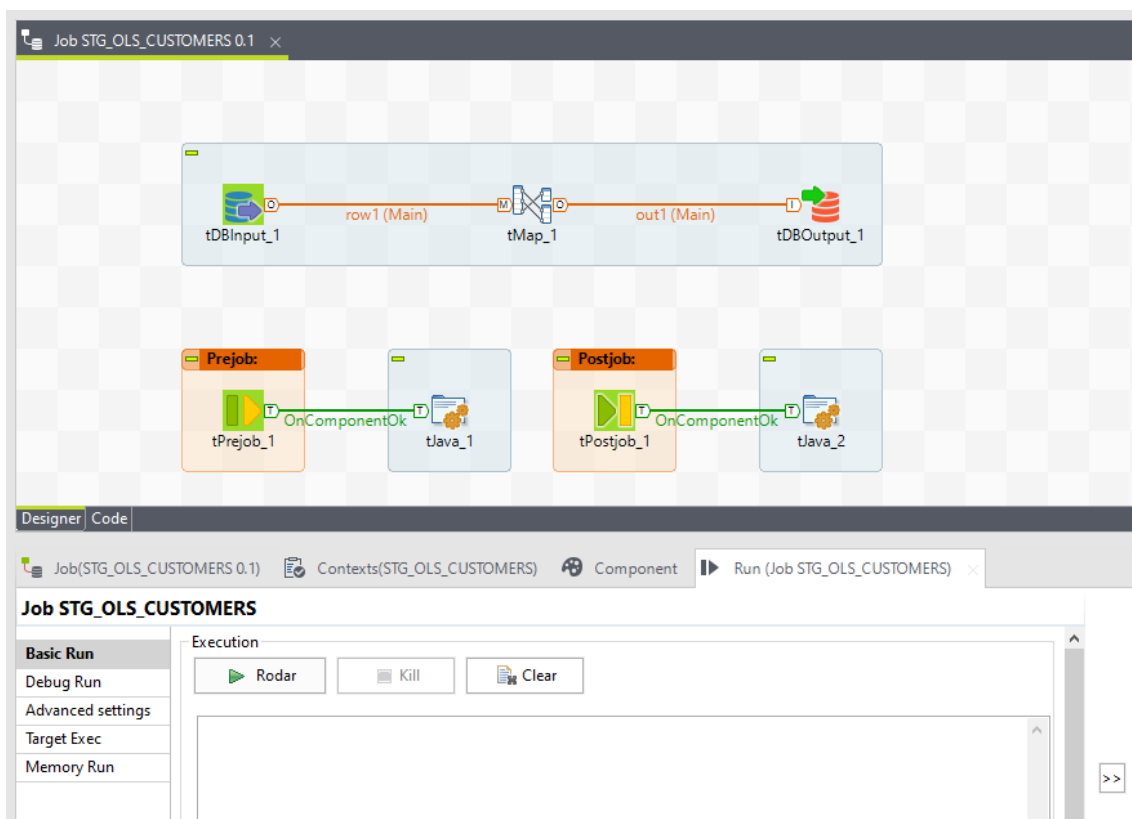
2.2. DMOls STAGGINGS

Para a carga das stages, a versão do python foi bem trabalhosa dado todas as validações e os processos assíncronos desenvolvidos, para a versão do Talend foi bem mais simples podendo entregar valor rapidamente para o projeto.

Levando em conta a “Stage Ols Customers”, a primeira coisa é criar o schema no Sql Server chamado “StgOls”, este schema do Sql Server vai ser responsável apenas por armazenar as stages do projeto.

Após isso, para a versão do Talend basta utilizar o componente de Input que é a conexão com o banco e a execução de uma query onde os dados serão propagados a partir de uma saída principal (chamada de fluxo “main”), e a partir disso, propaga-se a saída para um componente chamado TMap, que nada mais é que a etapa de fazer alguma eventual transformação, e por fim, o Output é as consultas de insert/update em algum lugar.

O fluxo geral está representado na imagem abaixo, note que o PreJob e PostJob são apenas para o mapeamento de Logs, igual o “print” do python que utilizei.

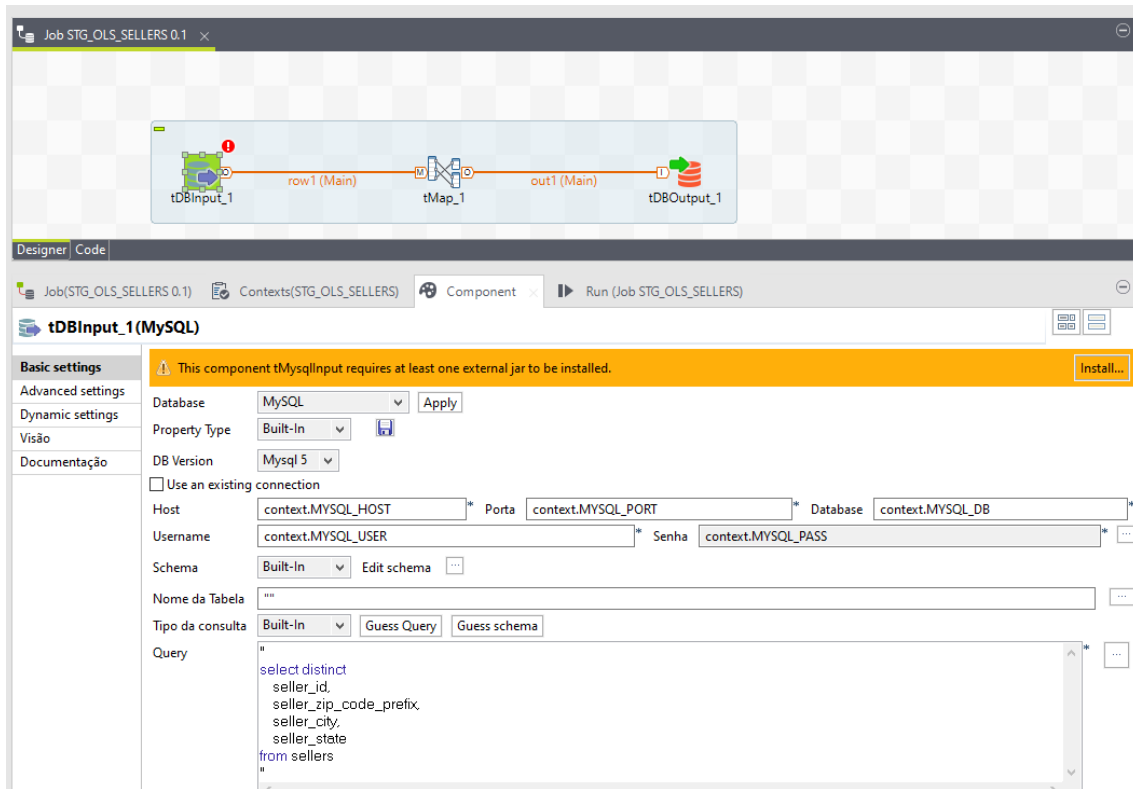


Esta da imagem é o fluxo principal, dos dados de uma Stage.

Fazendo as analogias com a versão do Python, o input é a consulta realizada com a classe Database, a parte do TMap é justamente a transformação e a criação da Data de

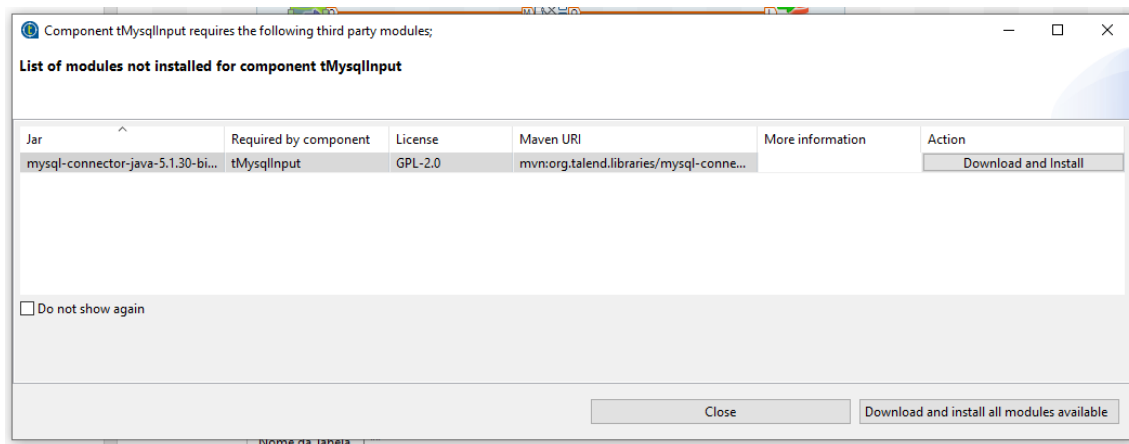
Processamento e o Output é a carga via método `to_sql` da classe `DataFrame` onde a carga vai acontecer abstraído no método `to_sql`.

O Componente Input do Talend recebe alguns parâmetros importantes, como as credenciais e o select com os dados que vão ser propagados no fluxo main, considere na imagem abaixo o fluxo agora para a Stage Ols Sellers que vai ser executada no banco MySQL agora.



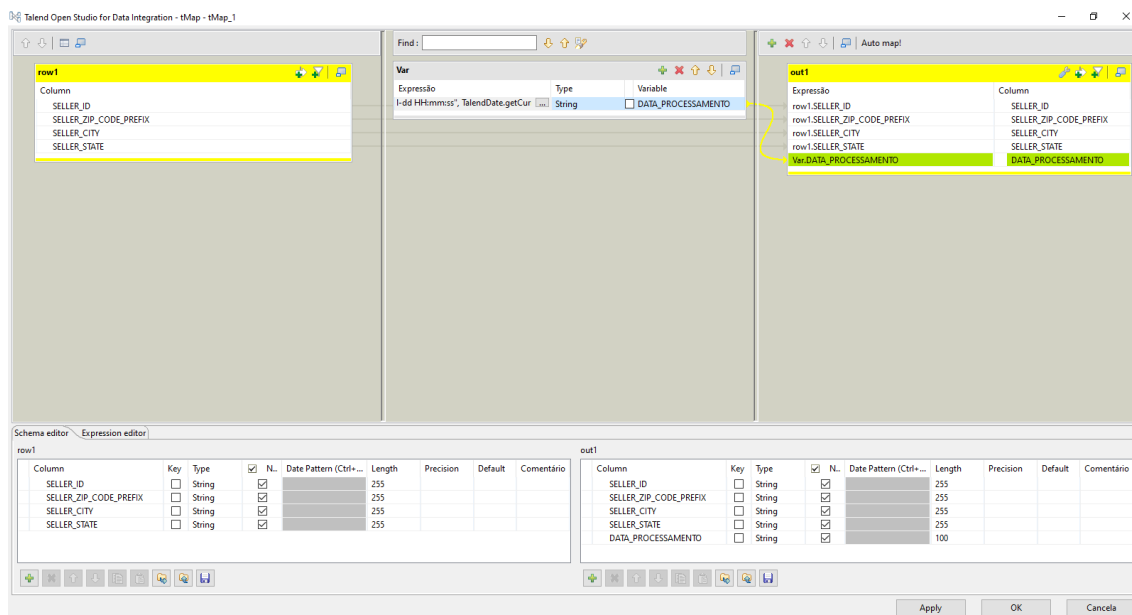
Neste caso, existem alguns parâmetros de conexões que estão nas variáveis context que foi definido anteriormente e a query em formato string (por isso os aspas) a ser executada e propagada pelo fluxo "main" como indica a linha laranja com a setinha e o "M" dentro.

Repare que para o MySQL, é necessário instalar uma classe java (arquivo java) que vai ser responsável pela conexão com o banco de dados, para isso é só clicar em install e o próprio Talend vai ser responsável por coletar a versão da classe correta para a versão do conector do banco selecionada no dropdown chamado DB Version.



Fazendo as analogias com a versão Python, todo esse processamento de ler o arquivo com a consulta sql e coletar a conexão com o banco já é abstraído dentro das classes.

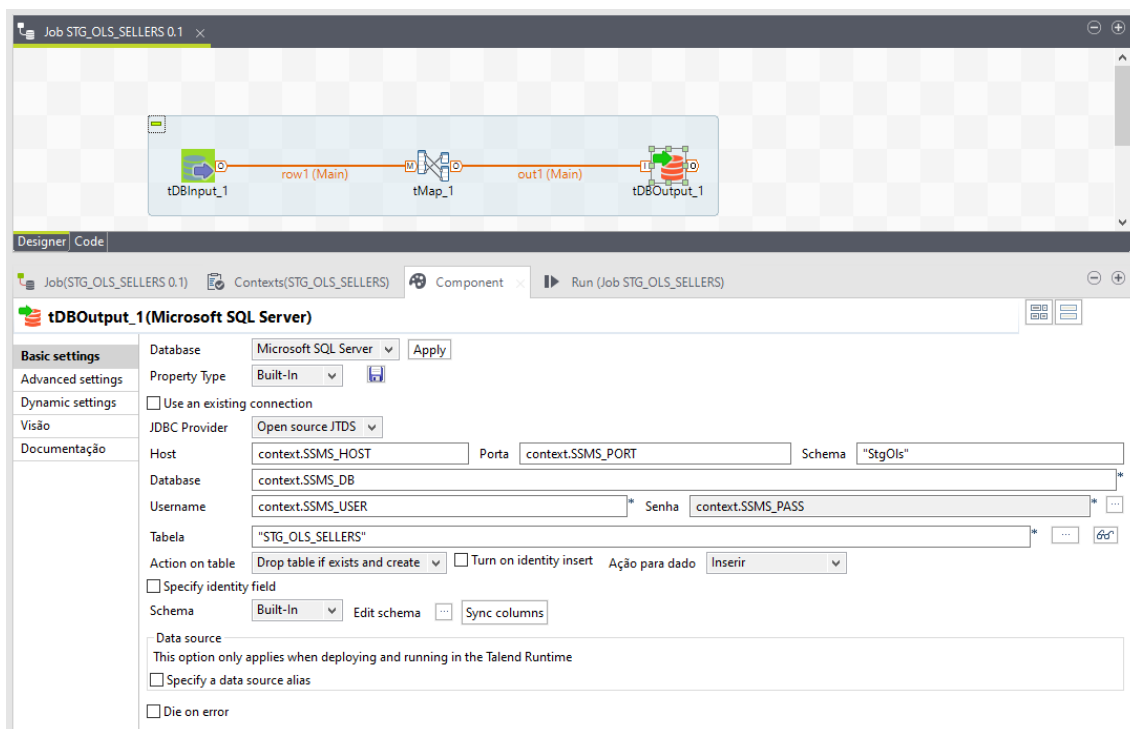
Agora, já dentro do TMap, é apenas propagado os dados do fluxo main e incluído um código em java que irá gerar a data e hora de quando o Job foi executado, este mesmo já estão na parte do Python também abstraídos dentro da classe.



Fiz dessa forma pois, como os dados da stage são gerados na hora que o job foi executado, devemos limpar a stage novamente no destino, e sem a data de processamento não conseguimos ver na Dw se a stage foi executada ou não, assim na tabela final da stage existe mais uma coluna chamada DATA_PROCESSAMENTO responsável por indicar a data e horas que a tabela foi carregada, se o processo roda de hora em hora, espero que a cada gora que visualizar a tabela, sejam dados novos e para verificar isso basta visualizar esta coluna.

Tanto na versão Talend quanto na versão de Python é deletado a tabela stage e criada uma novamente do zero com os dados coletados da fonte como está representado no output do Job em Talend na imagem abaixo.

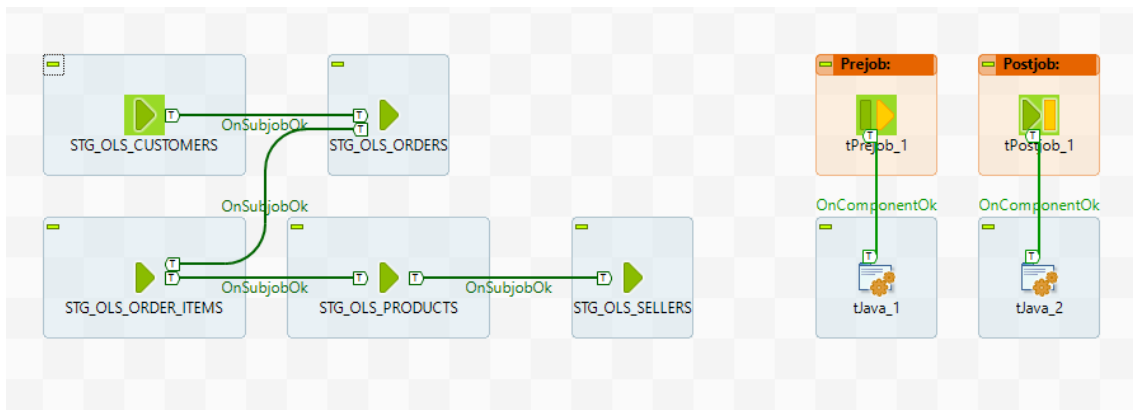
Define-se o nome da tabela, schemas e as credenciais, existem muitas outras opções interessantes a serem exploradas como “Die on Error” que irá parar o processo de ETL caso o script apresenta algum erro, a mesma solução foi desenvolvida diferente em python com um decorador responsável por um retry, caso não der certo mesmo com o retry a coleta da stage, então essa stage vai ser seguida no fluxo e o pipeline não vai quebrar, e na hora da inserção essa stage que deu problema não vai ser executada.



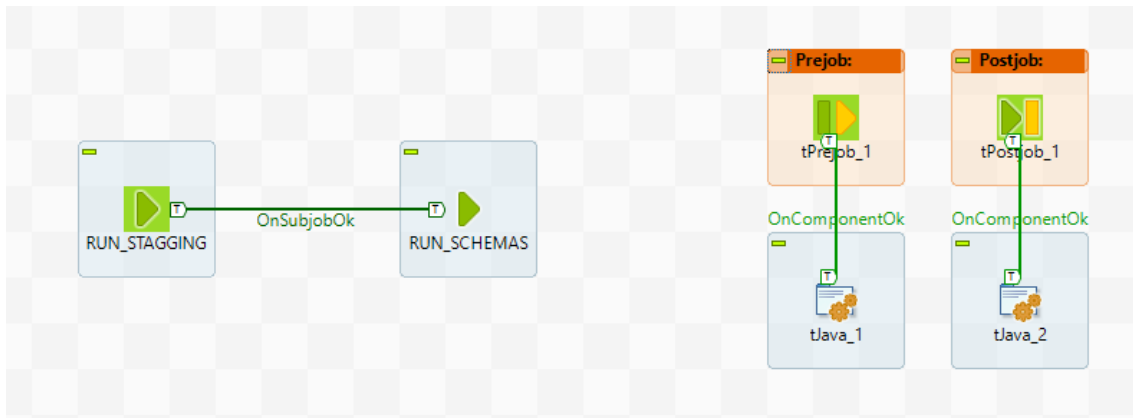
E, por fim, é deletado a tabela a cada vez que o Talend ou o Python são executados.

Existe mais um detalhe, na versão do Talend, todos os tipos de dados das colunas são no formato varchar(255), assim pelo menos caso de problema de conversão de tipo em algum momento do ETL as stages já estão previamente carregadas.

Seguindo esta mesma ideia, é gerado um Job específico com todas as stages, esse job é responsável por executar todas as stages em sequência e não em async igual a versão do Python, basta executar este Job e todas as stages são iniciadas (Job Run_Staggings do Talend).



As stages já foram configuradas, basta executar / chamar elas no main, igual fiz no arquivo main.py da versão de Python, só que para o Talend seria um Job novo chamado MAIN, similar a algo assim:



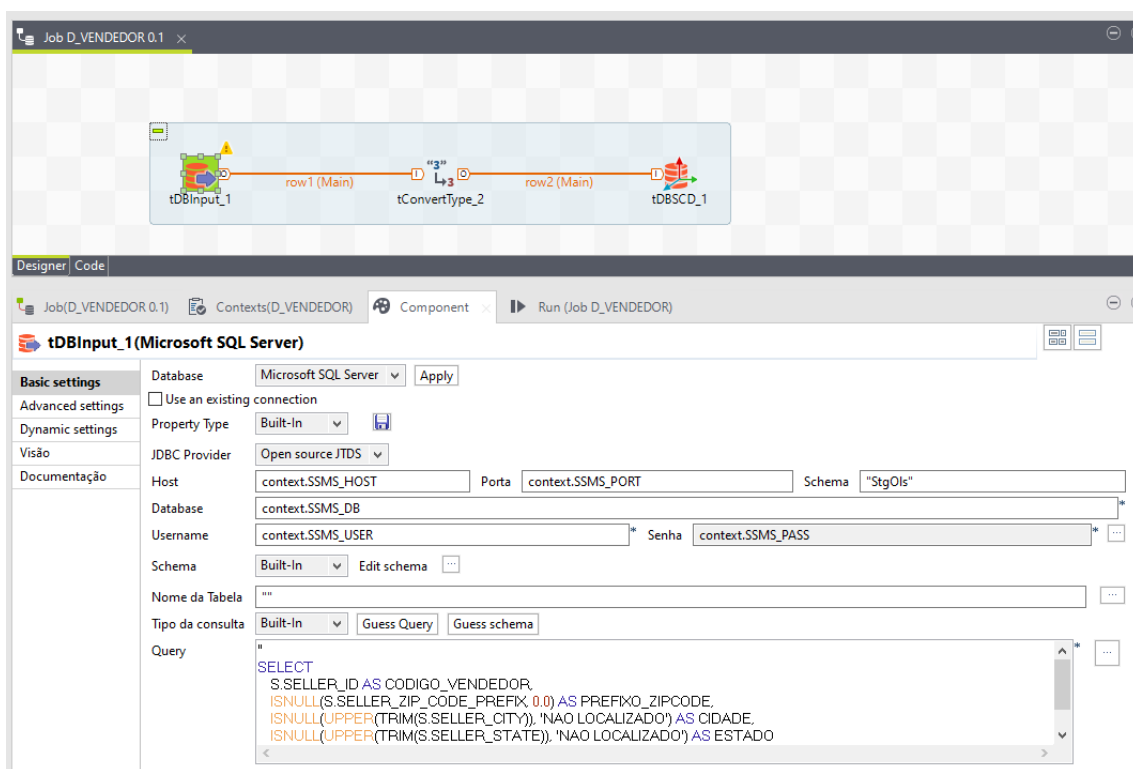
Nesse caso, só existirá a RUN_STAGGING seguindo o texto, precisa-se criar agora as modelagens das dimensões e fatos.

2.3. DMOls SCHEMAS

Essa é a etapa de carga das dimensões e das fatos, essa parte é bem simples também, só que com alguns truques, as linhas (os dados) das dimensões e fatos devem ser atualizados ou inseridos, exceto algumas exceções como fatos que são limpas completamente e carregadas do zero e tabelas que são incrementais apenas, mas na maioria das vezes, principalmente para as dimensões elas nunca devem ser deletadas pois é possível perder a referência do SK que vai ser citado ao decorrer do texto, essa etapa precisa-se definir em consultas sql de inserção ou atualização para a versão do Python, para a versão do Talend estas já estão abstraídas dentro dos componentes SCD.

Lembrando que agora é utilizado as tabelas stages para montar as tabelas dimensões e fatos, estas tabelas já estão no schema StgOls dentro do destino.

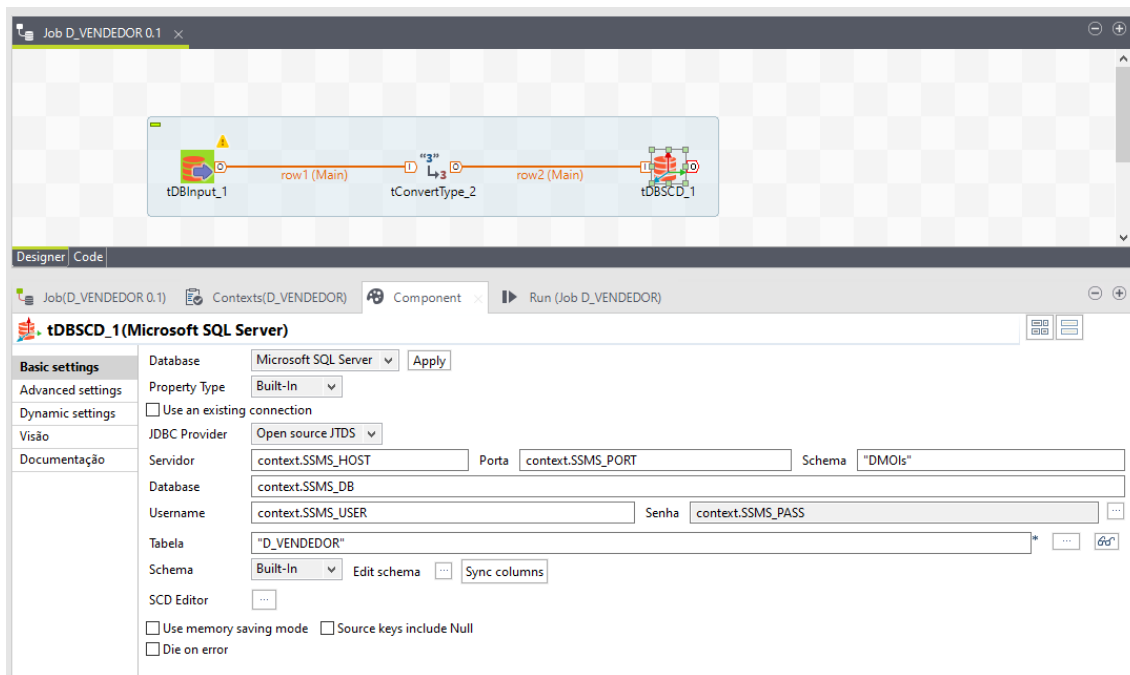
Neste caso, para o Talend, apenas utiliza o select dentro do componente de input que vai ser propagado ao longo do fluxo main, em vez de utilizar o componente de TMap, na imagem é utilizado um componente de casting para transformar os tipos de dados antes de salvar no banco SQL Server.



Para este caso, tanto o Schema quanto a tabela já devem estar criadas dentro do banco de dados com os seus respectivos index também, igual está na pasta de create_tables dentro da pasta sql no repositório do github.

Para a versão do Python, primeiro é executado o Insert e após isso o Update em duas consultas separadas e não no mesmo fluxo como o Talend, neste caso o Talend já faz essa tratativa para você.

Como citei, a tabela não vai ser deletada e sim inserido dados novos dado uma chave de verificação “id” ou atualizado os mesmos dados com este mesmo “id”.



O componente que utilizei foi o SCD e também ele recebe alguns parâmetros.

A Source Key é a chave de atualização da tabela, o Surrogate Keys é como se fosse o “id” na modelagem dimensional da dimensão ou fato que é utilizada nas referências de dimensões ou fatos no modelo star schema ou snowflake schema.

E nas colunas Type 0..3 são os tipos da dimensão, neste caso, é apenas uma tabela que vai ser de referência de vendedores, então vai ser apenas inserido coisa nova ou atualizado às linhas já existentes desta tabela chamada DMOIs.D_VENDEDOR pela chave CODIGO_VENDEDOR como está na imagem abaixo.

Editor do componente SCD

Filtro

Unused

Source keys

CODIGO_VENDEDOR

Surrogate keys

name SK_D_VENDEDOR

creation Auto increment

complement

Type 0 fields

Type 1 fields

CIDADE
ESTADO
PREFIXO_ZIPCODE

Type 2 fields

Versioning

type	name	creation	complement
start	scd_start	Job start time	
end	scd_end	NULL	
<input type="checkbox"/> version	scd_version		
<input type="checkbox"/> active	scd_active		

Type 3 fields

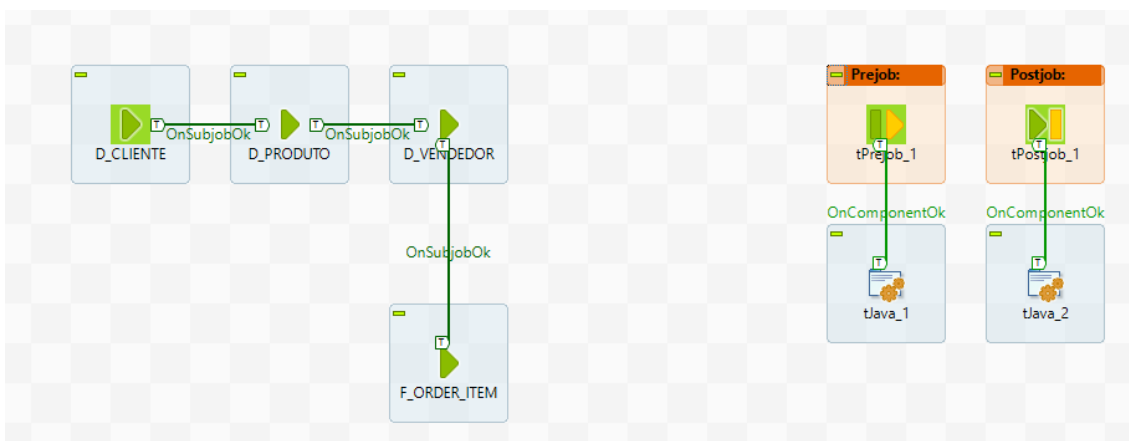
current value	previous value

OK Cancel

Toda esta lógica é criada por consultas sql de insert ou update na versão do Python.

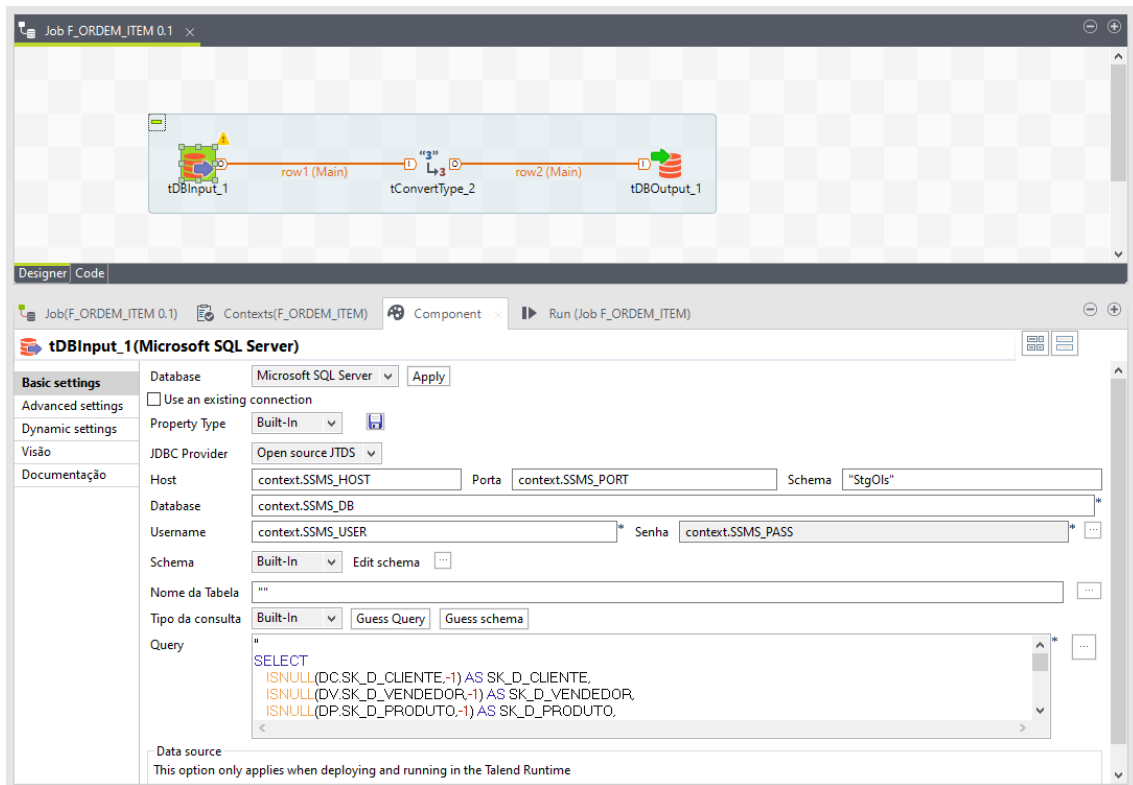
Por fim, basta criar as demais dimensões e incluir elas no job RUN_SCHEMAS, finalizando as stages e a modelagem.

No caso do Python e também do Talend, coloquei no mesmo “Job” / classe, assim fica correto a analogia das duas ferramentas para fazer o mesmo processo de ETL, pois a versão do python irá carregar os selects de dimensões e executar eles e após isso ja executar em seguida as consultas das fatos, que seguem a mesma analogia do Talend apresentado na imagem abaixo.

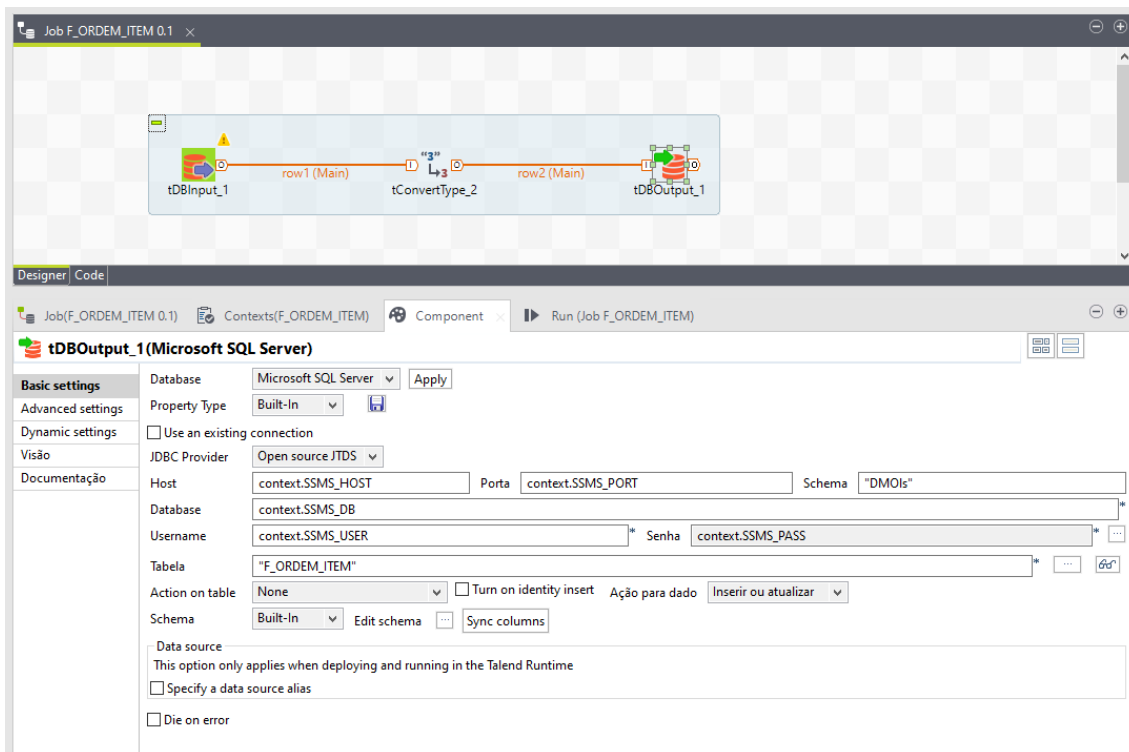


Para a Fato, utilizei a mesma arquitetura das stages, apenas mudando o Output para atualização e inserção ao invés da deleção como é feita nas stages.

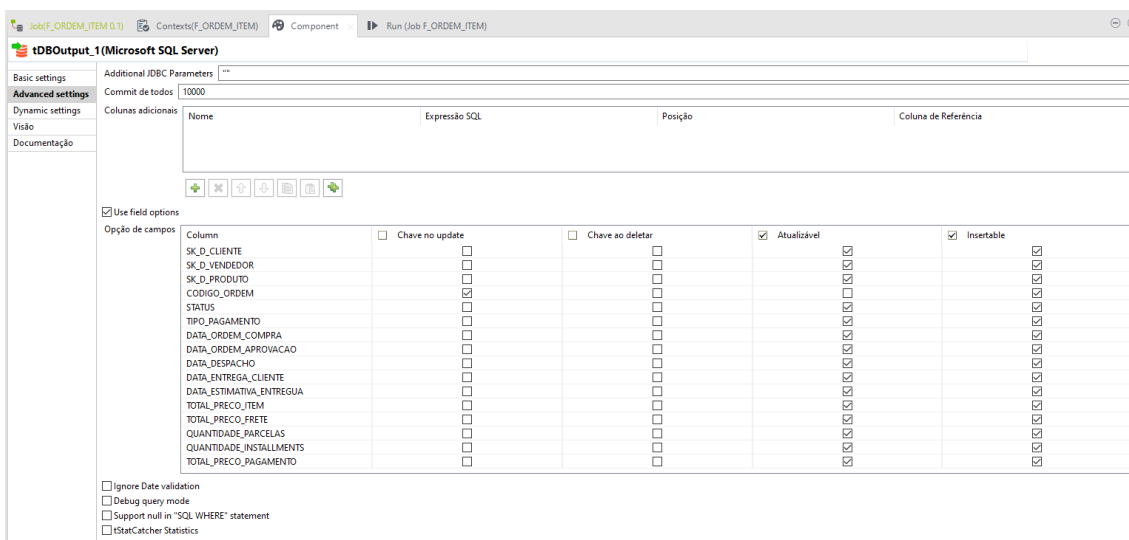
Para o Input é a mesma configuração das stages, mas não coletando da fonte e sim da Dw, pois os dados já estão no schema **StgOls** e já foram previamente **tratados**.



A única atualização neste caso é o output, nas opções Action on Table e Action on Data (o meu está em Português pela metade).



A ação é inserir ou atualizar, sem deletar a tabela, apenas inserir novas linhas ou atualizar baseado em alguma chave de verificação, para definir as chaves de atualização é nas opções avançadas do componente onde todas são possíveis de serem inseridas na tabela mas a coluna CODIGO_ORDEM é a chave “id” em outras palavras é a coluna de atualização das linhas da fato que vai garantir que vai ser inserido coisa nova ou atualizado linhas já existentes.



Por exemplo, o acompanhamento de uma venda, se a coluna “venda_status” está em “processamento” e foi salva na fato com este status, “logo logo” este mesmo status para

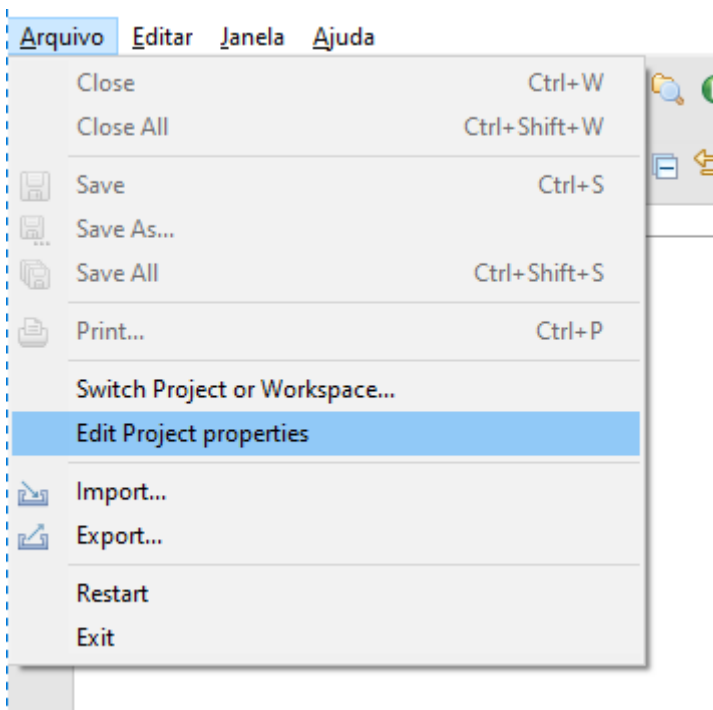
esta mesma venda pode ser atualizada, para venda concluída ou distratada ou em análise, etc, a linha vai sofrer atualização, e portanto deve ser atualizada na fato.

2.4. DMOls EXECUÇÃO DO ETL

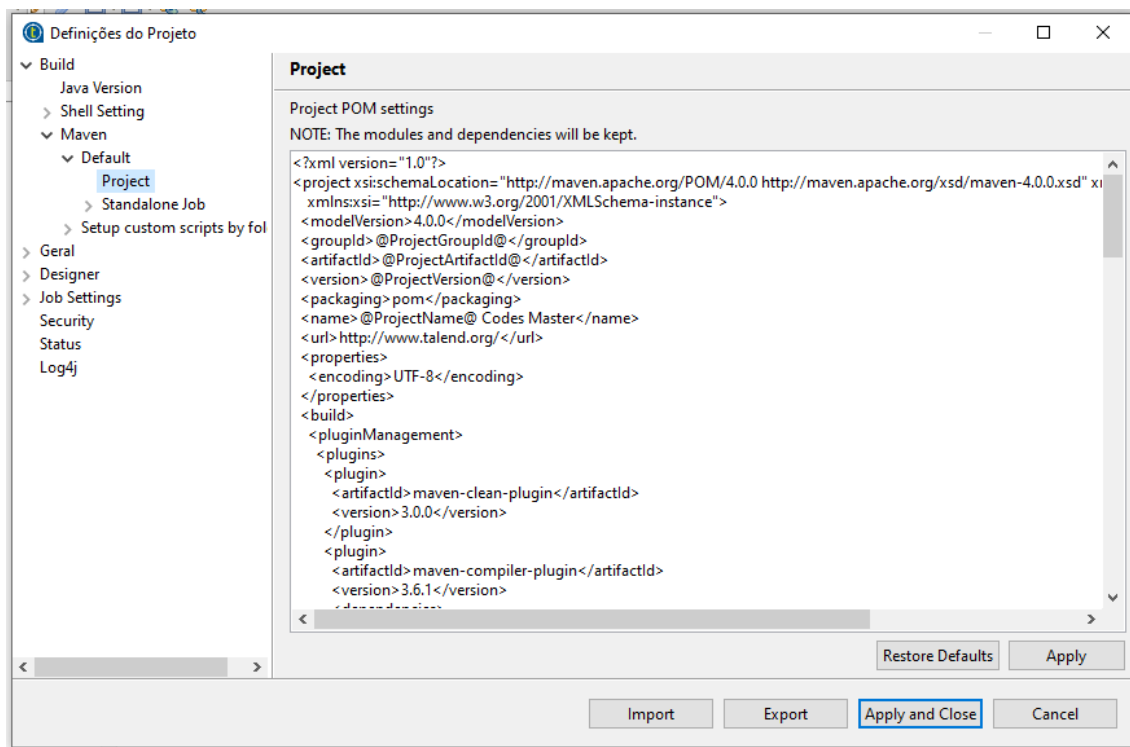
Para finalizar, basta executar o Job main ou o script main.py, ambos irão executar o ETL do projeto DMOls se tudo estiver devidamente configurado como as bases de dados.

Para gerar um script a ser executado sem precisar abrir o Talend e executar manualmente, deve-se seguir alguns passos.

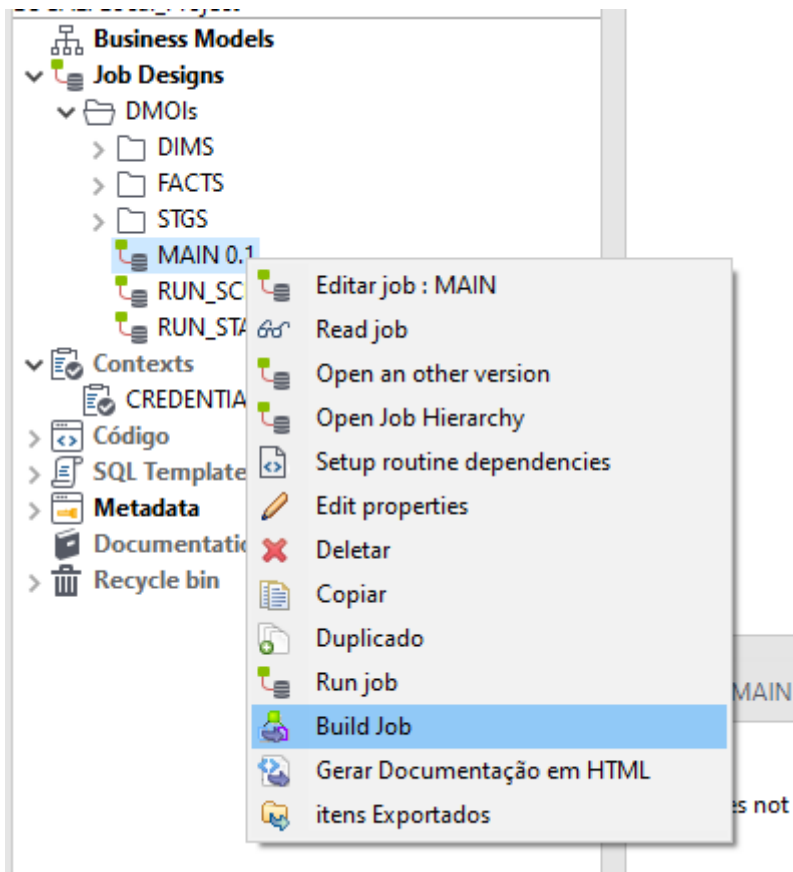
Para a versão do Talend, o primeiro é a atualização dos “poms” do java, a primeira coisa a se fazer é fechar todos os Jobs abertos para evitar erros, e após isso seguir o passo a passo das imagens abaixo.



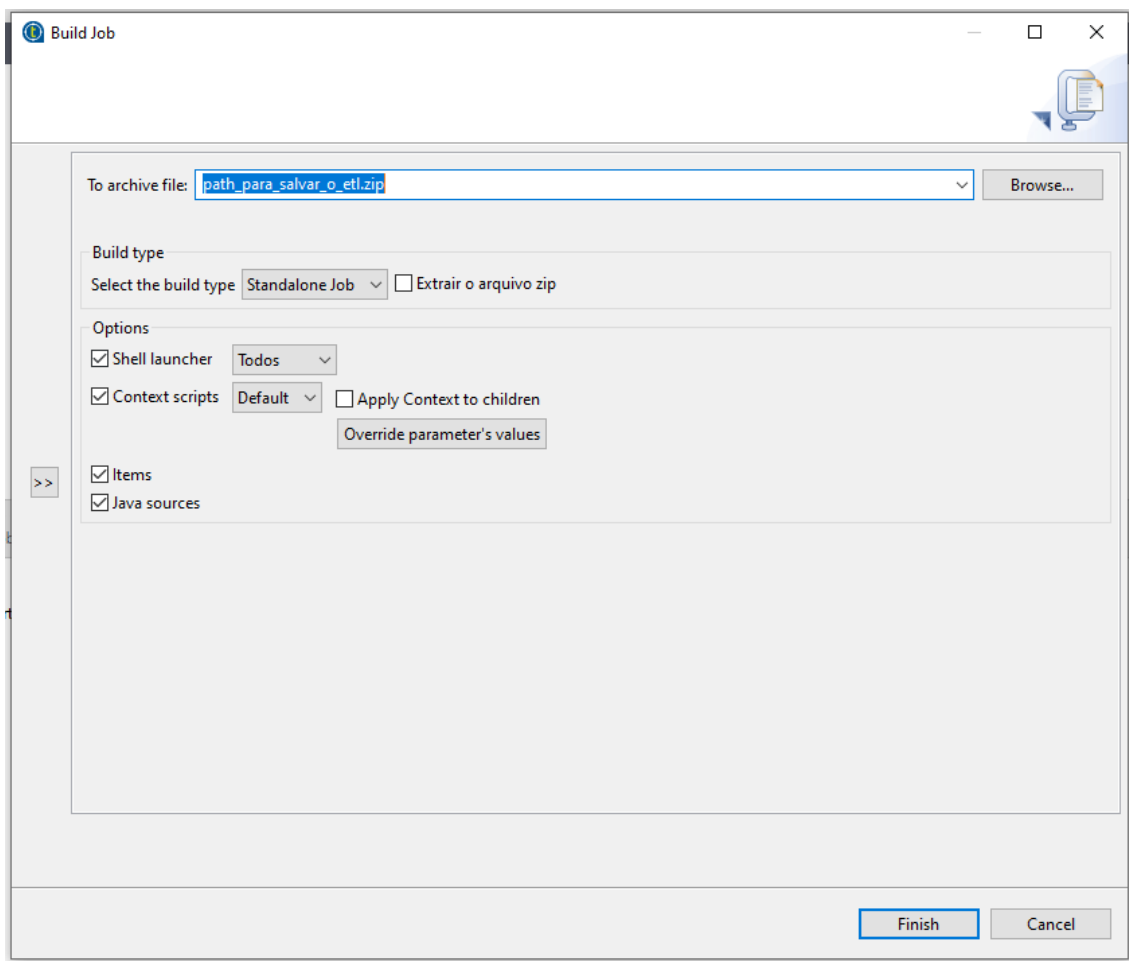
E navegar até a opção “Project” e por fim **Apply and Close**.



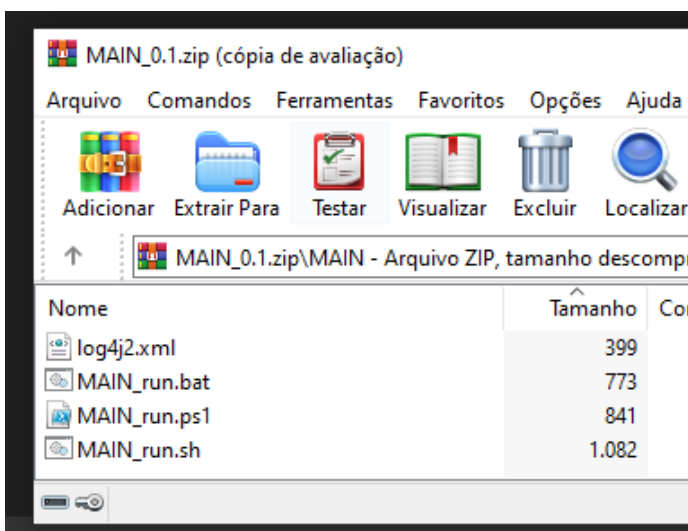
Após isso, como toda a lógica está no main, você pode gerar os scripts para serem executados seguindo este passo a passo.



Então, irá abrir um popup para você especificar um path para salvar e clicar em finish. As demais opções são para especificar quais exportações irá fazer e quais context (credenciais) vai ser utilizada.



Chamei o arquivo de Main.zip. Dentro deste arquivo existem os arquivos MAIN_run.*, assim você consegue executar o job em diferentes s.o, apenas executando o script, ex: no windows pelo cmd o arquivo MAIN_run.bat e pelo powershell o arquivo MAIN_run.ps1.



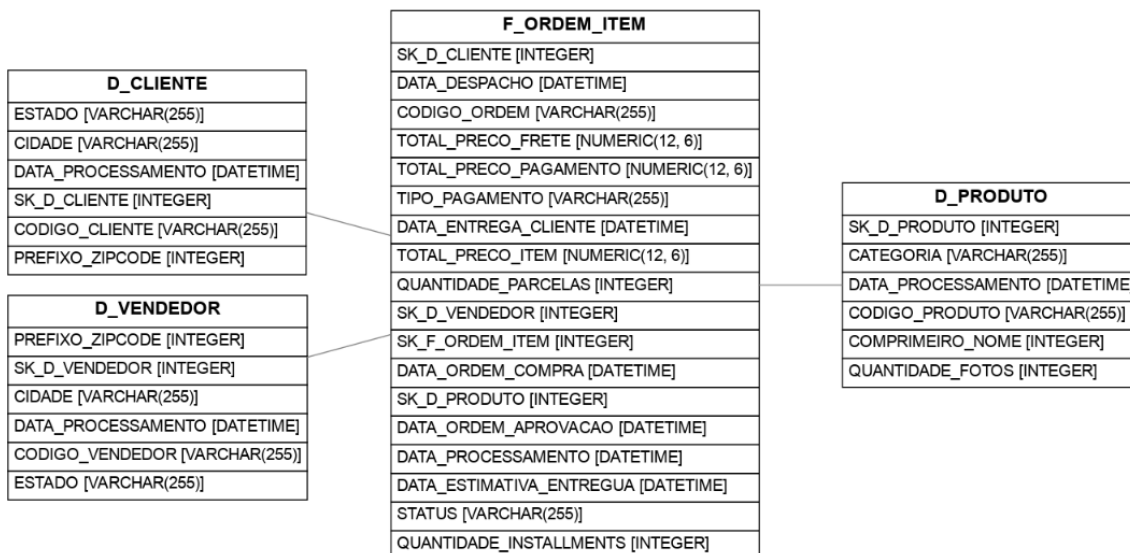
Já para a versão do Python, basta criar um script .sh ou .ps1 com as variáveis de ambiente das credenciais e a execução do script dentro de um ambiente de desenvolvimento como o do poetry ou, por exemplo, em um container com a ajuda do docker.

2.5. DMOls Docs

Quando o processo de ETL já executa a algum tempo, é muito importante a catalogação desse dado, informar qual é necessidade da criação deste processo de ETL, catalogar métricas do banco de dados, por exemplo, se a tabela é gigantesca, é necessário um processo que verifique o crescimento dessa tabela diariamente para verificar se existem possíveis anomalias durante a carga e diversos outros fatores que influenciam e devem ser acompanhados ao longo do tempo.

A forma mais simples de catalogação é a elaboração do Schema, no talend não existem ferramentas para desenvolver o esquema de alguma tabela (não até onde eu saiba), mas da de exportar a interface gráfica dele como um relatório de como o processo de ETL é construído e roda hoje em dia.

Gerando o diagrama automático da DMOls para a única fato existente, temos a seguinte imagem:



O diagrama foi gerado pela ferramenta graphviz e er_diagram do Python, mas, para as demais fatos de outros DM, o código já vai trabalhar automaticamente, o código está na pasta catalog do repositório no github.

Outro processo importante é o acompanhamento diário de determinadas tabelas, com esse acompanhamento podemos montar gráficos, estimar tendências, analisar anomalias e também fazer previsões e estimativas de custos, inúmeras vantagens, a ideia é bem simples.

Esse é o resultado do processo de ETL que irá no banco, para todas as tabelas, stages, dimensões e fatos e coletar informações dessas tabelas, ou pode ser apenas para as fatos, enfim, não importa, o mais importante é essas análises feitas com esses dados, nesse caso abaixo é apenas uma linha de uma tabela fato, nela podemos analisar números como a quantidade de linhas e métricas voltadas ao banco em KB (deixei a unidade de

medida para não gerar dúvidas aos analistas) da tabela específica, métricas do banco de dados SQL Server.

O formato está invertido apenas para análise, na DW vai ser armazenado em formato de linha mesmo e não em coluna.

df_history_table_sizes.tail(1).T

✓ 0.0s

	DM	DMOIs
ULTIMA_DATA_PROCESSAMENTO	2023-06-22 23:02:39.917000	
NOME	F_ORDEM_ITEM	
LINHAS	103616	
TAMANHO_RESERVADO	23440 KB	
TAMANHO_DADO	17544 KB	
TAMANHO_INDEX	5672 KB	
SEM_USO	224 KB	
DATA_BASE	2023-06-28	

Note que a última coluna, chamada DATA_BASE é um indicativo.

Essa coluna indica que, nesse dia (28 de Junho de 2023) a tabela F_ORDEM_ITEM tinha essas respectivas métricas, amanhã quando o processo de ETL executar novamente, essa tabela vai possuir outras métricas, e o indicativo é justamente a DATA_BASE.

No final, temos essa tabela na Dw.

*<localhost> Script-12 x

SELECT * FROM DMControl.TABLE_SIZES

Results 1 x

	PROCESSAMENTO	NOME	LINHAS	TAMANHO_RESERVADO	TAMANHO_DADO	TAMANHO_INDEX	SEM_USO	DATA_BASE
1	01	STG_OLS_CUSTOMERS	99441	12680 KB	12632 KB	8 KB	40 KB	2023-06-28
2	04	STG_OLS_ORDER_ITEMS	103616	34568 KB	34544 KB	8 KB	16 KB	2023-06-28
3	03.000	STG_OLS_ORDERS	99441	16520 KB	16496 KB	8 KB	16 KB	2023-06-28
4	09	STG_OLS_PRODUCTS	32951	3272 KB	3232 KB	8 KB	32 KB	2023-06-28
5	08	STG_OLS_SELLERS	3095	328 KB	288 KB	8 KB	32 KB	2023-06-28
6	01.000	D_CLIENTE	99443	12880 KB	7640 KB	5048 KB	192 KB	2023-06-28
7	06.000	D_PRODUTO	32953	4560 KB	2688 KB	1696 KB	176 KB	2023-06-28
8	09.000	D_VENDEDOR	3097	592 KB	240 KB	192 KB	160 KB	2023-06-28
9	09.000	F_ORDEM_ITEM	103616	23440 KB	17544 KB	5672 KB	224 KB	2023-06-28

Para um exemplo mais concreto do que acabei de falar, considere esse caso real desta mesma análise:

SELECT * FROM DwControle.ETL_ACOMPANHAMENTO_TABELAS

WHERE NOME = 'F_TERMOS' ORDER BY DATA_BASE

Messages

ULTIMA_DATA_PROCES...	NOME	LINHAS	TAMANHO_RESERVADO	TAMANHO_DADO	TAMANHO_INDEX	SEM_USO	DATA_BASE	DATA_PROCESSAMENTO
2023-06-05 01:18:27.000	F_TERMOS	85619	33600 KB	15048 KB	10024 KB	8528 KB	2023-06-05	2023-06-05 10:14:54.880
2023-06-06 01:16:54.000	F_TERMOS	85624	33600 KB	15048 KB	10024 KB	8528 KB	2023-06-06	2023-06-06 10:13:52.680
2023-06-07 01:17:14.000	F_TERMOS	85634	33600 KB	15048 KB	10024 KB	8528 KB	2023-06-07	2023-06-07 10:14:55.647
2023-06-08 01:23:31.000	F_TERMOS	85654	33672 KB	15056 KB	10032 KB	8584 KB	2023-06-08	2023-06-08 10:14:14.387
2023-06-09 01:19:29.000	F_TERMOS	85662	33672 KB	15056 KB	10032 KB	8584 KB	2023-06-09	2023-06-09 10:19:25.860
2023-06-10 01:27:48.000	F_TERMOS	85666	33672 KB	15056 KB	10032 KB	8584 KB	2023-06-10	2023-06-10 10:15:12.107
2023-06-11 01:18:31.000	F_TERMOS	85670	33672 KB	15056 KB	10032 KB	8584 KB	2023-06-11	2023-06-11 10:35:16.917
2023-06-12 01:15:54.000	F_TERMOS	85678	33672 KB	15056 KB	10032 KB	8584 KB	2023-06-12	2023-06-12 10:20:12.053
2023-06-13 01:21:34.000	F_TERMOS	86486	33952 KB	15152 KB	10184 KB	8616 KB	2023-06-13	2023-06-13 10:14:32.323
2023-06-14 01:13:00.000	F_TERMOS	86531	34088 KB	15152 KB	10208 KB	8728 KB	2023-06-14	2023-06-14 10:14:52.037
2023-06-15 01:24:24.000	F_TERMOS	86551	34088 KB	15160 KB	10216 KB	8712 KB	2023-06-15	2023-06-15 10:14:56.190
2023-06-16 01:27:57.000	F_TERMOS	86571	34088 KB	15160 KB	10216 KB	8712 KB	2023-06-16	2023-06-16 10:14:08.117
2023-06-17 01:34:06.000	F_TERMOS	86588	34088 KB	15160 KB	10216 KB	8712 KB	2023-06-17	2023-06-17 10:15:20.577
2023-06-18 01:37:22.000	F_TERMOS	86611	34088 KB	15168 KB	10216 KB	8704 KB	2023-06-18	2023-06-18 10:14:34.767
2023-06-19 01:18:20.000	F_TERMOS	86615	34088 KB	15168 KB	10216 KB	8704 KB	2023-06-19	2023-06-19 10:36:44.710
2023-06-20 01:24:18.000	F_TERMOS	86623	34088 KB	15168 KB	10216 KB	8704 KB	2023-06-20	2023-06-20 10:15:00.277
2023-06-21 01:29:37.000	F_TERMOS	86636	34088 KB	15168 KB	10224 KB	8696 KB	2023-06-21	2023-06-21 16:41:55.383
2023-06-24 02:22:20.000	F_TERMOS	86706	34088 KB	15176 KB	10240 KB	8672 KB	2023-06-24	2023-06-24 10:15:04.883
2023-06-25 01:37:54.000	F_TERMOS	86760	34088 KB	15184 KB	10248 KB	8656 KB	2023-06-25	2023-06-25 10:14:00.230
2023-06-26 02:28:58.000	F_TERMOS	86765	34088 KB	15184 KB	10248 KB	8656 KB	2023-06-26	2023-06-26 10:13:17.730
2023-06-27 01:21:56.000	F_TERMOS	86786	34088 KB	15184 KB	10248 KB	8656 KB	2023-06-27	2023-06-27 12:10:21.293
2023-06-28 01:23:06.000	F_TERMOS	86823	34088 KB	15192 KB	10248 KB	8648 KB	2023-06-28	2023-06-28 10:14:40.157

Nesse mesmo caso, apenas o nome do schema e da tabela mudaram, mas as métricas são as mesmas, para a tabela, “F_TERMOS”, uma tabela qualquer, poderia ser staging, etc, que é atualizada diariamente, nesse exemplo existem dados de acompanhamento de mais de um mês praticamente.