Documento de Referência

Funcional-Técnico

Cliente: **Zinzane**

Projeto: **AGING**

Objetivo: Realizar o processamento de cálculo do aging.

Versão: 1.0 (Jun/2016)

1. Escopo do Projeto

Extrair os dados da base do Cliente em um servidor remoto e movê-los para uma stage area no servidor da TDVTEC. Em seguida, validar e tratar estes dados antes de movê-los para o repositório de data mart. E por fim efetuar o processamento de cálculo do aging, com base nos seguintes critérios: por Loja e por Cia.

2. Descrição da Regra do Cálculo

O cálculo do aging consiste em determinar para cada peça armazenada hoje a quanto tempo ela está armazenada. Para isso precisamos para cada sku do estoque em cada loja da empresa varrer todos as entradas, da mais recente para a mais antiga, e identificar a data da entrada.

Armazenamos a data da entrada e a quantidade de peças, pois desta forma conseguimos mostrar a informação do Aging em diversos níveis e formas distintas de agregação.

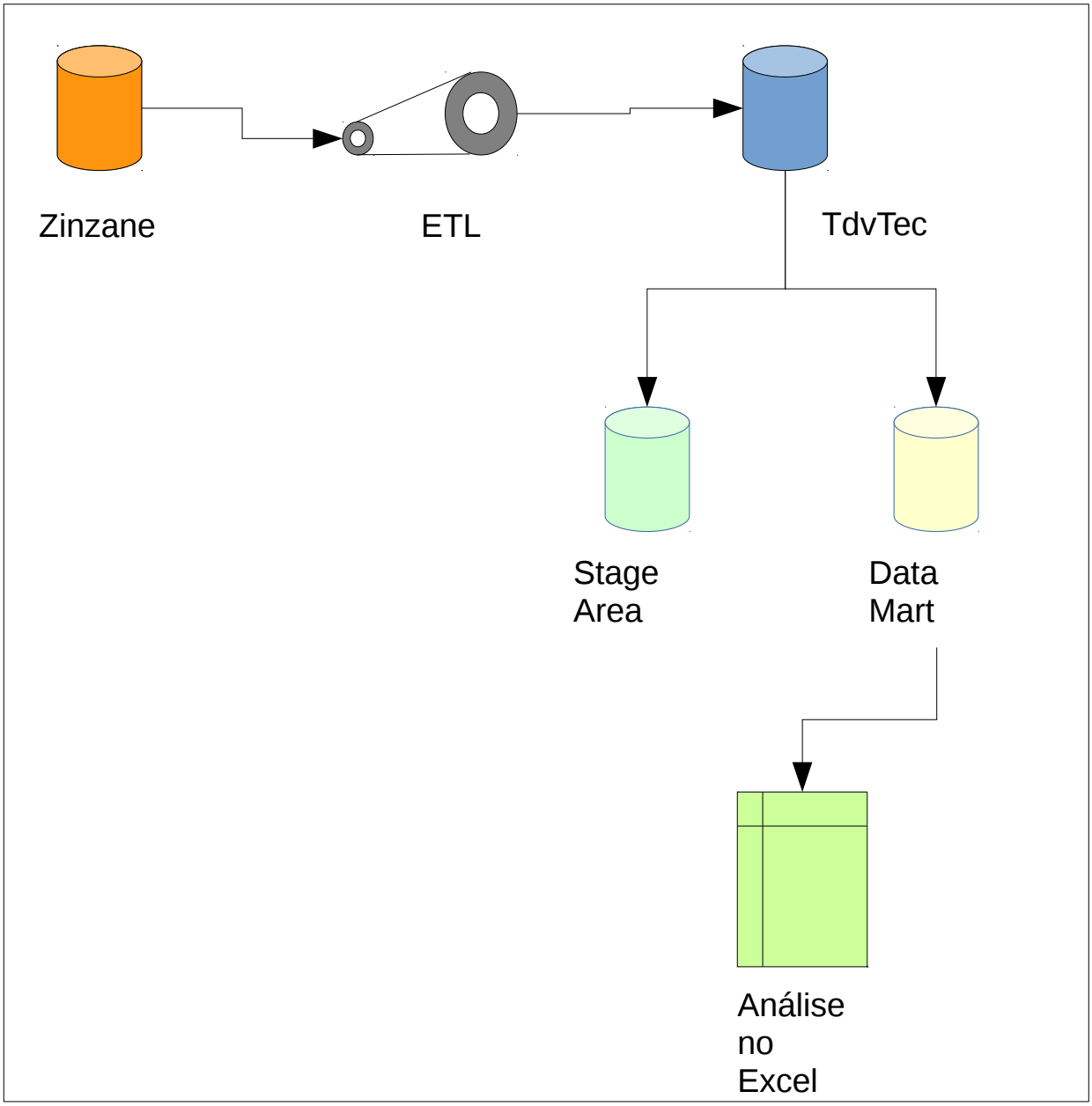
Principais Passos:

a) Criar um consolidado com todas as movimentações;
b) Abrir um cursor de estoque com todos os skus com estoque positivo, ordenar por local e sku (apenas para facilitar caso o processamento seja interrompido);
c) Para cada linha do cursor de estoque deve-se pegar o próximo sku, armazenar o estoque atual em uma variável de controle;
d) Abrir o cursor consolidado de movimentações montado no passo anterior e percorrer o cursor de movimentações de estoque. Então, para cada linha do cursor de movimentação deve-se verificar se o saldo de estoque na variável de controle é maior que a quantidade recebida;
e) Caso sim, então grava na tabela resultado, reduzindo da variável de controle a quantidade de estoque da quantidade recebida;
f) Caso não, então grava na tabela resultado. Neste caso grava na tabela resultado, o valor da variável de controle do estoque e não o valor da tabela de movimentação;
g) Efetua o commit da transação, porém somente podemos fazer isso quando o estoque do sku em um local chegar a zero;
h) Passa para o próximo produto do local.

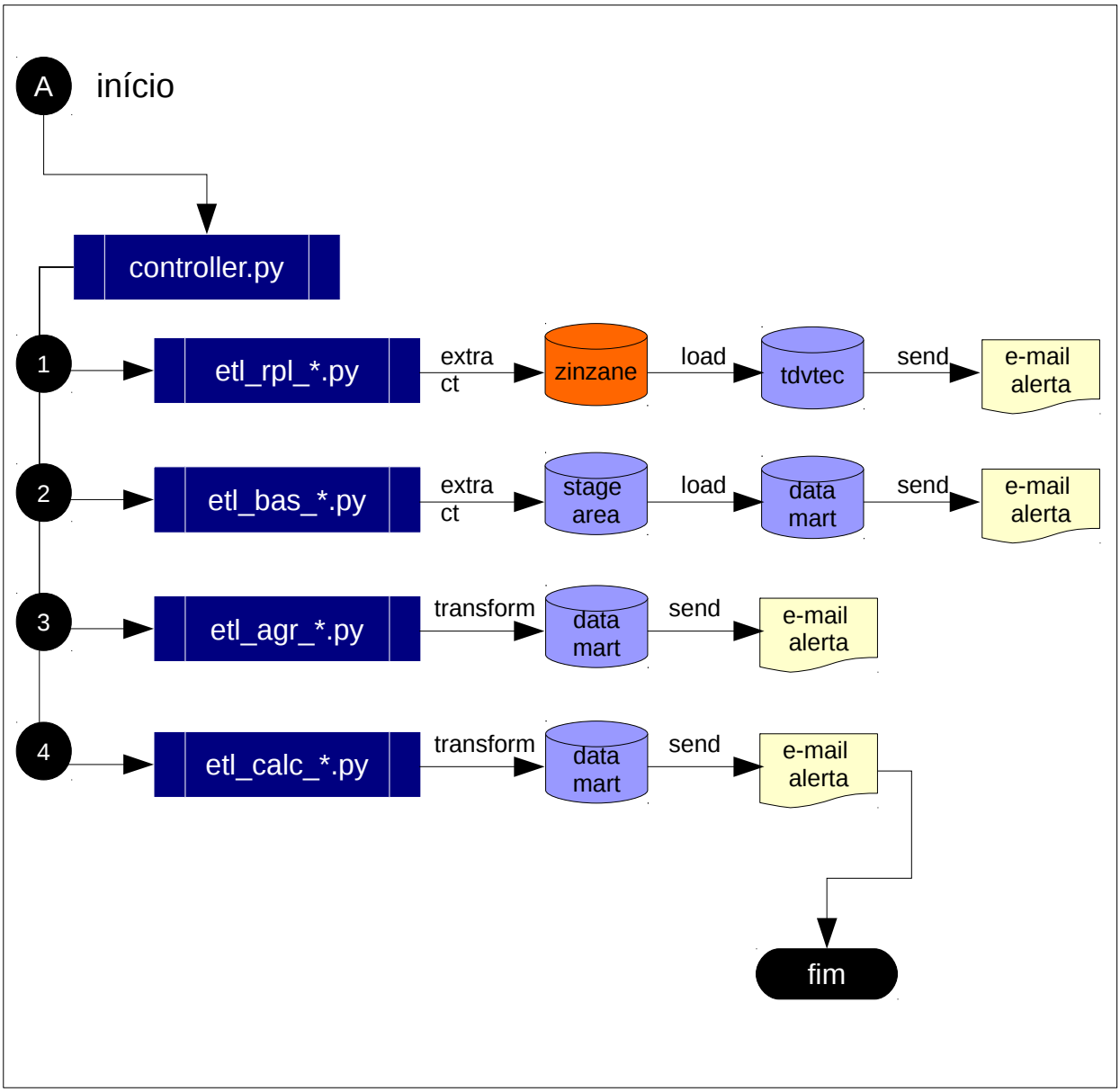
O processamento do cálculo é bastante longo e corre o risco de ser interrompido a qualquer momento. Então, é preciso criar um mecanismo de controle de processamento que permita retomar o processamento do ponto onde parou.

3. Desenho da Arquitetura da Solução

Arquitetura Técnica



Arquitetura Funcional



4. Pré-Requisitos

- ✔ Sistema Operacional = Windows/Linux
- ✔ SSH (Linux) ou OpenSSH (Windows)
- ✔ MySQL 5.5 ou superior
- ✔ Pycharm 3 ou superior
- ✔ Python 2.7.x ou superior
- ✔ Bibliotecas Python: pymssql, MySQLdb

Dados do Ambiente de Produção

- SSH – 75.126.137.89 [web525.webfaction.com]
  - user = tdvtec
  - pwd = 768aabb1
- MySQL – 75.126.143.46:30599
  - user = cli_zin_admin
  - pwd = adminzinzane
  - database = cli_zin

5. Nomenclaturas e Padrões de Desenvolvimento

Tabelas:

- RPL_: são as tabelas réplica dos dados da Zinzane, possuem os mesmos nomes de campos e todos os campos são do tipo varchar.
- CAD_: são dados cadastrados apenas na base gerencial, são utilizados para complementar ou especificar dados vindos do corporativo.
- CTRL_: são tabelas de controle da execução das interfaces.
- BAS_: é a segunda camada após a réplica, nas tabelas de base os dados ainda são armazenados na mesma granularidade que no corporativo. As colunas já são renomeadas para o padrão da plataforma de BI, os tipos dos campos já são ajustados para o dado que eles armazenam. Em alguns casos já existe a alteração de conceito nesta tabela. Ex.: Venda e Devolução de Venda são bases distintas no Corporativo, aqui ambas se juntam para formar a visão de venda com devolução.
- AGR_: bases agregadas a partir dos dados carregados nas tabelas BAS_. Daqui serão consumidos os dados gerenciais, apesar de não acessarem estas tabelas diretamente, o acesso será sempre encapsulado em views. Antes de criar uma tabela AGR_, é preciso avaliar se é necessário, pois em muitos casos podem-se agregar diretamente nas "DAT_" ou “VIS_”.
- DAT_ ou VIS_: são visões gerenciais. O acesso externo será sempre através destas tabelas. Uma DAT_ que faz agregações, pode com o tempo se tornar lenta, caso isso ocorra será preciso criar um processo noturno de agregação e um objeto AGR_. Neste momento a DAT_ passa apenas a retornar os dados sem processamento de agregação, permitindo assim o desacoplamento das visões externas das fontes de dados.

Scripts:

- ETL_: identifica o script como sendo uma rotina específica para extrair, tratar e carregar os dados da Zinzane para a TDVTEC.
- LOG_: identifica o script como sendo uma rotina específica para registrar o log de execução das interfaces. Este log é gravado em uma tabela CTRL_.
- RPL_: identifica o script como sendo uma rotina específica para extrair, tratar e carregar os dados da Zinzane para a TDVTEC. Este script popula as tabelas RPL_, que são as tabelas réplica dos dados da Zinzane. Elas possuem os mesmos nomes de campos e todos os campos são do tipo varchar.
- BAS_: identifica o script como sendo uma rotina específica para extrair, tratar e carregar os dados da Zinzane para a TDVTEC. Este script popula as tabelas BAS_, que é a segunda camada após a réplica. Nas tabelas de base os dados ainda são armazenados na mesma granularidade que no corporativo. As colunas já são renomeadas para o padrão da plataforma de BI, os tipos dos campos já são ajustados para o dado que eles armazenam. Em alguns casos já existe a alteração de conceito nesta tabela.
- AGR_: identifica o script como sendo uma rotina específica para extrair, tratar e carregar os dados da Zinzane para a TDVTEC. Este script popula a tabela AGR_, que são as tabelas agregadas a partir dos dados carregados nas tabelas BAS_. Daqui serão consumidos os dados gerenciais, apesar de não acessarem estas tabelas diretamente, o acesso será sempre encapsulado em views. Antes de criar uma tabela AGR_, é preciso avaliar se é necessário, pois em muitos casos podem-se agregar diretamente nas "DAT_" ou “VIS_”.
- CALC_: identifica o script como sendo uma rotina específica para extrair, tratar e carregar os dados da Zinzane para a TDVTEC. Estes scripts realizam o processamento de cálculo do aging e armazenam o resultado na tabela DAT_. Estas tabelas são visões gerenciais. O acesso externo será sempre através destas tabelas.
- CFG_: são os arquivos de configuração que contem as definições dos parâmetros de conexão com os servidores e databases da Zinzane e da TDVTEC.

6. Principais Funções

- config.py

```
"""
Summary: CLI - database extract utility
Description: Generating a file with all the information to connect with
the databases of the ZINZANE/TDVTEC
"""
```

- controller.py

```
"""
Summary: CLI - database extract utility
Description: Monitor and control all activities to move the data from
ZINZANE server source to TDVTEC server target
"""
```

- log_function.py

```
"""
Summary: CLI - database extract utility
Description: Registering all the activities in the database of the TDVTEC
"""
```

- run_one_etl.py

```
"""
Summary: CLI - database extract utility
Description: Execute just one job/ETL process
"""
```

7. ***Como processar os dados***

Existem duas funções que possibilitam processar os dados: “controller.py” e “run_one_etl.py”. A função “controller.py”, processa os dados de todos os jobs/ETL. A função “run_one_etl.py”, processa os dados de um job/ETL específico.

Para executar todos os processos de ETL manualmente (**controller.py**), é necessário que:

- Acesse o host remote via SSH:

$ ssh tdvtec@75.126.137.89

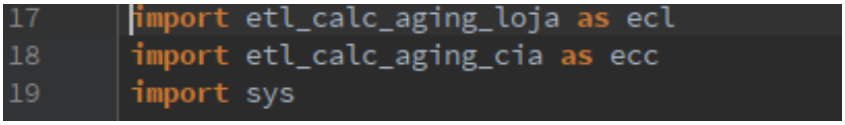
- Selecione o seguinte caminho:

$ cd webapps/cli_zin_py

- Digite o seguinte comando:

$ python controller.py

Para executar um processo específico de ETL manualmente (**run_one_etl.py**) é necessário que algumas alterações sejam feitas, antes de executar este script. Deve-se importar a biblioteca referente ao script que você deseja executar.



Em seguida, é necessário alterar o bloco de código (**__main__**) que faz chamada ao processo ETL que foi importado anteriormente.

```
65
66  # ================================================================================
67   # Part 2 - Executing the job/ETL process
68  # ================================================================================
69  if __name__ == '__main__':
70      # Job/ETL process
71      target = 'DAT_AGING_LOJA'
72      print ">> Executando a Carga " + target + " ..."
73      status, msg = ecl.etl_dat_aging_loja(target)
74      send_message(target, status, msg)
75
76      target = 'DAT_AGING_CIA'
77      print ">> Executando a Carga " + target + " ..."
78      status, msg = ecc.etl_dat_aging_cia(target)
79      send_message(target, status, msg)
80
81      # Send a message to system administrator and close the application
82      print ">> Fim do processamento."
83      sys.exit(0)
84
```

Após efetuar estas mudanças no script, já é possível executá-lo, para isso é necessário que:

- Acesse o host remote via SSH:

$ ssh tdvtec@75.126.137.89

- Selecione o seguinte caminho:

$ cd webapps/cli_zin_py

- Digite o seguinte comando:

$ python run_one_etl.py

8. Agendamento de Tarefas

O agendamento de tarefas é realizado pelo Crontab. Abaixo segue um resumo dos principais comandos deste utilitário.

- crontab -e => Criar/Editar um arquivo crontab.
- crontab -l => Listar o conteúdo de um arquivo crontab.
- crontab -r => Apagar um arquivo crontab.
- crontab -v => Listar o último arquivo editado.

Estrutura do comando:

{minuto} {hora} {dia} {mês} {dia semana} {comando}

O script será executado nos dias 15 e 30 de cada mês às 22 horas.

$ 0 22 15,30 * * /usr/bin/python /webapps/cli_zin_py/controller.py

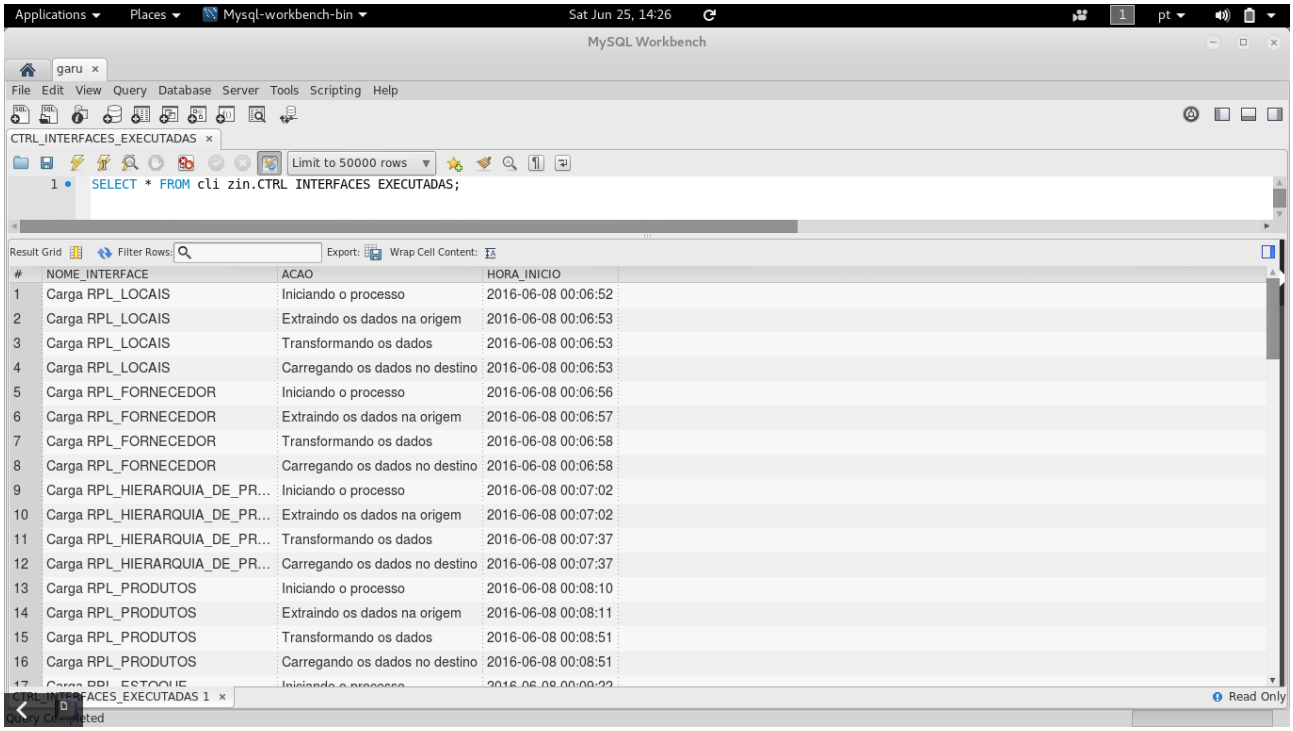
9. Emails de Alertas

Os emails de alerta, fazem parte do processamento de dados e são disparados ao final de cada job/ETL. As funções “controller.py” e “run_one_etl.py”, são responsáveis pelo envio destes emails de alertas. O intuito destes alertas é notificar o andamento do processamento e também avisar se o job/ETL foi concluído com sucesso ou apresentou algum problema.

Em cada uma destas funções existe uma rotina chamada “send_message”.

10. Controle das Interfaces

Em cada processamento de job/ETL, as atividades executadas são gravadas em uma tabela no banco de dados, chamada de CTRL_INTERFACES_EXECUTADAS. A rotina “grava_log_interface”, é responsável pelo registro destas atividades e está localizada dentro da função “**log_function.py**”.



11. Artefatos

Os principais artefatos que foram criados e estão sendo entregues juntamente com este documento, são: os scripts etl em python (.py), scripts de banco de dados (sql) e os arquivos de configuração (.etl).

Scripts de banco de dados (.sql):

- create-tables-mysql-rpl => cria as tabelas da stage area no banco de dados da TDVTEC;
- create-table-interfaces-executadas => cria a tabela de controle das interfaces executadas na stage area da TDVTEC;
- create-tables-mysql-bas => cria as tabelas do data mart no banco de dados da TDVTEC;
- create-table-mysql-agr => cria a tabela de agregação no data mart da TDVTEC;
- create-table-mysql-dat => cria a tabela de visão gerencial no data mart da TDVTEC;
- insert-table-agr =>insere dados consolidados na tabela de agregação da TDVTEC;
- del-data-rpl-bas => apaga os dados das tabelas na stage area e no data mart da TDVTEC;
- check-data-rpl-bas => verifica os dados das tabelas na stage area e no data mart da TDVTEC;
- drop-tables-mysql-rpl => apaga as tabelas na stage area da TDVTEC;
- drop-tables-mysql-bas => apaga as tabelas no data mart da TDVTEC.

Scripts ETL em python:

- config_file => Gera um arquivo com todos os parâmetros de conexão com o banco de dados da ZINZANE/TDVTEC;
- controller => Monitora e controla todas as atividades para mover os dados da ZINZANE para a TDVTEC;
- etl_agr_movint => Agrega os dados de movimentações;
- etl_bas_ajuinv => Move os dados da stage area para o data mart;
- etl_bas_devcli => Move os dados da stage area para o data mart;
- etl_bas_estoqu => Move os dados da stage area para o data mart;
- etl_bas_fornec => Move os dados da stage area para o data mart;
- etl_bas_hiepro => Move os dados da stage area para o data mart;
- etl_bas_locais => Move os dados da stage area para o data mart;
- etl_bas_movint => Move os dados da stage area para o data mart;
- etl_bas_produt => Move os dados da stage area para o data mart;
- etl_bas_recebi => Move os dados da stage area para o data mart;
- etl_calc_aging_cia => Realiza o cálculo do Aging por Cia;
- etl_calc_aging_loja => Realiza o cálculo do Aging por Loja;
- etl_rpl_ajuinv => Move os dados da ZINZANE para a stage area da TDVTEC;
- etl_rpl_devcli => Move os dados da ZINZANE para a stage area da TDVTEC;
- etl_rpl_estoqu => Move os dados da ZINZANE para a stage area da TDVTEC;
- etl_rpl_fornec => Move os dados da ZINZANE para a stage area da TDVTEC;
- etl_rpl_hiepro => Move os dados da ZINZANE para a stage area da TDVTEC;
- etl_rpl_locais =>Move os dados da ZINZANE para a stage area da TDVTEC;
- etl_rpl_movint => Move os dados da ZINZANE para a stage area da TDVTEC;
- etl_rpl_produt => Move os dados da ZINZANE para a stage area da TDVTEC;

- etl_rpl_recebi => Move os dados da ZINZANE para a stage area da TDVTEC;
- log_function => Registra todas as atividades no database da TDVTEC;
- run_one_etl => Executa somente um job/ETL específico.

Arquivos de configuração:

- cfg_src.etl => possui os parâmetros de configuração para conexão com o servidor e banco de dados da Zinzane.
- cfg_tgt.etl => possui os parâmetros de configuração para conexão com o servidor e banco de dados da TDVTEC.