Relatório de Modelagem do Serviço de Transferência de Dados

Produto 6.3.1–1

Versão 2.1

Outubro 2018

Histórico da Revisão

Data	Versão	Histórico	Autor
31/10/2018	*1.0*	*Elaboração do relatório*	*INECO*
07/11/2018	*1.1*	*Revisão do relatório*	*Ábdon*
14/11/2018	*2.0*	*Incorporação das correções*	*INECO*
27/11/2018	*2.1*	*Atualização das correções*	*INECO*

Sumário

Índice de figuras

1 APRESENTAÇÃO

O presente relatório é um dos entregáveis previstos na Carta de Acordo celebrada entre o Projeto do PNUD BRA 13/013 e a empresa pública espanhola de Ingeniería y Economia del Transporte – INECO para atendimento ao Observatório Nacional de Transporte e Logística – ONTL.

Dados da contratação	
Item	**Descrição**
Instrumento de contratação	Carta de Acordo PNUD BRA 13/013
Agência Implementadora	Ingeniería y Economía del Transporte – INECO
Assinatura	Setembro/2017
Início do Projeto	02/10/2017
Gerente do Projeto INECO	Enrique Monfort
Gerente do Projeto EPL	Jony Marcos do Valle Lopes
Coordenadora responsável EPL	Lilian Campos Soares
Dados do relatório	
Fase	Fase 6.3 – Concepção e consultoria técnica na implantação de um serviço de transferências de dados para o banco de dados do Sistema de Informações
Etapa	Etapa 6.3.1 – Concepção dos serviços de transferência de dados
Documento/Entregável	Produto 6.3.1-1 – Relatório de Modelagem do Serviço de Transferência de Dados

2 INTRODUÇÃO

Dando início à **Fase 3** de *concepção e consultoria técnica na implantação de um serviço de transferências de dados para o banco de dados do sistema de informações*, a INECO por meio desse documento pretende descrever os recursos necessários para o recebimento dos dados pela EPL e a respectiva integração dos mesmos na base de dados que atenderá o Observatório Nacional de Transporte e Logística da EPL no Brasil.

A construção ou o desenvolvimento propriamente dito, assim como a configuração dos mesmos nos recursos de hardware e software aplicáveis, incluindo os serviços de comunicação necessários e os programas implementados para este fim, serão realizados pela EPL.

Lembrando que os dados exibidos pelo ONTL não serão gerados pela EPL e sim obtidos de diversas fontes provedoras de dados já descritas nos documentos da Fase 1 (*Concepção do Sistema de Informações*).

A partir desses dados provenientes das fontes, é necessário um trabalho de extração, transformação e carga em um formato que atenda os componentes do sistema de informações do Observatório.

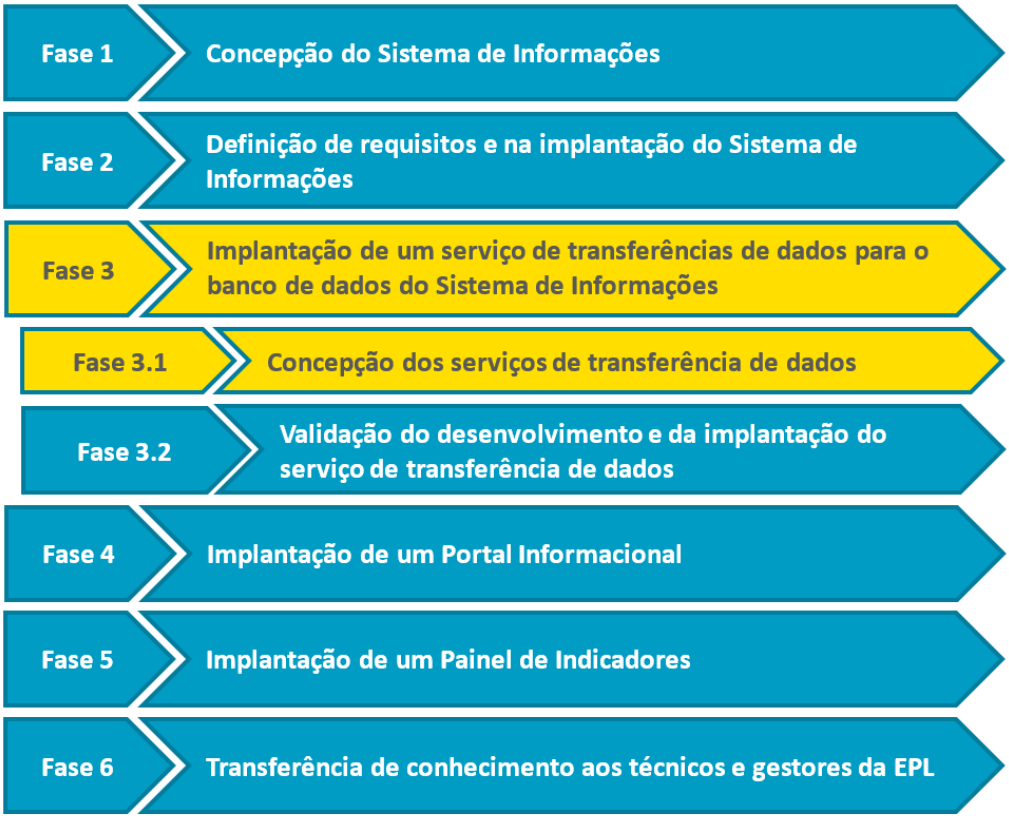
Por esse motivo, aqui descreveremos como os dados da fonte serão disponibilizados para o processo de transformações e cargas dos mesmos. Também será especificado o Framework que suportará os procedimentos de Extração, Transformação e Carga desses dados, bem como a evolução da base que armazenará a informação de cada processo de tratamento desses.

Será também descrita uma ampliação da base de dados utilizada nesses procedimentos que otimizará o trabalho com o QLIK (ferramenta de análise de dados escolhida pela CONIL/GEPDL) e possibilitará manter o rastreamento e auditoria dos processos de ETL.

3 ALCANCE E OBJETIVOS

Abaixo está representado um breve esquema das fases deste projeto destacando em amarelo a etapa em que este relatório se encontra:

Figura 1. Fases do Projeto



Fonte: Elaboração própria

Este relatório dá início à **Etapa 3.1 "Concepção dos serviços de transferência de dados"**, dentro da **Fase 3 "Implantação de um serviço de transferência de dados para o banco de dados do sistema de informações"**, e seu objetivo principal é descrever o(s) serviço(s) de transferência de dados que receberá e incorporará os dados na base de dados do ONTL.

4 ABORDAGEM METODOLÓGICA PARA O DESENVOLVIMENTO DO SISTEMA

O sistema que atenderá as necessidades do ONTL será desenvolvido de forma a concentrar a disponibilização das informações relevantes ao panorama do ONTL em um único espaço que disponibilize os artefatos do sistema integrado à base de dados.

Para tanto, os dados serão obtidos de variadas fontes, trabalhados, tratados e transformados para serem armazenados de forma a otimizar seu acesso pelos recursos do Observatório.

Dando início à Fase 3 (*Implantação de um serviço de transferência de dados para o banco de dados do sistema de informações*) este relatório estará organizado de forma a especificar o recebimento, o tratamento e o armazenamento desses dados.

A descrição do Framework ETL definido para o Observatório está baseada na estrutura de pacotes utilizada pelo MS-SSIS (*Microsoft SQL Server Integration Service*), ferramenta escolhida pela CONIL para realizar a integração dos dados das fontes ao repositório do ONTL.

Conforme já mencionado na introdução desse documento, será também abordada a ampliação realizada no modelo de dados para que a base de dados do ONTL atenda de forma eficiente aos recursos utilizados pelo mesmo. A nomenclatura utilizada para os objetos dessa ampliação, assim como os objetos já definidos anteriormente, segue os padrões definidos no “Processo de Desenvolvimento de Software - PDS Padrão de Banco de Dados” fornecido pela EPL.

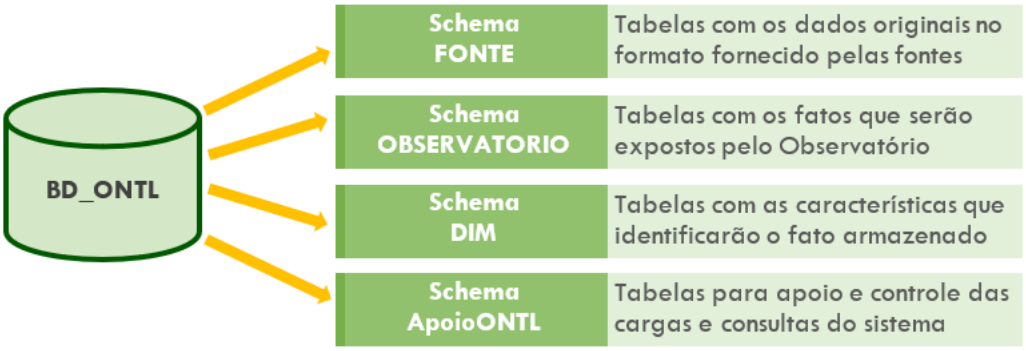
A modelagem de dados foi realizada na via Enterprise Architect, versão 13.0 e os modelos ampliados para os DataSets já analisados foram enviados no documento Produto 6.1.3-a.1 – Relatório dos Modelos de Dados (Atualização 01 - Ampliação do modelo).

5 AMPLIAÇÃO DO MODELO DA BASE DE DADOS DO OBSERVATÓRIO

Em meio ao estudo de Concepção do Serviço de Transferência e Transformação de Dados do ONTL fez-se necessário ampliar o modelo da Base de Dados que atenderá ao sistema do Observatório de modo a manter a integridade dos dados ao longo dos processos de ETL. A rastreabilidade e a auditoria das execuções desses processos e a tornará mais eficiente seu acesso por ferramentas de análise de BI (como o QLIK, escolhido pela CONIL).

Até o momento, a base de dados estava definida com o modelo a seguir:

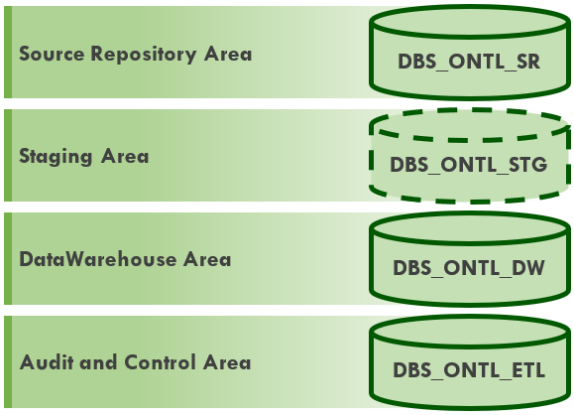
Figura 2. Modelo de Base de Dados anterior



Fonte: Elaboração própria

Com a nova proposta para o modelo, serão criados bancos de dados (e seus respectivos schemas) de forma a manter a segurança, integridade e manutenção mais independente de acordo com o conteúdo que cada base armazenará e com o trabalho que será executado pelos processos de ETL:

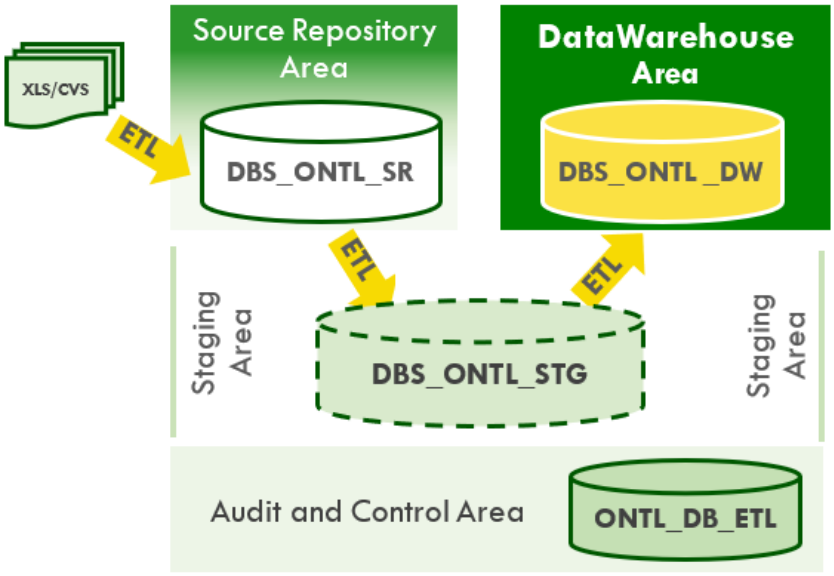
Figura 3. Novo modelo de Base de Dados sugerido



Fonte: Elaboração própria

A figura a seguir ilustra como será a integração dos processos ETLs (que serão descritos na próxima seção) com os bancos de dados sugeridos nesta ampliação:

Figura 4. Utilização dos Bancos de Dados sugeridos pelos processos de ETL



Fonte: Elaboração própria

Cada Banco de Dados terá sua organização de schemas para melhor controle e organização dos trabalhos dos processos de ETL:

- **DBS_ONTL_SR:** utilizado pela *Source Repository Area* - Área de "origem" dos dados que serão alimentados a partir do conteúdo dos arquivos fornecidos.
  → **Schema APOIO:** origem das tabelas de apoio do sistema.
  → **Schema COMUM:** tabelas cujos dados darão origem às dimensões.
  → **Schema <FONTE>:** haverá um schema para cada fonte onde estarão reunidas as tabelas contendo os dados originais de cada arquivo no formato fornecido pelas fontes.
- **DBS_ONTL_DW:** utilizado pela *DataWarehouse Area* - Área onde os dados serão lidos pelas ferramentas de exploração de dados.
  → **Schema APOIO:** tabelas para apoio e controle das cargas e consultas do sistema.
  → **Schema DIM:** tabelas de dimensões que caracterizarão os dados do Observatório.
  → **Schema OBSERVATORIO:** tabelas com os fatos que serão expostos pelo Observatório.

- **DBS_ONTL_STG:** utilizado pela *Staging Area* - Essa área representa a camada de transição dos dados entre sua área de origem (SR) e a área do DataWarehouse (DW).
  → **Schema APOIO:** tabelas auxiliares para o trabalho dos processos de ETL ao alimentar as tabelas de apoio do DataWarehouse.
  → **Schema DIM:** tabelas auxiliares para o trabalho dos processos de ETL ao alimentar as tabelas de fatos do DataWarehouse.
  → **Schema OBSERVATORIO:** tabelas auxiliares para o trabalho dos processos de ETL ao alimentar as tabelas de dimensões do DataWarehouse.
- **DBS_ONTL_ETL:** utilizado pela *Audit and Control Area* durante a execução dos processos ETL. Não contem dados do Observatório, apenas dados internos de controle e auditoria dos processos ETL.

Além dessa ampliação e reorganização de objetos dos bancos de dados, foram incluídos alguns campos nas tabelas para auditoria e controle do trabalho dos processos de ETL:

→ **vbn_etlhashkey:** chave interna que identifica cada entrada de dado utilizada pelos processos ETL.
→ **int_etlcreatedby:** identifica o processo que inseriu o registro pela primeira vez.
→ **dtt_etlcreateddate:** armazena data e hora de inclusão do registro pela primeira vez.
→ **int_etlmodifiedby:** identifica o último processo que alterou o registro.
→ **dtt_etlmodifieddate:** armazena a última data e hora em que modificou o registro.

Os bancos de dados criados viabilizam o trabalho dos processos de ETL sob o framework definido. Além disso, possibilitam melhor controle e auditoria desse trabalho e seus resultados e, a partir disso, realizar melhorias contínuas nos processos de ETL.

A ampliação realizada torna mais prático e eficiente o trabalho de desenvolvimento dos painéis analíticos com o QLIK ou qualquer outra ferramenta de exploração de dados para análise de BI disponível atualmente no mercado.

Sugerimos que, para cada base de dados criada nos diferentes ambientes, haja dois arquivos de sistema operacional: um arquivo de dados e um arquivo de registro de transações. Os arquivos de dados devem conter os dados e outros objetos como tabelas, índices, *views*, *StoredProcedures*. Os arquivos de registros devem conter a informação necessária para recuperar todas as transações da base de dados. Os arquivos de dados podem ser agrupados para melhor administração e alocação.

Pontos a considerar ao definir arquivos operacionais e grupos de arquivos:

- A maior parte das bases de dados funcionam corretamente com um arquivo de dados e um arquivo de registro de transações.
- Se a ideia for utilizar vários arquivos de dados, melhor criar um segundo grupo de arquivos para o adicional e convertê-lo em grupo de arquivos pré-determinado. Dessa forma, o arquivo principal conterá apenas objetos e tabelas do sistema.
- Para maximizar o rendimento, recomenda-se criar arquivos ou grupos em tantos discos disponíveis quantos seja possível e distribuir em grupos de arquivos diferentes os objetos que competem de forma intensa por espaço.
- Utilize grupos de arquivos para permitir a colocação de objetos em determinados discos físicos.
- Disponha em grupos de arquivos distintos as diferentes tabelas que se utilizem das mesmas consultas de combinação. Desse modo, o rendimento melhorará devido a busca de dados combinados que realizam as operações de entrada e saída (E/S) paralelas nos discos.
- Distribua em diferentes grupos de arquivos as tabelas de acesso frequente e os índices no cluster que pertençam a essas tabelas. Desse modo, o rendimento aumentará devido às operações de E/S em paralelo que são realizadas se os arquivos se encontram em discos físicos distintos.
- É recomendável não colocar os arquivos de registros de transação no mesmo disco físico dos demais arquivos e grupos de arquivos.

Outro ponto a considerar são as cópias de segurança das plataformas, já que realizar bons backups é uma tarefa complexa. Por isso, no período onde a carga de dados é mais frequente, recomendamos cópias de segurança diárias de todos os grupos de arquivos que formam a área de *Source Repository* (dbs_ontl_sr) e do *DataWarehouse* (dbs_ontl_dw). Não é necessário realizar cópias de segurança da área de *Staging*.

Como ponto evolutivo para o futuro, foi implementado nas base de dados a funcionalidade de esquemas que oferece muitas vantagens em um ambiente SQL Server já que os esquemas são uma forma de classificar os objetos da base de dados. Tudo isso para ter um melhor controle lógico e uma melhor organização dos objetos e não se perder entre eles se tiver uma grande quantidade deles. Combinado com o nível apropriado de permissão por usuário (quando se aplique), os esquemas podem ser uma ferramenta de proteção de dados muito efetiva, possibilitando alta disponibilidade do sistema. Isso pode ser extrapolado para o *DataWarehouse* e, quando for necessário, implementar segurança a nível de exploração de dados privados que não publicados diretamente no Observatório, mas analisados internamente para embasar alguma ação de apoio à tomada de decisões.

6 DISPONIBILIZAÇÃO DOS DADOS UTILIZADOS PARA O ONTL

Os responsáveis pela seleção dos dados primários que alimentarão o ONTL, após a busca por tais dados, geram arquivos com o formato que será reconhecido pelos procedimentos de carga definidos para alimentar a base de dados do Observatório.

Para armazenar e organizar esses arquivos, sugerimos definir uma área na rede interna da EPL onde esses arquivos sejam disponibilizados. Ali seriam armazenados os arquivos sempre que novos dados estiverem disponíveis para a carga na base de dados do Observatório Nacional de Transporte e Logística do Brasil. Essa seria a única fonte de entrada da base de dados do Observatório.

Seguindo essa sugestão, os arquivos estariam armazenados em um sistema de arquivos próprio ao qual apenas o Framework ETL e usuários autorizados pela CONIL teriam acesso. Os usuários que a CONIL defina como responsáveis pela disponibilização de tais arquivos e os processos de carga terão acessos diferenciados a essa área.

Os arquivos seriam disponibilizados nessa área através de um protocolo de transferência de arquivos preferencialmente criptografado (SFTP) por acesso autenticado via controle de usuário e senha definidos pela EPL.

No momento em que os parceiros e colaboradores estiverem prontos para fornecer os arquivos no formato reconhecido pelos processos de carga, poderiam receber um usuário de SFTP com acesso a suas respectivas pastas para disponibilizar tais arquivos de dados.

Como os arquivos aí disponibilizados serão a fonte dos dados que carregados na Base de Dados do ONTL através dos processos de ETL a serem desenvolvidos, esses processos também precisam receber acesso a essas pastas.

Para ser utilizada de acordo o framework que foi definido e será descrito mais adiante, essa área deverá ter a seguinte estrutura:

- ONTL File Repository
  - COMUM
    aqui estarão os arquivos origem das dimensões comuns a vários DataSets.
    → os processos de carga precisam receber acesso de leitura nessa pasta.
  - <FONTE> (*deverá ser disponibilizada uma pasta para cada fonte de dados)*
    aqui estarão os arquivos originais das fontes que darão origem aos fatos do Observatório.
    → os processos de carga precisam receber acesso de leitura nessa pasta.
    - ERROR
      utilizada pelos processos ETL em caso de erro no processamento do arquivo
      → os processos de carga precisam receber acesso de leitura e escrita nessa pasta.
    - PROCESS
      utilizada pelos processos ETL em caso de sucesso no processamento do arquivo
      → os processos de carga precisam receber acesso de leitura e escrita nessa pasta.

7 FRAMEWORK PARA EXTRAÇÃO, TRANSFORMAÇÃO E CARGA DE DADOS

Após a obtenção dos dados fornecidos pelas diversas fontes, faz-se necessário o tratamento dos mesmos para que possam ser de maior proveito para o trabalho do Observatório. Para isso, serão desenvolvidos processos automáticos de extração, transformação e carga desses dados.

Com o objetivo de simplificar e padronizar o desenvolvimento dos processos de ETL (Extração, Transformação e Carga) e manter o rastreamento dos mesmos, foi concebido um Framework ETL para o ONTL. Esse Framework oferece uma estrutura pré-definida que permitirá realizar a implementação dos processos de carga dos DataSets de maneira padronizada, diminuindo assim a "curva de aprendizado" que envolve implementar todas as transformações necessárias. O modelo estrutural definido proporciona um pacote pai e grupos de pacotes filhos com tarefas pré-determinadas para, dentre outras coisas, registrar a execução, seus erros e informações de auditoria de cada registro manipulado.

O Framework proporcionará a base para a criação dos pacotes do MS-SSIS (*Microsoft SQL Server Integration Service*) permitindo integrar os dados das diferentes fontes ao repositório do ONTL.

Como ilustrado anteriormente na Figura 4, a Área de Repositório de Origem (*Source Repository Area*) receberá os dados originais das fontes e atuará como única origem para a carga dos dados no DataWarehouse que será explorado pelo Observatório. O Repositório de Origem (*Source Repository – SR*) conterá os dados no mesmo formato disponibilizado pela fonte, sem nenhuma transformação.

Seguem alguns benefícios do Framework ETL idealizado para o ONTL:

- **Foco no desenvolvimento coerente:** proporciona modelos para a criação de pacotes pais e filhos, mantendo a coerência estrutural ao longo de todo o processo de desenvolvimento da ETL.
- **Registro de operação incorporado:** o próprio processo registra dados sobre sua execução, por exemplo: quando começou a ETL, quando terminou, qual foi o resultado da execução, quantos registros foram manipulados. Dessa forma, não é necessário escrever código separado com essa finalidade.
- **Auditoria integrada:** o próprio processo audita cada registro manipulado durante sua execução (ou execução em lote) indicando qual registro se atualizou ou foi adicionado.
- **Desenvolvimento mais rápido (menor curva de aprendizagem):** como proporciona um modelo para a criação de pacotes pais e filhos, sua implementação é muito rápida uma vez

que basta copiar o modelo existente e realizar os ajustes necessários para personalizar cada operação.

- **Capacidade de carga completa e de carga delta (incremental):** a estrutura definida permite executar um mesmo pacote em modo carga completa ou em modo carga delta. Como predeterminado, a primeira execução realizará a carga completa e as seguintes realizarão cargas incrementais. Caso seja necessário, o pacote pai sempre pode ser configurado para fazer uma carga completa de determinada origem para as tabelas do SR. Quando for assim, removerá previamente os dados já carregados e carregará um novo conjunto de dados da origem especificada.
- **Gerenciamento dinâmico de índices:** em certos casos, quando se tem um volume grande de dados ou as alterações são numerosas, primeiro se deve liberar os índices na tabela de destino, carregar os dados e voltar a criar os índices. Com o modelo definido, apenas será necessário especificar quais índices devem ser liberados antes da carga dos dados e quais índices devem ser recriados depois da carga dos dados, o processo se encarregará de todo o resto.
- **Reinício a partir do POF (Ponto de Falha):** uma falha durante o processo ETL pode estar relacionada com uma fonte inativa, problemas de rede e etc. A estrutura do framework definido permite gerenciar corretamente a falha de forma que, quando reiniciado, o processo ETL comece a partir do ponto em que falhou. Por exemplo, em um caso em que haja 10 pacotes filhos para carregar dados em 10 tabelas diferentes, a ETL foi iniciada e completou a carga de dados em 8 tabelas e não pode completar a carga nas demais, quando for reiniciado o processo começará a carga apenas das tabelas que faltaram. Esse é o comportamento pré-determinado, mas pode ser alterado e configurado a qualquer momento para iniciar do zero.

De forma geral, existem 2 tipos de pacotes definidos nesse framework:

➢ **Pacote Pai:** para cada fonte de dados deverá ser criado um pacote pai independente que atuará como controlador para a integração total dos dados dessa fonte. Esse pacote registrará a informação de início de lote na tabela dbs_ontl_etl.ETL.BatchHeader, chamará cada um dos pacotes filhos e transmitirá a esses pacotes as configurações necessárias. De acordo com o resultado da execução, também registrará a informação de sucesso ou falha do processo na base dbs_ontl_etl.

Recomendação: para manter íntegra sua capacidade de administração, cada pacote pai deve chamar um máximo de 40 pacotes filhos. Se for necessário mais de 40 filhos, o pacote pai deve ser dividido em múltiplos para garantir de cada um se encarregue apenas de 30 a 40 pacotes filhos.

- **Pacote Filho:** para cada tabela ou cada arquivo deverá ser criado um pacote filho. Cada um desses pacotes receberá informações de configuração de seu pacote pai e registrará a hora de início de sua execução, o tempo de finalização do pacote filho, o resultado da execução (sucesso ou falha), informação de falha se houve alguma, quantidade de linhas criadas, alteradas, etc. na tabela dbs_ontl_etl.ETL.BatchDetail. Cada pacote filho terá duas rotas possíveis de fluxo de dados:

  - **Carga completa:** de forma predeterminada, é como se realizará a primeira execução dos pacotes. Essa carga extrai um conjunto completo de dados da origem e alimenta a tabela destino. Caso seja necessário, é possível configurar o pacote para que realize a carga completa em outro momento, porém, nesse caso, os dados carregados anteriormente serão totalmente descartados e nova carga completa será realizada.

  - **Carga incremental (delta):** é a forma predeterminada para as execuções posteriores dos pacotes. A carga incremental dos dados será realizada segundos essas 2 lógicas:

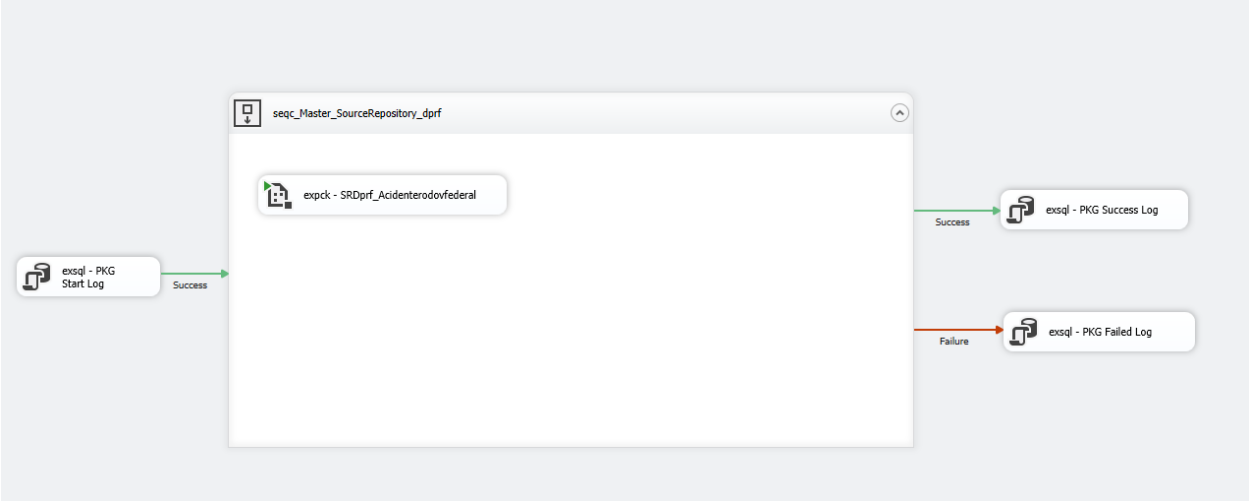
    → **Tabela de origem <u>com</u> marca de tempo:** com a ajuda de uma marca de tempo (data e hora de alteração) na fonte, é identificado o conjunto de registros modificados, extrai da tabela fonte apenas o "novo" e alimenta a tabela de destino.

    → **Tabela de origem <u>sem</u> marca de tempo:** quando as linhas *delta* (alteradas ou incluídas desde a última carga) não podem ser identificadas (por não terem marca de tempo), não é possível diferencia na fonte o conjuntos de registros modificados, portanto, a cada execução deverá: ingressar um conjunto completo de dados da tabela fonte em uma tabela intermediária; gerar uma *hashkey* para cada linha (com o valor concatenado de todas as colunas); realizar uma comparação entre as linhas da tabela intermediária e as linhas da tabela destino para identificar o conjunto de registros modificados e incluídos; e, finalmente alimentar a tabela destino com as diferenças.

7.1 Estrutura do Pacote Pai

Segue abaixo a estrutura de um Pacote Pai definido para nesse framework e a descrição de seus componentes:

Figura 5. Exemplo de estrutura de um Pacote Pai

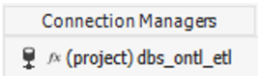


Fonte: Elaboração própria

- **exsql - PKG Start Log** → Esta tarefa chama a StoredProcedure *uspLogPackageStart* do banco dbs_ontl_etl para registrar o início do pacote pai ou do lote na tabela ETL.BatchHeader e devolver o ID do processo (BatchID). Também busca na tabela ETL.SourceSystem a informação de configuração que o pacote pai utilizará durante sua execução, como cadeias de conexão, data da última atualização e etc.
- **seqc_Master_SourceRepository_<fonte name>** → Este contêiner de sequência possui tarefas de execução de pacote ("*Execute Package Tasks*") para chamar cada um dos pacotes filhos. As informações de configuração de chamada passam do pacote pai para seus pacotes filhos.
- **exsql - PKG Failed Log** → Esta tarefa chama a StoredProcedure *uspLogPackageCompletion* de *dbs_ontl_etl* para registrar informações de falha na execução do pacote pai na tabela ETL.BatchHeader.
- **exsql - PKG Success Log** → Esta tarefa chama a StoredProcedure *uspLogPackageCompletion* de dbs_ontl_etl para registrar informações na tabela ETL.BatchHeader em caso de execução realizada com sucesso.

O pacote pai contém um gerenciador de conexões para conectar-se com o banco de dados dbs_ontl_etl.

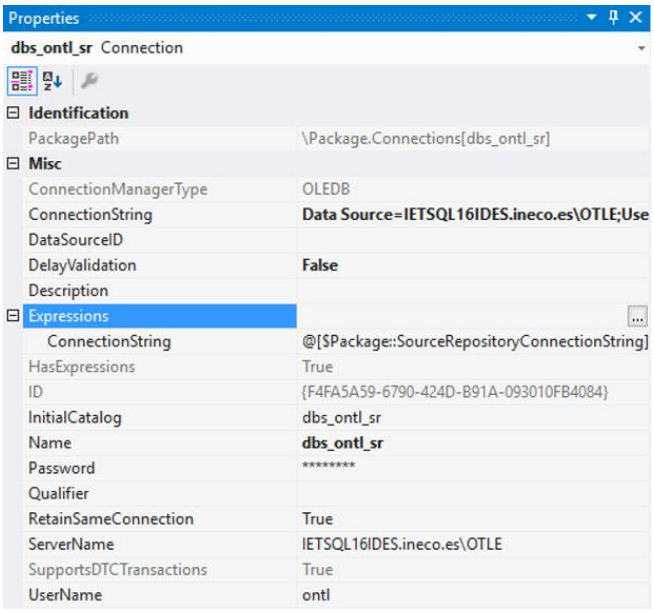
Figura 6. Gerenciador de conexões do Pacote Pai



Fonte: Elaboração própria

Esse gerenciador de conexões obtém os dados para conexão do parâmetro de nível de projeto ETLDatabaseConnectionString. Quando os pacotes forem implementados, esse parâmetro indicará a informação da cadeia de conexão para conectar-se ao dbs_ontl_etl e eventualmente usará o pacote pai para se conectar ao banco e obter informação adicional de configuração.

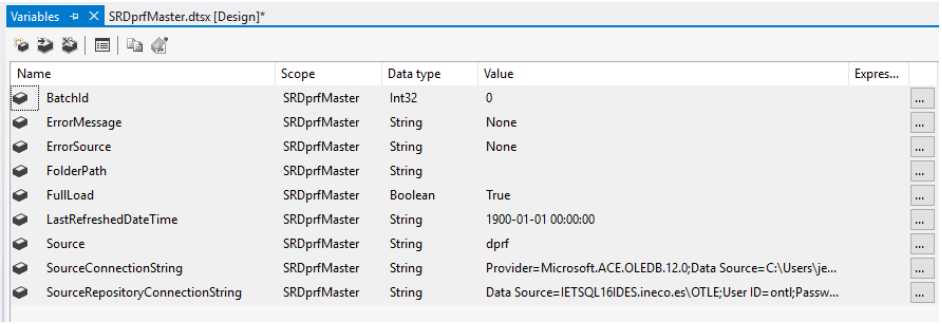
Figura 7. Configuração de conexões do Pacote Pai



Fonte: Elaboração própria

O pacote pai possui muitas variáveis necessárias a seu funcionamento:

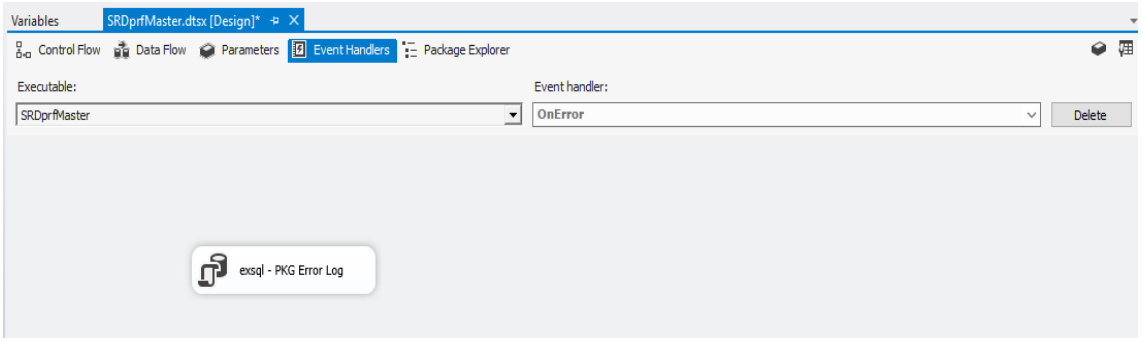
Figura 8. Variáveis de configuração do Pacote Pai



Fonte: Elaboração própria

Uma vez que o alcance das variáveis (System::ErrorSource e System::ErrorDescription) está no evento, o pacote pai contém um controlador no evento OnError a tarefa **exsql - PKG Error Log** que copia os valores das variáveis do pacote pai para que possam ser utilizadas fora daí.

Figura 9. Controlador de Eventos do Pacote Pai

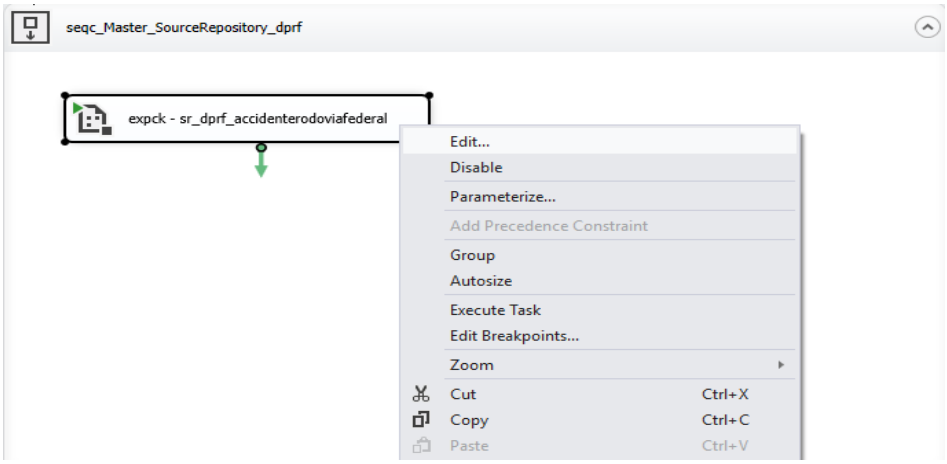


Fonte: Elaboração própria

O pacote pai funciona como um coordenador, executando os pacotes filhos (um pacote filho para carga de dados de cada fonte), enviando para esses as informações de configuração, evitando assim que tais pacotes necessitem voltar a ler essas informações ao serem executados.

Para especificar as configurações que devem ser passadas para o pacote filho basta clicar com o botão direito na tarefa de execução de pacote e escolher “Editar...” como mostrado a seguir.

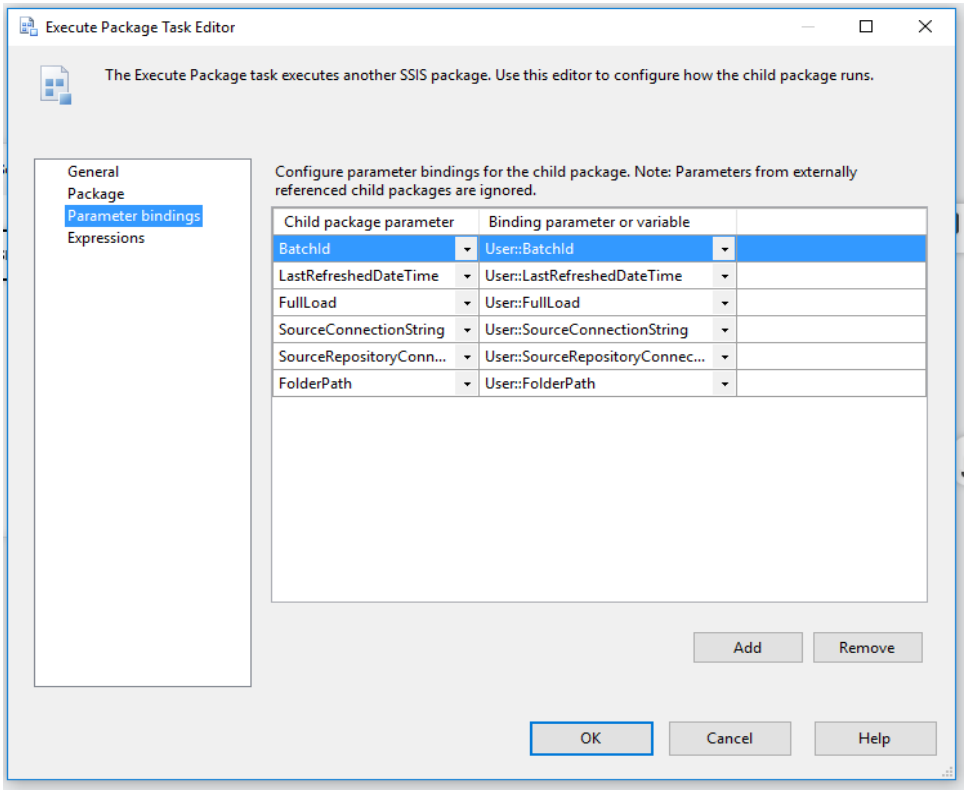
Figura 10. Passando configurações ao Pacote Filho



Fonte: Elaboração própria

No editor de tarefas do pacote (*Execute Package Task Editor*), clicando nos parâmetros da esquerda é possível associar os valores das variáveis do pacote pai aos parâmetros do pacote filho.

Figura 11. Definindo valores de parâmetros do Pacote Filho

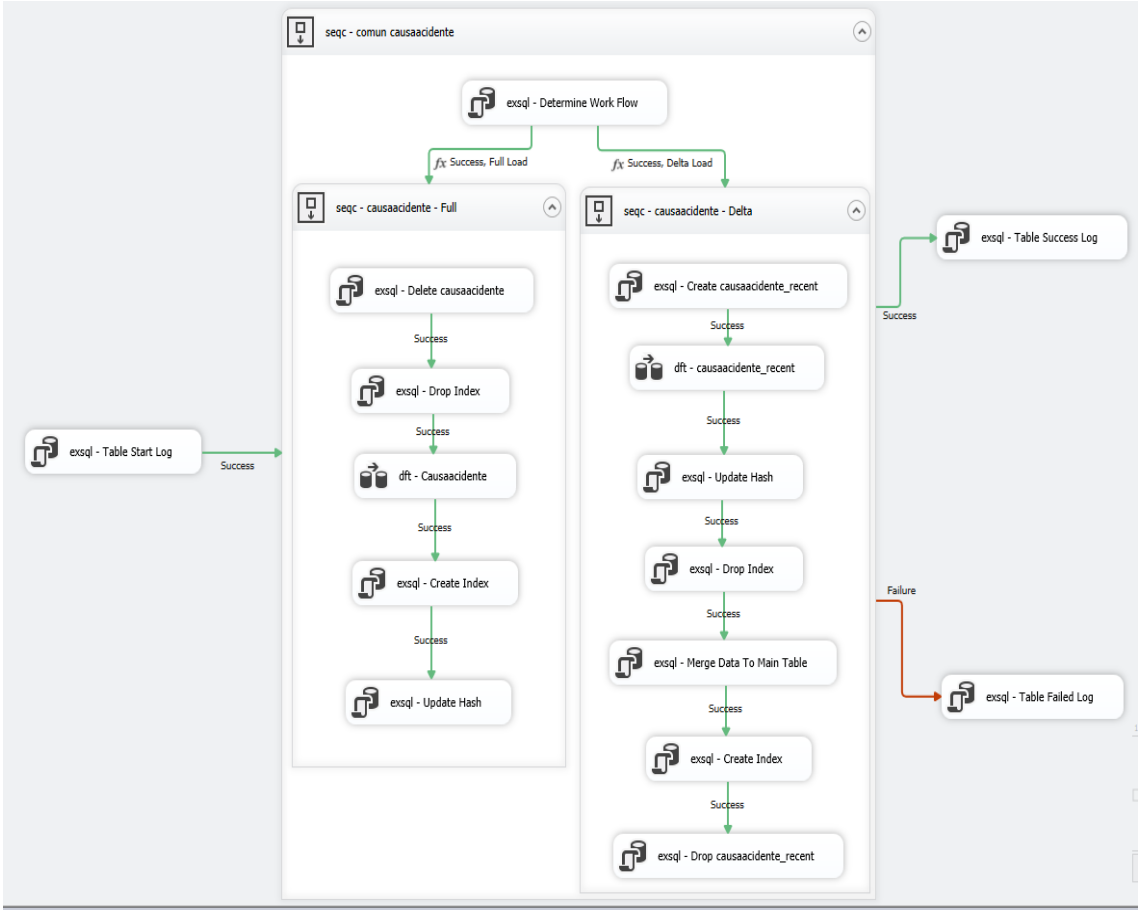


Fonte: Elaboração própria

7.2 Estrutura do Pacote Filho

Os pacotes pais de cada fonte de dados controlarão e executarão pacotes filhos para cada tabela ou arquivo carregado. Esses pacotes filhos terão a estrutura a seguir:

Figura 12. Estrutura de um Pacote Filho



Fonte: Elaboração própria

- **exsql – Table Start Log** → Esta tarefa de execução SQL chama a StoredProcedure *uspLogTableStart* de dbs_ontl_etl para registrar o início do processamento na tabela ETL.BatchDetail.

- **exsql – Determine Work Flow** → Esta tarefa de execução SQL determina o fluxo do pacote em função da informação de configuração passada pelo pacote pai.

- **seqc – <destination-table-name> - Full** → Este contêiner de sequência possui todas as tarefas que a serem executadas durante a carga completa:

  → **exsql – Truncate <destination-table-name>**: Esta tarefa de execução SQL trunca a tabela de destino para deixar espaço para a carga dos dados.

  → **exsql – Drop Index:** Essa tarefa chama a StoredProcedure *ETL.uspManageIndex* e só funciona quando existe uma ou mais entradas para a tabela de destino com Action=FullLoadDropIndex. Ela será responsável por liberar os índices da tabela de destino (caso Action=FullLoadDropIndex na tabela ETL.ETLTableIndex) na ordem de prioridade definida. A prioridade é importante para que os índices não agrupados possam liberados antes dos demais. Se não desejar descartar os índices durante o processo de ETL, simplesmente não adicione nenhuma entrada na tabela ETL.ETLTableIndex para a tabela de destino ou atualize o indicador IsActive=Falso para registros com Action=FullLoadDropIndex.

  → **dft – <destination-table-name>:** Esta tarefa de fluxo de dados realiza a carga completa de dados da tabela de origem para a tabela de destino.

  → **exsql – Create Index:** Essa tarefa chamada a StoredProcedure *ETL.uspManageIndex* e só funciona quando existe uma ou mais entradas para que a tabela de destino crie índices com Action=FullLoadCreateIndex. Ela será responsável por criar os índices na tabela de destino caso Action=FullLoadCreateIndex tenha sido definido em ETL.ETLTableIndex e na ordem de prioridade definida. A prioridade aqui também é importante para que os índices agrupados sejam criados antes dos demais. Se não desejar criar os índices durante o processo de ETL, simplesmente não adicione nenhuma entrada na tabela ETL.ETLTableIndex para a tabela de destino ou atualize o indicador IsActive=False para registros com Action=FullLoadCreateIndex.

  → **exsql – Update Hash:** Esta tarefa de execução SQL gera una *hashkey* para cada linha da tabela de destino com a concatenação dos valores de todos os campos e a armazena no campo vbn_etlhashkey.

- **seqc – <destination-table-name> - Delta** → Este contêiner de sequência possui todas as tarefas a serem executadas durante a carga incremental/delta:
  → **exsql – Create <destination-table-name>_recent:** Esta tarefa de execução SQL cria uma tabela intermediária que servirá de apoio para realizar a inclusão dos dados.

  → **dft – <destination-table-name>_recent:** Esta tarefa de fluxo de dados realiza a carga incremental dos dados. Caso a marcação de tempo esteja disponível na tabela de origem, a carga para a tabela intermediária se baseará na última data e hora de execução com sucesso do pacote pai. Ou, caso a marcação de tempo não esteja disponível na tabela de origem, então todo o conjunto de dados da fonte é carregado na tabela intermediária.

  → **exsql – Update Hash:** Esta tarefa de execução SQL gera una *hashkey* para cada linha da tabela de destino com a concatenação dos valores de todos os campos e a armazena no campo vbn_etlhashkey da tabela intermediária.

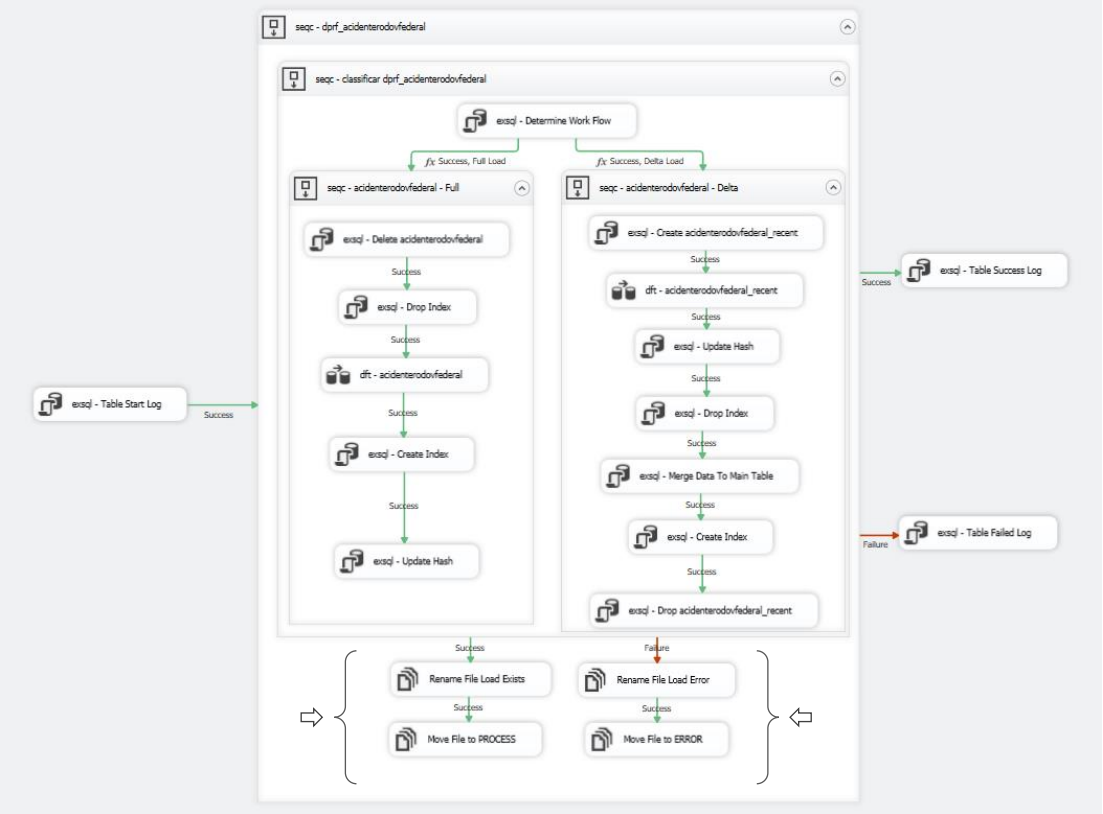
  → **exsql – Drop Index:** Essa tarefa chama a *StoredProcedure* *ETL.uspManageIndex* e só funciona quando existe uma ou mais entradas para a tabela de destino com Action=DeltaLoadDropIndex. Ela será responsável por descartar os índices da tabela de destino (caso Action=DeltaLoadDropIndex na tabela ETL.ETLTableIndex) e na ordem de prioridade definida. A prioridade é importante para que os índices não agrupados sejam descartados antes dos demais. Se não desejar descartar os índices durante o processo de ETL, simplesmente não adicione nenhuma entrada na tabela ETL.ETLTableIndex para a tabela de destino ou atualize o indicador IsActive=Falso para registros com Action=DeltaLoadDropIndex.

  → **exsql – Merge Data To Main Table:** Esta tarefa de execução SQL chama a instrução MERGE para fundir os dados da tabela intermediária na tabela de destino em função de combinação primária ou única:
    - Se a chave primária não existe no destino, insere a linha.
    - Se a chave primária existe no destino, mas a *hashkey* é diferente, atualiza a linha
    - Se a chave primária existe no destino e a *hashkey* também coincide, não faz nada.

  → **exsql – Create Index:** Essa tarefa chama a *StoredProcedure* *ETL.uspManageIndex* e só funciona quando existe uma ou mais entradas para a tabela de destino com Action=DeltaLoadCreateIndex. Ela será responsável por descartar os índices da tabela de destino (caso Action=DeltaLoadCreateIndex na tabela ETL.ETLTableIndex) na ordem de prioridade definida. A prioridade é importante para que os índices agrupados sejam criados antes dos demais. Se não desejar descartar os índices durante o processo de ETL, simplesmente não adicione nenhuma entrada na tabela ETL.ETLTableIndex para a tabela de destino ou atualize o indicador IsActive=Falso para registros com Action=DeltaLoadCreateIndex.

→ **exsql - Drop <destination-table-name>_recent:** Esta tarefa de execução SQL trunca a tabela intermediária para deixar espaço para a carga dos dados.

➢ Para as tabelas de fatos é adicionado um novo contêiner (seqc – classificar <nome-fonte>_<nome-tabela-destino>) com todas as tarefas de execução mencionadas anteriormente e, dependendo da saída (com sucesso ou erro), são incluídas tarefas de manipulação de arquivos:



⇨ **Rename File Load Exists:** Caso a execução do processo de carga do arquivo em questão termine com sucesso, esta tarefa cria uma cópia do arquivo origem com o nome alterado para: <nome-arquivo-fonte>_<data-da-carga>.<formato-arquivo-fonte>.

⇨ **Rename File Load Error:** Caso ocorra algum erro na execução do processamento do arquivo em questão, esta tarefa cria uma cópia do arquivo origem com o nome alterado para: <nome-arquivo-fonte>_<data-da-carga>.<formato-arquivo-fonte>.

⇨ **Move File to PROCESS:** Em caso de êxito na execução, esta tarefa move o arquivo com nome alterado para a pasta PROCESS.

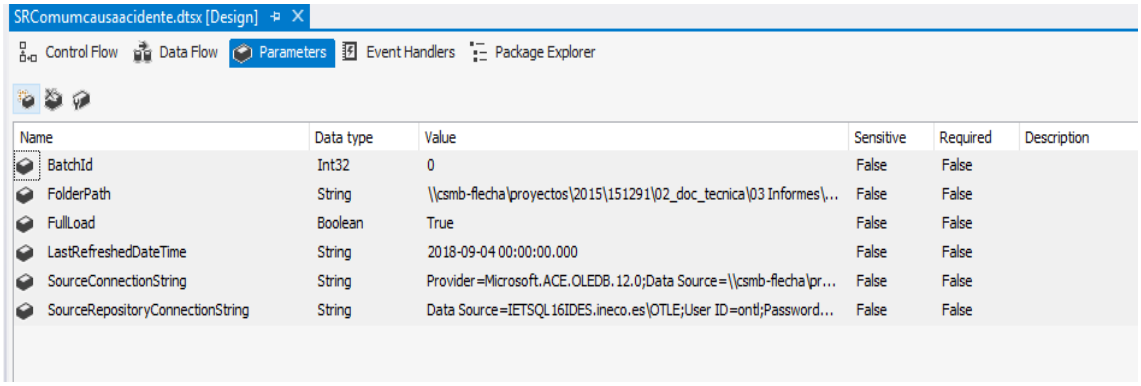
⇨ **Move File to ERROR:** Em caso de erro na execução, esta tarefa move o arquivo com nome alterado para a pasta ERROR. Este arquivo poderá ser revisado observando os erros detectados e armazenados em ETL.BatchDetail.

- **exsql – Table Success Log** → Em caso de sucesso no processamento, esta tarefa de execução SQL chama a StoredProcedure *ETL.uspLogTableCompletion* para registrar em ETL.BatchDetail o sucesso da carga dos dados na tabela destino.

- **exsql – Table Failed Log** → Em caso de erro no processamento, esta tarefa de execução SQL chama a StoredProcedure *ETL.uspLogTableCompletion* para registrar em ETL.BatchDetail o erro na carga dos dados na tabela destino.

A seguir, temos um conjunto de parâmetros necessários a cada pacote filho cujos valores serão recebidos de seu pacote pai:

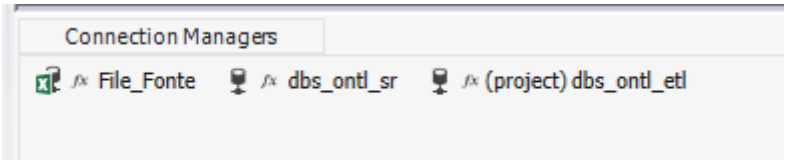
Figura 13. Parâmetros de um Pacote Filho



Fonte: Elaboração própria

➢ O pacote filho terá 3 gerenciadores de conexão à base de dados, como mostrado abaixo.

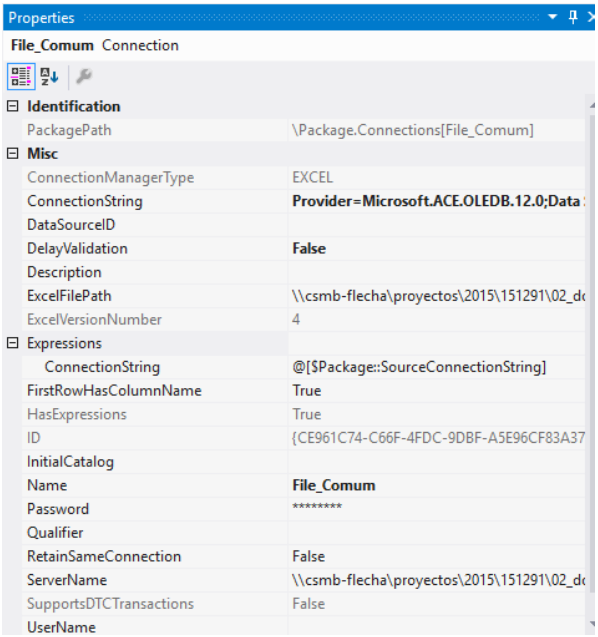
Figura 14. Administradores de conexões dos Pacotes Filhos



Fonte: Elaboração própria

- **File_Fonte** → Este gerenciador é utilizado para se conectar aos arquivos origem das fontes de dados e obtém seus dados de conexão do pacote pai através dos parâmetros do pacote filho em que se encontra, como mostrado a seguir, e se configura em tempo de execução.

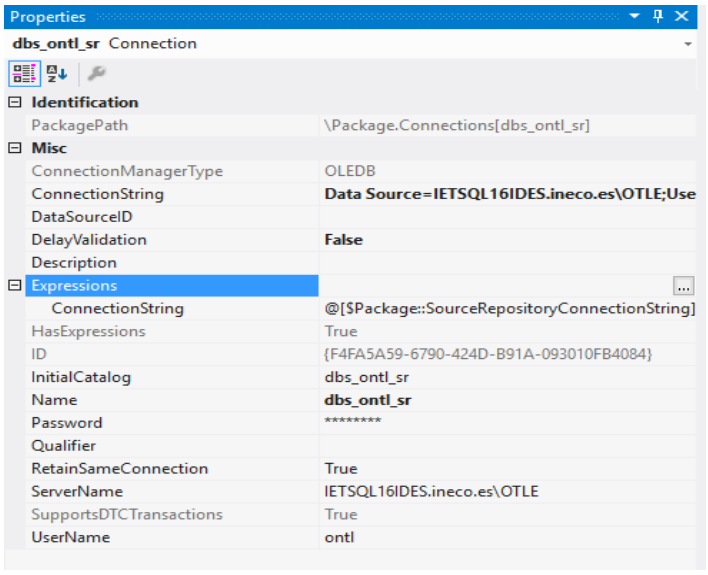
Figura 15. Parâmetros utilizados pelo gerenciador de conexão File_Fonte



Fonte: Elaboração própria

- **dbs_ontl_sr →** Este gerenciador é utilizado para se conectar ao banco de dados dbs_ontl_sr (*source repository*) e obtém seus dados de conexão do pacote pai através dos parâmetros do pacote filho em que se encontra (como mostrado a seguir) e se configura em tempo de execução.

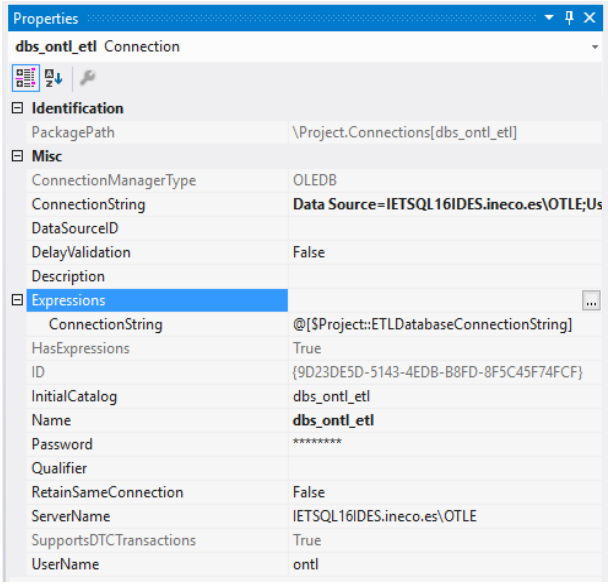
Figura 16. Parâmetros utilizados pelo gerenciador de conexões dbs_ontl_sr



Fonte: Elaboração própria

- **dbs_ontl_etl** – Este gerenciador é utilizado para se conectar ao banco de dados dbs_ontl_etl e obtém seus dados de conexão do pacote pai através dos parâmetros do pacote filho em que se encontra, como mostrado a seguir, e se configura em tempo de execução.

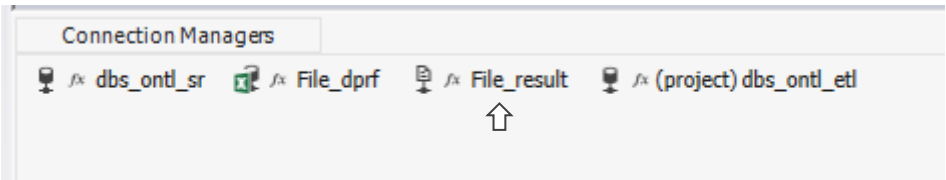
Figura 17. Parâmetros utilizados pelo gerenciador de conexões dbs_ontl_etl



Fonte: Elaboração própria

➢ Para os pacotes filhos de carga das tabelas de fatos é adicionada uma nova conexão:

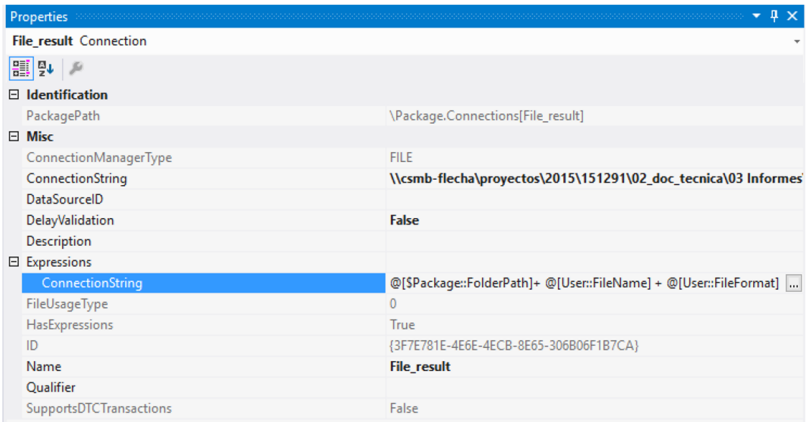
Figura 18. Novo gerenciador de conexões para tabelas de fatos



Fonte: Elaboração própria

⇨ **File_result:** Este gerenciador de conexões é utilizado para conectar-se com a localização onde estará o arquivo de resultado do processamento (dependendo do sucesso ou não da execução) e obtém sua informação de conexão dos parâmetros passados por seu pacote pai, como no exemplo a seguir:

Figura 19. Configuração da conexão File_Result para tabelas de fatos

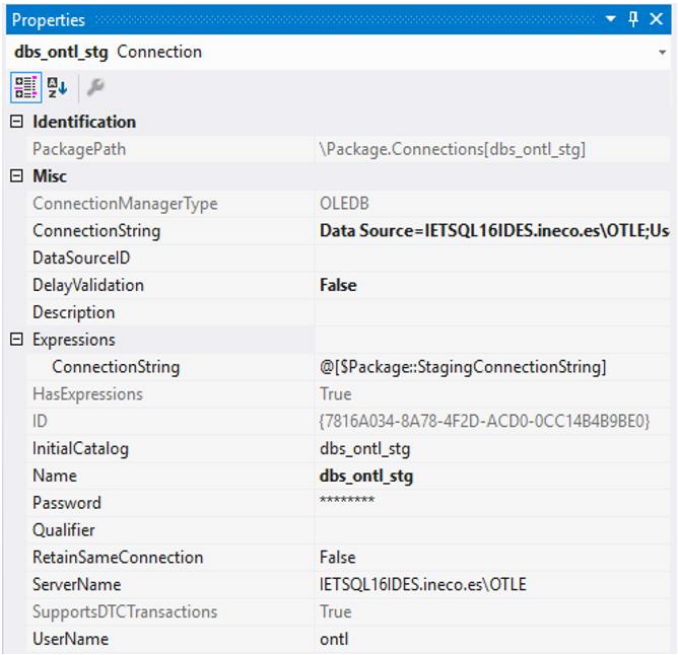


Fonte: Elaboração própria

- As demais conexões existentes variam dependendo da área de trabalho onde se encontre o processo de execução (*Staging* ou *DataWarehouse*):

  - **dbs_ontl_stg** – Este gerenciador de conexão é utilizado para se conectar ao banco de dados dbs_ontl_stg (*base da Staging Area*) e obtém seus dados de conexão do pacote pai através dos parâmetros do pacote filho em que se encontra (como mostrado a seguir) e se configura em tempo de execução.

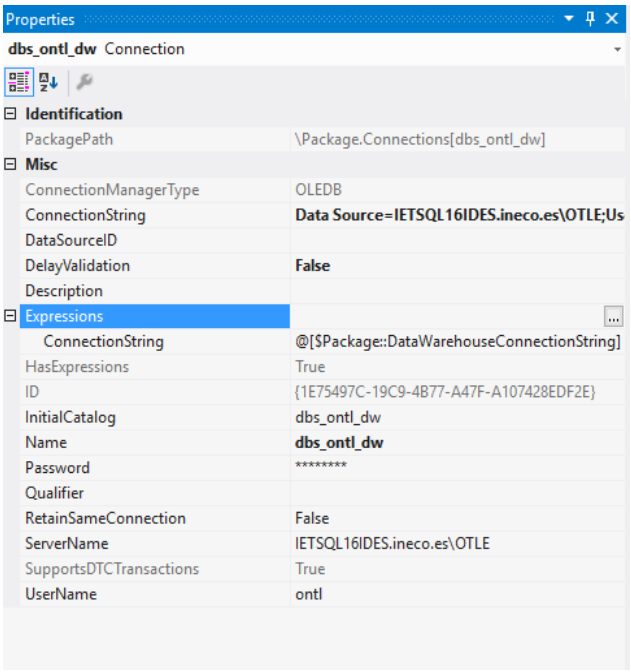
Figura 20. Gerenciador de conexões para a área de *staging*



Fonte: Elaboração própria

- **dbs_ontl_dw** – Este gerenciador de conexão é utilizado para se conectar ao banco de dados dbs_ontl_dw (*base do Datawarehouse*) e obtém seus dados de conexão do pacote pai através dos parâmetros do pacote filho em que se encontra (como mostrado a seguir) e se configura em tempo de execução.

Figura 21. Novo gerenciador de conexões para a área do DataWarehouse



Fonte: Elaboração própria

7.3 Estrutura das tabelas utilizadas para administração e controle dos processos ETL

Durante o trabalho dos processos de ETL definidos no framework para o ONTL são utilizadas 4 tabelas:

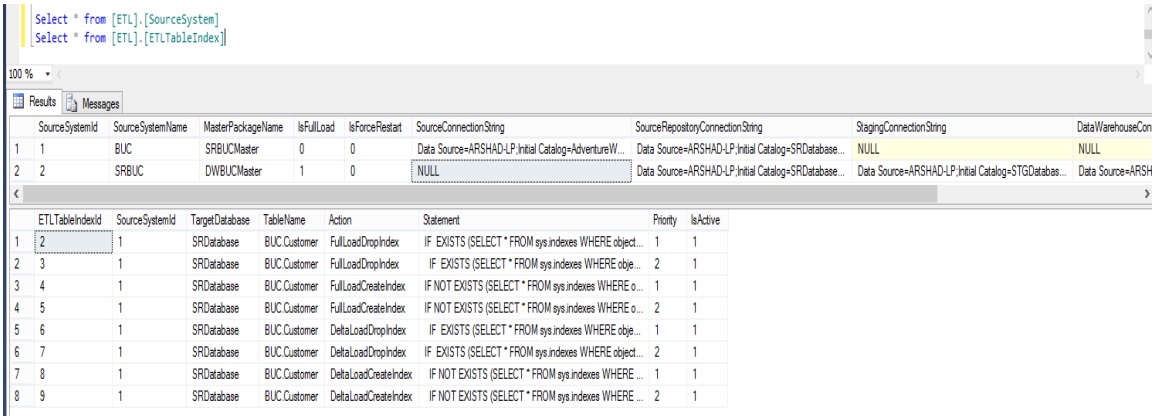
- **ETL.SourceSystem**
  Esta tabela guarda informação relacionada à origem de cada fonte, por exemplo, cadeias de conexão do sistema fonte, métodos de autenticação, se a carga deve ser completa ou incremental, se deve começar a partir do POF (ponto de falha) ou começar um novo lote de execuções para extrair dados, dentre outros detalhes desse tipo.

Campos da Tabela	Descrição do campo
SourceSystemId	Número gerado automaticamente.
SourceSystemName	Nome do sistema fonte.
MasterPackageName	Nome do pacote pai que será chamado para a carga de dados desta fonte em particular.
IsFullLoad	Este flag indica carga completa (1) ou incremental (0). De forma predeterminada, a carga completa se realizará na primeira execução do processo e as demais serão incrementais. Esse indicador ajuda a configurar algo diferente a partir do pacote Pai, podendo assim ser configurado para realizar uma carga completa desde a origem dos dados para o pacote de uma fonte específica.
IsForceRestart	Este flag indica se o pacote requer forçar o reinício (1) ou não (0). Por default, ao ser reiniciado depois de uma falha, o marco de trabalho do processo começaria a carregar dados apenas para as tabelas falhadas e não faria nada com as tabelas que se completaram na última execução. Este comportamento predeterminado pode ser alterado com essa configuração para realizar uma carga total mesmo depois de falha.
SourceConnectionString	Cadeia de conexão do sistema de fontes (arquivos excel o csv).
SourceRepositoryConnectionString	Cadeia de conexão da base de dados do repositório de origem (*source repository* – dbs_ontl_sr).
StagingConnectionString	Cadeia de conexão da base de dados temporária (*staging área* – dbs_ontl_stg).
DataWarehouseConnectionString	Cadeia de conexão da base de dados de exploração do Observatório (*DataWarehouse* – dbs_ontl_dw).
LastRefreshDateTime	Última data e hora de atualização. Indica a data e hora em que a extração de dados desta fonte em particular foi completada com sucesso e que foi usada para extrair dados de forma incremental.
FolderPath	Caminho da pasta onde estarão disponíveis os arquivos origem da fonte cuja carga esteja baseada em sistema de arquivos.

- **ETL.ETLTableIndex**
  Esta tabela mantém informação relacionada aos índices que se descartam antes de uma carga de dados e que voltarão a ser criados depois dela bem como a prioridade de execução dos mesmos. Isso assegura que a fragmentação de índices esteja sob controle.

Campos da Tabela	Descrição do campo
ETLTableIndexId	Número gerado automaticamente.
SourceSystemId	Referência à tabela do sistema fonte.
TargetDatabase	Nome da base de dados.
TableName	Nome da tabela.
Action	Estas são as 4 ações em uso atualmente: • FullLoadDropIndex • FullLoadCreateIndex • DeltaLoadDropIndex • DeltaLoadCreateIndex
Statement	Declaração do índice que se executará em função da ação.
Priority	Sequência de execução de extração.
IsActive	Flag para indicar Ativo (1) ou Inativo (0).

Figura 22. Exemplos de dados encontrados nas tabelas Source System e ETLTableIndex



```
Select * from [ETL].[SourceSystem]
Select * from [ETL].[ETLTableIndex]
```

100 %

Results | Messages

	SourceSystemId	SourceSystemName	MasterPackageName	IsFullLoad	IsForceRestart	SourceConnectionString	SourceRepositoryConnectionString	StagingConnectionString	DataWarehouseCon
1	1	BUC	SRBUCMaster	0	0	Data Source=ARSHAD-LP;Initial Catalog=AdventureW...	Data Source=ARSHAD-LP;Initial Catalog=SRDatabase...	NULL	NULL
2	2	SRBUC	DWBUCMaster	1	0	NULL	Data Source=ARSHAD-LP;Initial Catalog=SRDatabase...	Data Source=ARSHAD-LP;Initial Catalog=STGDatabas...	Data Source=ARSH

	ETLTableIndexId	SourceSystemId	TargetDatabase	TableName	Action	Statement	Priority	IsActive
1	2	1	SRDatabase	BUC.Customer	FullLoadDropIndex	IF EXISTS (SELECT * FROM sys.indexes WHERE object...	1	1
2	3	1	SRDatabase	BUC.Customer	FullLoadDropIndex	IF EXISTS (SELECT * FROM sys.indexes WHERE obje...	2	1
3	4	1	SRDatabase	BUC.Customer	FullLoadCreateIndex	IF NOT EXISTS (SELECT * FROM sys.indexes WHERE o...	1	1
4	5	1	SRDatabase	BUC.Customer	FullLoadCreateIndex	IF NOT EXISTS (SELECT * FROM sys.indexes WHERE o...	2	1
5	6	1	SRDatabase	BUC.Customer	DeltaLoadDropIndex	IF EXISTS (SELECT * FROM sys.indexes WHERE obje...	1	1
6	7	1	SRDatabase	BUC.Customer	DeltaLoadDropIndex	IF EXISTS (SELECT * FROM sys.indexes WHERE object...	2	1
7	8	1	SRDatabase	BUC.Customer	DeltaLoadCreateIndex	IF NOT EXISTS (SELECT * FROM sys.indexes WHERE ...	1	1
8	9	1	SRDatabase	BUC.Customer	DeltaLoadCreateIndex	IF NOT EXISTS (SELECT * FROM sys.indexes WHERE ...	2	1

Fonte: Elaboração própria

- **ETL.BatchHeader**
  Esta tabela armazena os metadados de execução a nível de pacote pai, por exemplo, quando se iniciou o pacote, quando foi finalizado, qual foi o resultado da última execução, mensagens de erro se houve alguma exceção.

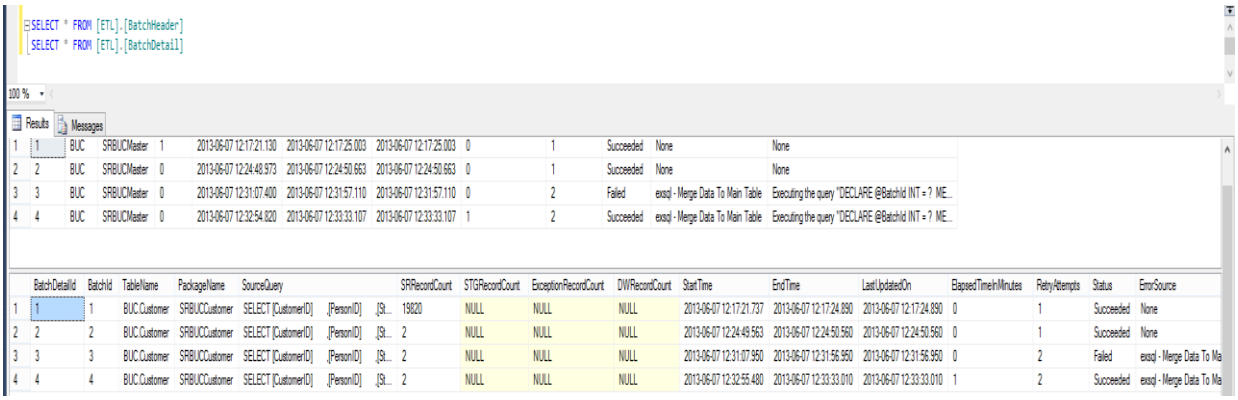
Campos da Tabela	Descrição do campo
BatchId	Número gerado automaticamente.
Source	Informação da tabela fonte.
PackageName	Nome do pacote.
FullLoad	Flag que indica carga completa (1) ou incremental (0).
BatchStartTime	Indica a hora em que se iniciou a execução do pacote.
BatchEndTime	Indica a hora em que a execução do pacote foi finalizada.
LastUpdatedOn	Data e hora da última atualização.
ElapsedTimeInMinutes	Diferença entre a hora de início e a hora de finalização, em minutos.
RetryAttempts	Número de tentativas em caso de falha.
BatchStatus	Indica o estado de execução do pacote: *Running* (em execução); *Succeeded* (finalizado com sucesso) ou *Failed* (finalizado com erro).
ErrorSource	Em caso de erro, indica qual componente de SSIS falhou.
ErrorMessage	Mensagem de erro real registrada através do controlador de erros do SSIS durante a falha.

- **ETL.BatchDetail**
  Esta tabela armazena os metadados de execução a nível de pacote filho, por exemplo, número de registros movidos da fonte, qual foi o resultado da última execução, mensagens de erro se houve alguma exceção, dentre outras coisas.

Campos da Tabela	Descrição do campo
BatchDetailId	Número gerado automaticamente.
BatchId	Referência à tabela de ETL.BatchHeader mediante BatchId.
TableName	Nome da tabela destino.
PackageName	Nome do pacote filho.
SourceQuery	Consulta utilizada para extrair dados da fonte.
SRRecordCount	Quantidade de registros carregados na tabela do repositório de origem (SR).
STGRecordCount	Quantidade de registros carregados na tabela intermediária (de *staging* - STG).
DWRecordCount	Quantidade de registros carregados na área de exploração do Observatório (*DataWarehouse* - DW).
StartTime	Indica a hora em que se iniciou a execução do pacote.
EndTime	Indica a hora em que a execução do pacote foi finalizada.
LastUpdatedOn	Data e hora da última atualização.
ElapsedTimeInMinutes	Diferença entre a hora de início e a hora de finalização, em minutos.
RetryAttempts	Número de tentativas em caso de falha.

Campos da Tabela	Descrição do campo
Status	Indica o estado de execução do pacote: *Running* (em execução); *Succeeded* (finalizado com sucesso) ou *Failed* (finalizado com erro).
ErrorSource	Em caso de erro, indica qual componente de SSIS falhou.
ErrorMessage	Mensagem de erro real registrada através do controlador de erros do SSIS durante a falha.

Figura 23. Exemplos de dados encontrados nas tabelas BatchTableHeader e BatchDetail



Fonte: Elaboração própria

7.4 Fluxo de execução de pacotes no Framework

Para ilustrar melhor o fluxo de trabalho dos pacotes sugeridos para as ETLs do Observatório Nacional de Transportes e Logísticas do Brasil, o mesmo será esquematizado e descrito a seguir.

7.4.1 Fluxo de um Pacote Pai

O esquema abaixo ilustra o fluxo de trabalho de um pacote pai:

Package Start Logger	•Registra o horário de inicio do pacote e atualizada a tabela BatchHeader. •Contrói os argumentos necessários para os pacotes filhos.
Sequence Master Staging	•Executa todos os pacotes filhos. •Passa todas as informações aos pacotes filhos.
Package Success Log	•Finaliza o pacote (em caso de sucesso). •Registra o status da execução.
Package Error Log	•Em caso de erro: loga o status da execução do pacote. •Pára o pacote e registra os erros encontrados.

7.4.2 Fluxo de um Pacote Filho

O esquema a seguir ilustra o fluxo de trabalho de um pacote filho:

Package Start Logger	•Registra a execução do pacote na tabela BatchDetail. •Inicia os pacotes depois de registrar seu status em execução.
Start Package Execution	•Determina se o fluxo de carga será completo ou incremental. •Processa as ETLs baseadas no fluxo de carga definido.
Package Success Log	•Finaliza o pacote (em caso de sucesso). •Registra o status da execução.
Package Error Log	•Em caso de erro: loga o status da execução do pacote. •Pára o pacote e registra os erros encontrados.

A informação de registro de execução será armazenada em duas tabelas da base *dbs_ontl_etl*:

- [ETL]. [BatchHeader] – armazena informações relacionadas à execução de pacotes pais.
- [ETL]. [BatchDetail] – armazena informações relacionadas à execução de pacotes filhos.

7.4.3 Fluxo de trabalho para a carga completa (*Work Flow – Full Load*)

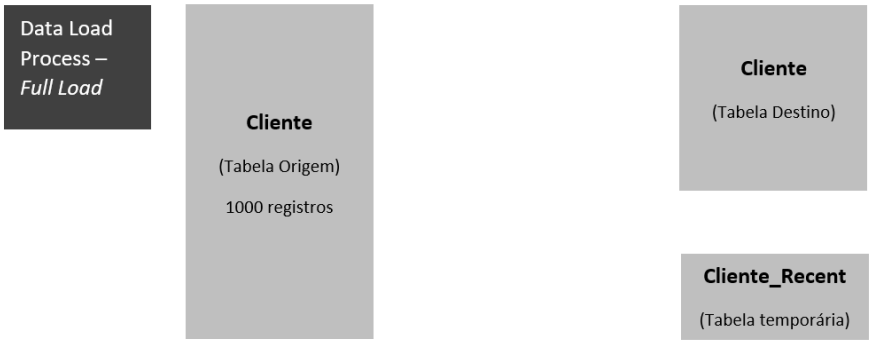
A carga completa começa por truncar o destino e ler os registros da tabela de origem. Antes de carregar a tabela de destino seus índices são apagados (lendo as configurações referentes a isso – já citadas anteriormente). Tais índices são recriados depois da carga da tabela.

Passo a passo:

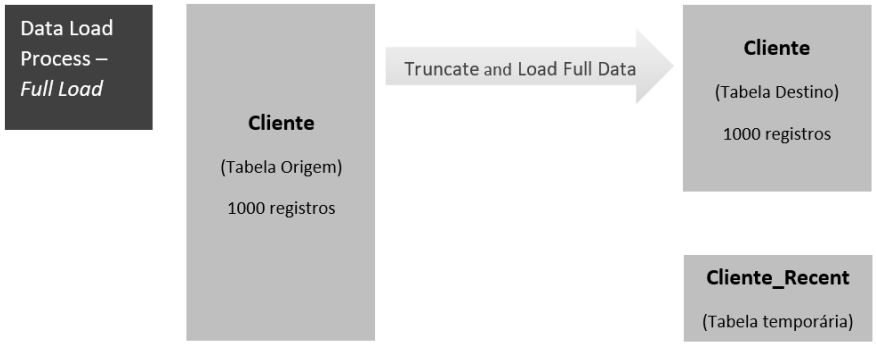
1. Truncar a tabela destino
2. Apagar índices (DROP INDEX) na tabela destino, caso esteja configurado para isso.
3. Realizar ETLs.
4. Recriar os índices apagados, caso esteja configurado para isso.
5. Gerar uma *hashkey* (chave de identificação unívoca) para cada linha da tabela.

Ilustrando com um exemplo:

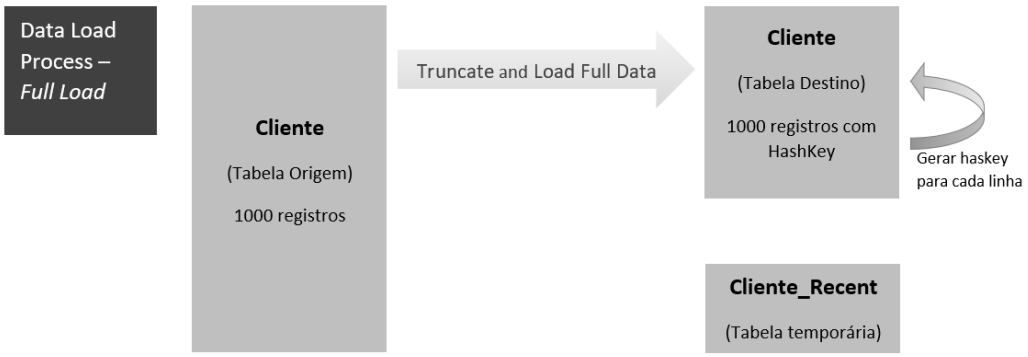
- Considere que há 1.000 registros na fonte de dados:



- O processo de carga completa trunca a tabela destino para assegurar de que não tenha dados da carga anterior e carrega o conjunto completo dos dados da origem para a tabela destino. Como mostrado a seguir, 1000 registros são levados da tabela origem para a tabela destino:



- Em seguida, é executado um processo para gerar a *hashkey* de cada linha da tabela destino (concatenando valores de todas as colunas dessa linha). Essa chave é armazenada no campo vbn_etlhashkey.



7.4.4 Fluxo de trabalho para a carga incremental (*Work Flow – Delta Load*)

O processo para a carga incremental dos dados carregará somente as alterações ocorridas desde a última carga realizada. Para realizar a carga incremental será utilizada uma tabela intermediária entre a tabela de origem e a tabela de destino - uma tabela temporária que será criada em tempo de execução do pacote de carga, que só existirá durante a execução de tal pacote e que será chamada de <nome_tabela_destino>_recent.

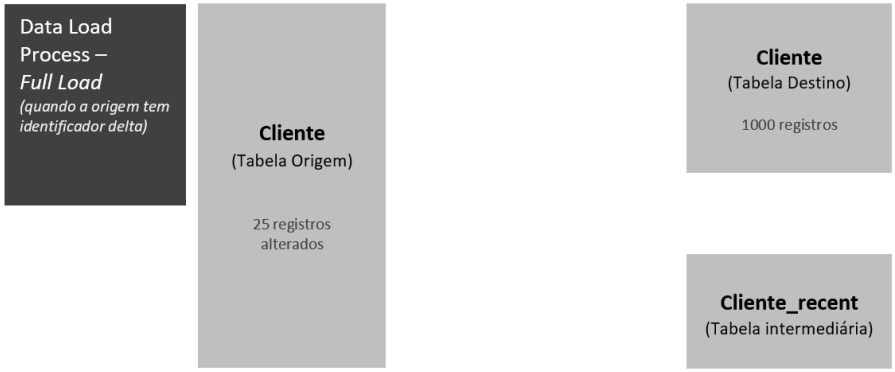
Todas as alterações recentes serão primeiramente carregadas na tabela intermediária e será gerada uma *hashkey* para cada uma das linhas dessa tabela. Para extrair os registros alterados recentemente na fonte será utilizado o campo de data da tabela fonte, comparado com a última data de atualização e serão inseridos na tabela intermediária apenas os registros alterados entre essas datas. Se a tabela fonte não tem campo de data, serão carregados todos os registros para a tabela intermediária e serão geradas hashkeys para cada linha desses registros. Em seguida, será realizada a fusão dos dados da tabela intermediária com a tabela destino através dessa hashkey.

Passo a passo:

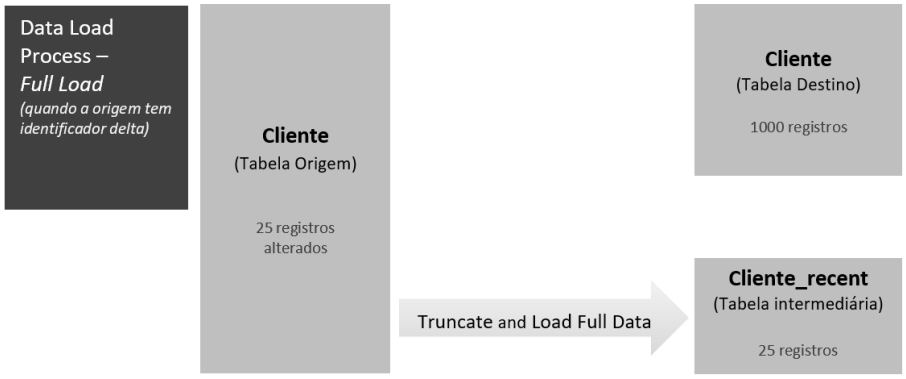
1. Truncar a tabela intermediária.
2. Obter todos os dados que foram criados ou modificados desde a última carga realizada.
3. Inserir esses dados na tabela intermediária.
4. Gerar uma *hashkey* (chave de identificação unívoca) para cada linha da tabela intermediária.
5. Apagar índices (DROP INDEX) na tabela destino, caso esteja configurado para isso.
6. Adicionar (MERGE) os registros modificados na tabela de destino.
7. Recriar os índices apagados, caso esteja configurado para isso.

Ilustrando com um exemplo (de quando a origem tem um identificador delta):

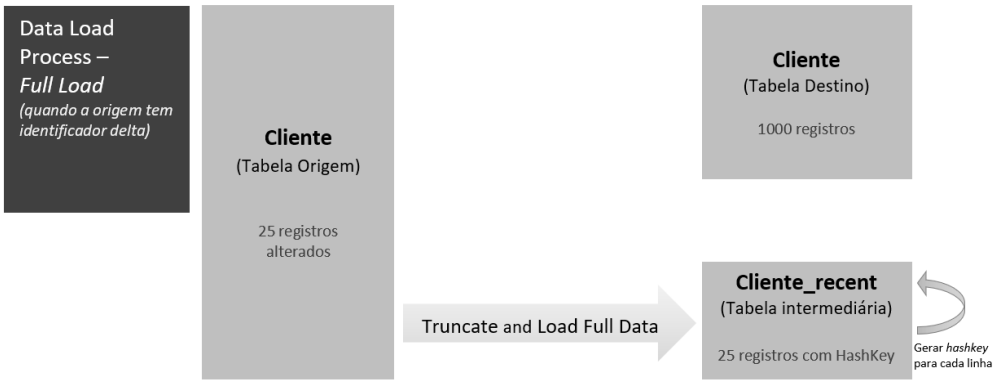
- Lembrando que, desde a última carga completa, temos 1000 registros na tabela destino. Ao realizar novo processo de ETL, são detectados 25 registros alterados/incluídos na fonte (identificados usando data e hora da fonte e fazendo uma comparação com a última data e hora de carga realizada na tabela destino).



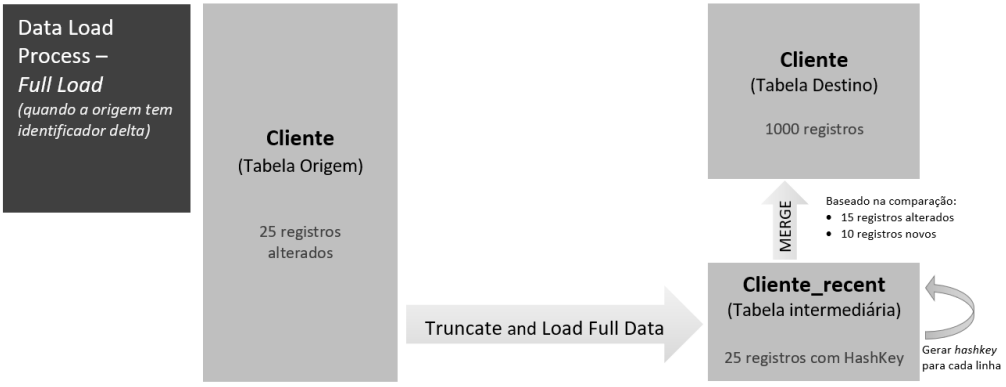
- Ao iniciar a execução do processo de ETL no modo delta, a tabela intermediária é truncada e os 25 registros alterados/incluídos são levados para ela.



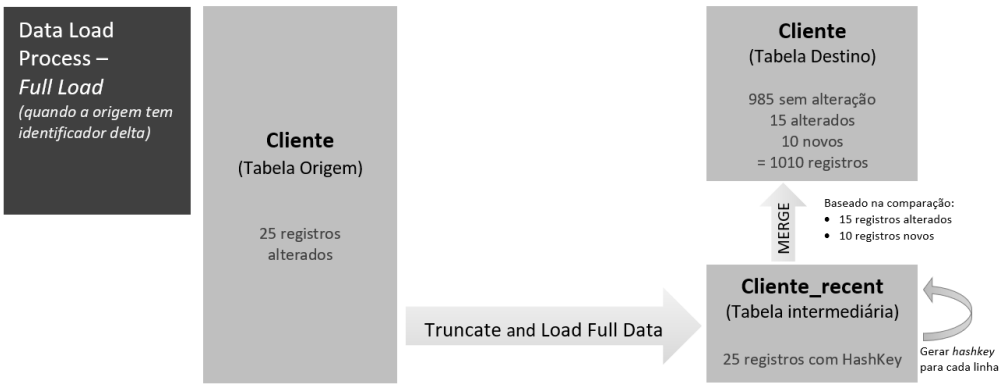
- O processo gera a *hashkey* de cada registro da tabela intermediária.



- Em seguida, o processo realiza uma comparação entre a tabela intermediária e a tabela de destino e fusiona (aglutina) os registros na tabela de destino utilizando as seguintes regras de comparação:
  - Se o ID do registro da tabela intermediária não existe na tabela destino, insere o registro.
  - Se o ID do registro da tabela intermediária existe na tabela destino, mas a *hashkey* é diferente, atualiza o registro.
  - Se coincidem o ID e a *hashkey* do registro, não faz nada.



- Durante a fusão foram identificados 10 registros cujo ID não existe na tabela destino e, portanto, são considerados novos registros e são inseridos na tabela destino (INSERT). Para outros 15 registros, coincidiram os IDs, mas as *hashkeys* eram diferentes e, portanto, esses registros foram atualizados (UPDATE). Os demais 985 registros da tabela destino não sofreram nenhuma alteração.



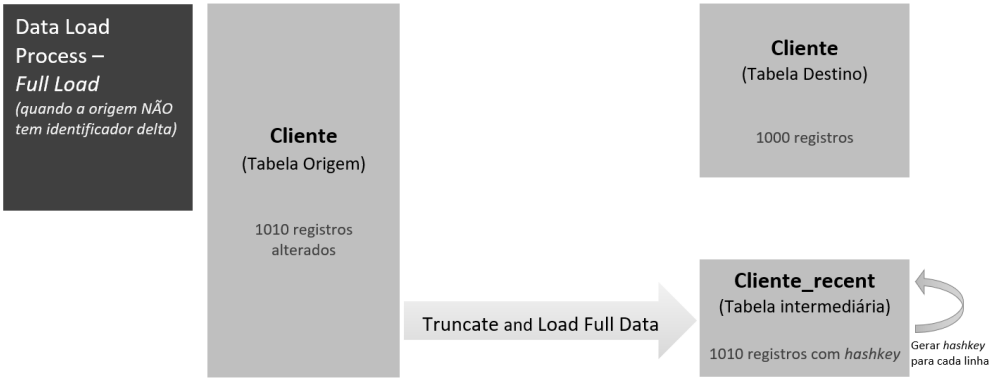
Ilustrando com um exemplo (de quando a origem não tem um identificador delta):

O exemplo anterior trata de um cenário onde a origem possui algum campo de data e hora que permite identificar registros delta (modificados de alguma forma). Agora, vejamos como funciona a carga incremental quando não há um campo como esse.

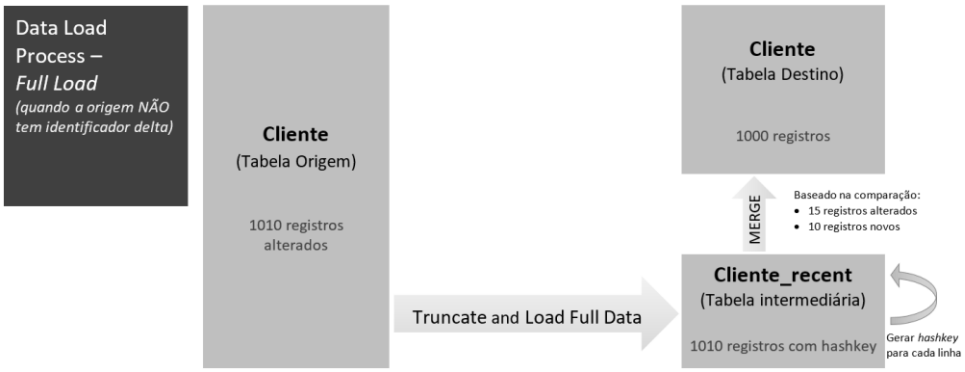
- Lembrando novamente que, desde a última carga completa, temos 1000 registros na tabela destino. Se supõe que, ao executar o processo ETL outra vez, haverá poucas alterações, mas não sabemos quais registros foram agregados ou alterados porque nesse caso não temos o campo de data e hora na fonte para fazer a comparação.

- Nesse caso, ao iniciar a execução do processo de ETL no modo delta, a tabela intermediária é truncada e todos os 1010 registros da fonte são levados para ela.

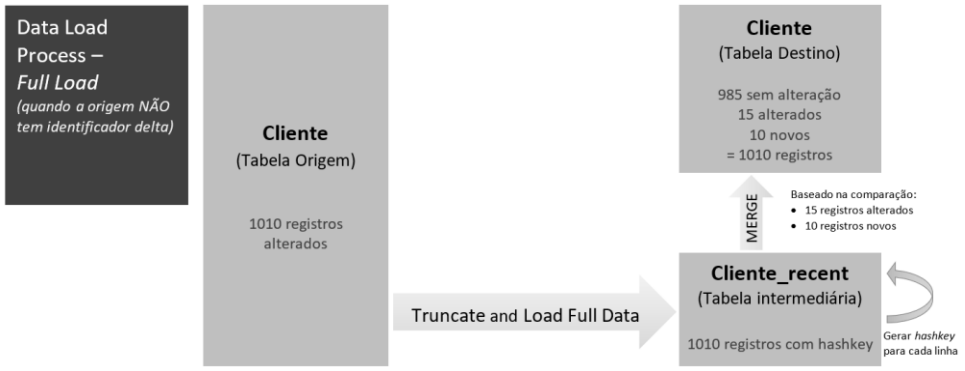
- O processo gera a *hashkey* de cada registro da tabela intermediária.



- Em seguida, o processo realiza uma comparação entre todos os 1010 registros da tabela intermediária e os registros da tabela de destino para, em seguida, fusionar tais registros na tabela de destino utilizando as mesmas regras de comparação:
  - Se o ID do registro da tabela intermediária não existe na tabela destino, insere o registro.
  - Se o ID do registro da tabela intermediária existe na tabela destino, mas a *hashkey* é diferente, atualiza o registro.
  - Se coincidem o ID e a *hashkey* do registro, não faz nada.



- Durante a fusão foram identificados 10 registros cujo ID não existe na tabela destino e, portanto, são considerados novos registros e são inseridos na tabela destino (INSERT). Para outros 15 registros, coincidiram os IDs, mas as *hashkeys* eram diferentes e, portanto, esses registros são atualizados (UPDATE). Os demais 985 registros da tabela destino não sofreram nenhuma alteração.



8 RESUMO E CONCLUSÕES

Como abordado ao longo desse documento, após a obtenção dos dados fornecidos pelas diversas fontes, faz-se necessário o tratamento dos mesmos para que possam ser de maior proveito para o trabalho do Observatório.

O presente documento descreveu a ampliação da base de dados que se tornou necessária para o acompanhamento da execução dos processos de carga e para otimizar sua utilização por ferramentas de BI. Além disso, sugeriu uma maneira para a disponibilização dos dados utilizados de modo a padronizar e viabilizar o início dos trabalhos dos processos de ETL. E, finalmente, como foco principal, descreveu detalhadamente o Framework criado para a implementação dos Serviços de Extração, Transformação e Carga dos dados iniciais na Base de Dados do ONTL.

Anexo a esse documento, com objetivo de facilitar e agilizar a implementação dos recursos aqui descritos, é incluída uma seção sobre a base de dados utilizada pela área de auditoria e controle dos processos ETLs baseados no Framework descrito.

O próximo documento da fase 3 revisitará os requisitos relacionados ao Serviço de Transferência de Dados levantados na primeira fase desse projeto e indicará como esses foram ou poderão ser atendidos com a implementação dos recursos aqui descritos.

9 APROVAÇÕES

Nome: Fernando Cámara de la Peña Cargo/Função: Técnico da Ineco alocado na sede da EPL Data:	Nome: Enrique Monfort Tomo Cargo/Função: Diretor de Projetos INECO do BRASIL Data:

Nome: Milton Sampaio Castro de Oliveira Cargo/Função: Assessor técnico da coordenação do Observatório - CONIL Data:	Nome: Lilian Campos Soares Cargo/Função: Coordenadora do Observatório - CONIL Data:

Nome: Jony Marcos do Valle Lopes

Cargo/Função: Gerente de Pesquisa e Desenvolvimento Logístico – GEPDL

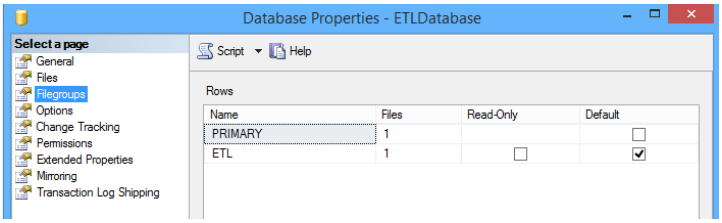
Data:

ANEXO I – CRIAÇÃO DA BASE DE CONTROLE DAS ETLS (DBS_ONTL_ETL)

Objetivando viabilizar, auxiliar e agilizar a implementação do Framework definido nesse documento, esse anexo se dedica a indicar criação e configuração da base de dados que será utilizada pela área de auditoria e controle dos processos de ETL que serão desenvolvidos: dbs_ontl_etl.

Antes de executar o script, é necessário que a base de dados esteja criada e configurada para tal.

Figura 24. Base de Dados dbs_ontl_etl criada



Fonte: Elaboração própria

Com a base de dados disponibilizada, é possível executar o script do arquivo abaixo para criar todos os objetos de dados necessários para a implementação e execução dos processos baseados no Framework ETL.

script_dbs_ontl_etl.sql