Guia de Boas Práticas de Desenvolvimento Oracle Data Integrator (ODI)

Preparado por: José Carlos, Paulo Santoro

Data: 14/03/19

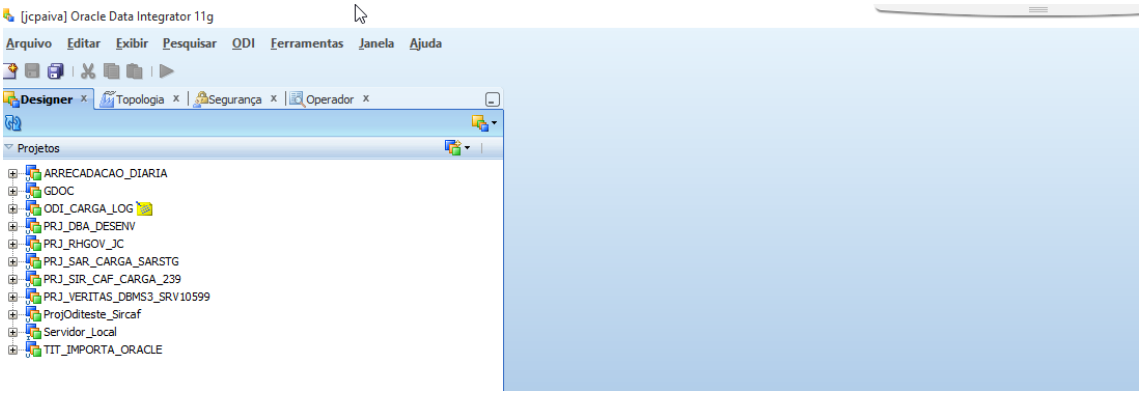
Sumário

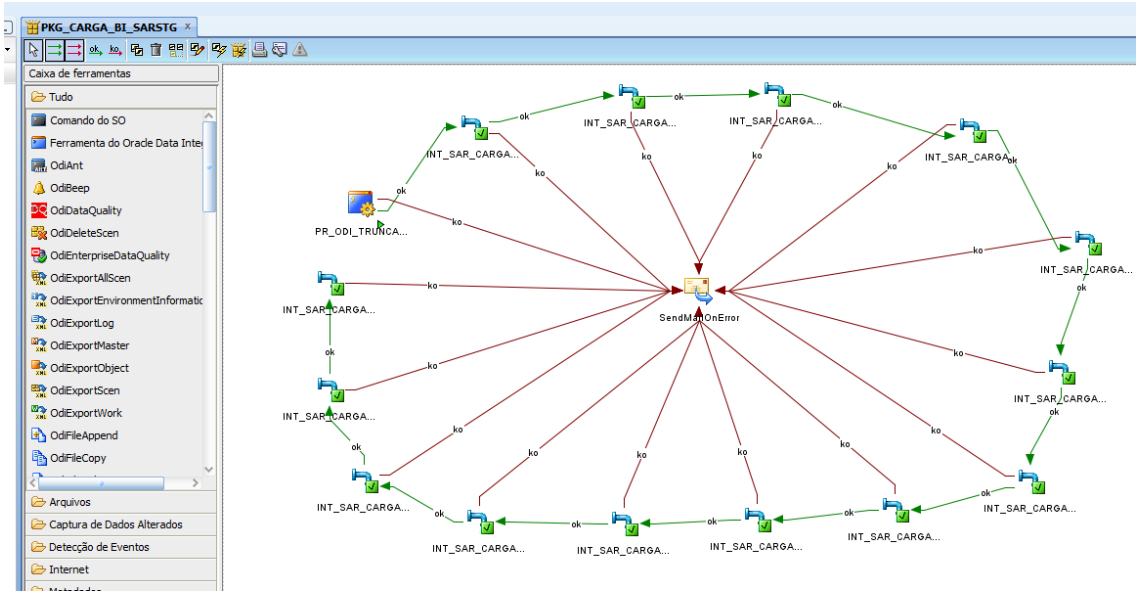
O objetivo deste documento é buscar, através das melhores práticas de desenvolvimento de pacotes em Oracle Data Integrator, obter ganhos de desempenho, além de sugerir melhores formas de Implementar, desenhar e monitorar os pacotes.

Este guia pode ser usado por qualquer time responsável pelo desenvolvimento de pacotes em Oracle Data Integrator (ODI).

É recomendado que este sirva de base para o desenvolvimento de pacotes com maior qualidade, evitando o uso de certas práticas que levam a um mau desempenho do mesmo.

Este manual se aplica a todas as atuais versões do Oracle Data Integrator (ODI).





Melhores práticas

O Oracle Data Integrator (ODI) é um produto muito poderoso quando utilizado corretamente. Infelizmente, alguns erros podem levar a resultados desastrosos em projetos de integração de dados. Aqui abordo melhores práticas para evitar os erros mais comuns em projetos com Oracle Data Integrator.
Além de padronizar e otimizar processos de ETL, o ODI é capaz de fazer a integração de diferentes tecnologias e bancos de dados em um único lugar, facilitando o trabalho de qualquer projeto que necessite fazer integração de dados.

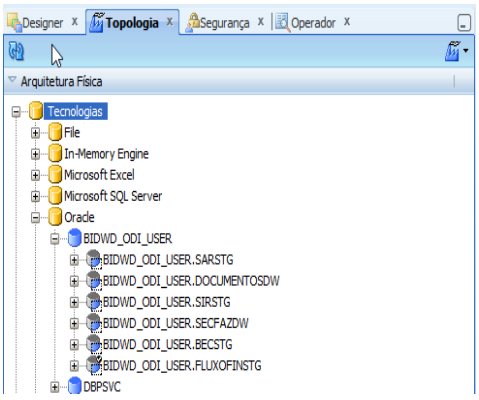
Este manual assume que o leitor tem alguma familiaridade com o Oracle Data Integrator.

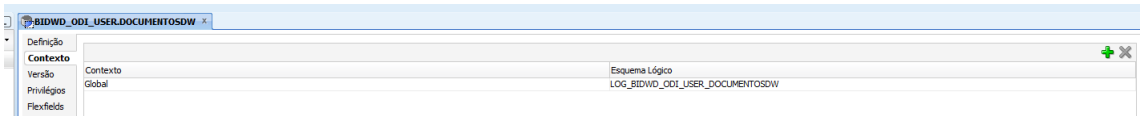
1 - Entenda e Use Corretamente os Contextos e a Topologia

Os Contextos e a Topologia são algumas das características mais poderosas do ODI em tempo de design e em tempo de execução de artefatos e vários ambientes diferentes.
No ODI, todos os desenvolvimentos bem como as execuções são feitos sobre uma Arquitetura Lógica (schemas lógicos, agentes lógicos), que se resolvem em um dado Contexto para uma Arquitetura Física (fonte de dados física/servidores de dados de destino/schemas e agentes de run-time do ODI). Os Contextos nos permitem chavear a execução dos artefatos de um ambiente (Contexto) para outro.
Agora, leia o parágrafo anterior novamente. Tenha certeza de que você compreendeu o conceito de Contexto, Arquitetura Lógica e Arquitetura Física.
Dois erros comuns que são cometidos em relação aos Contextos:

- Esquecer de mapear as arquiteturas física e lógica para um determinado Contexto. Isso é um erro de administração de Topologia que leva a falhas de execução em um dado Contexto. Para evitar isso, garanta que todos os recursos lógicos estão mapeados para recursos físicos nesse Contexto.
- Um grande erro é forçar o Contexto quando não é necessário. Nas interfaces ou nas procedures, existem caixas de seleção com a lista de Contextos, defina como “default” ou “execution context”. A não ser que seja realmente necessário forçar o contexto por uma razão funcional, deixe as caixas de seleção como estiverem. São raros os casos onde é realmente necessário forçar o Contexto.

Resumindo: Garanta que o seu entendimento sobre Contextos e Topologia esteja sólido. Defina cuidadosamente a Topologia e o mapeamento dos Contextos. Evite forçar Contextos.

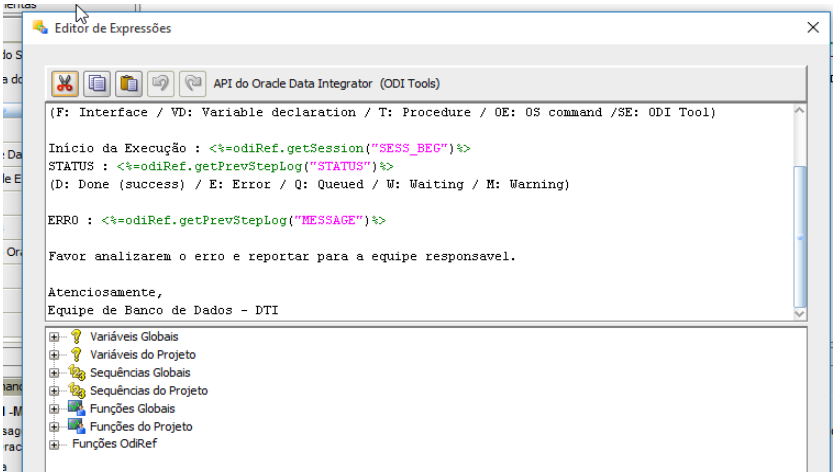


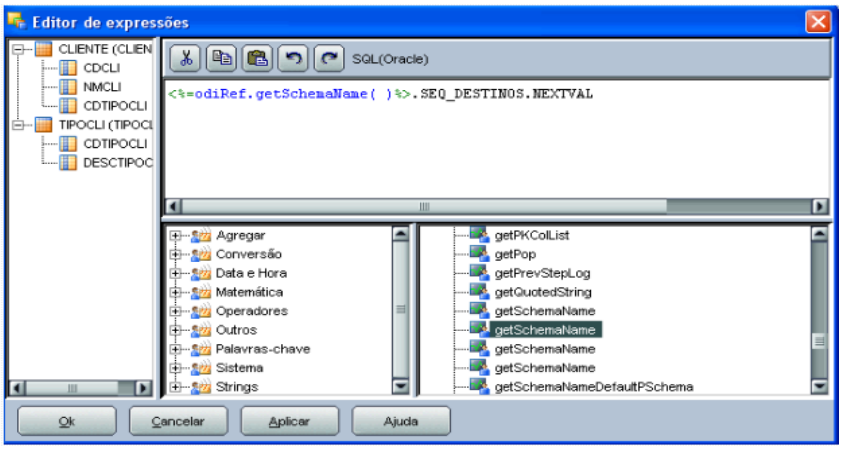


2 – Design Independente de Contexto

Em vários tipos de artefatos do ODI (procedures, variáveis, interfaces, etc.) é possível adicionar código SQL. Um erro muito comum é inserir nomes de objetos qualificados, como no exemplo abaixo que faz um DROP na tabela TEMP_001 num schema de staging:
DROP TABLE STAGING.TEMP_001
Isso é um "código dependente de contexto". Se você executa esse código em ambiente de produção onde a área de staging se chama PRD_STG, seu código não funciona. Perceba que os nomes dos schemas são definidos na Topologia, e os contextos acessam o schema correto dependendo do contexto de execução.
Nesse caso a pergunta é: Como usar isso no seu código?
Os Métodos de Substituição (OdiRef API) existem para disponibilizar no seu código os metadados armazenados no ODI tornando o código independente de contexto. Sendo assim, eles garantem que o nome qualificado da tabela em questão será gerado de acordo com o contexto onde o código está sendo executado.
Utilizando os Métodos de Substituição, o código genérico ficaria assim:
DROP TABLE `<%odiRef.getObjectName("L", "TEMP_001","W")%>`.

Consulte o Substitution Methods Reference Guide para mais informações sobre como utilizar essa API. O "expression editor" também ajuda muito.
Resumindo: Tão logo você comece a digitar um nome de schema, nome de database, nome de usuário ou qualquer informação referente à um servidor ou schema, pare, respire fundo e considere utilizar um Método de Substituição.
Exemplo do Uso das APIs do ODIRef.



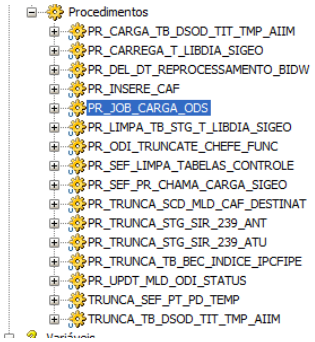


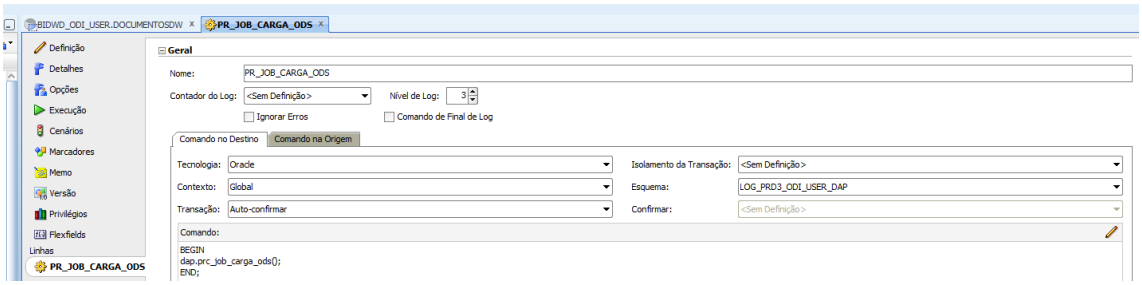
3 – Utilize Procedures Apenas Quando Necessário

As procedures permitem a execução de ações bem complexas, incluindo comandos SQL. Além disso, elas permitem a utilização das conexões source e target e ainda suportam binding. Isso significa que é possível mover dados de um lado para o outro usando apenas procedures.
Os desenvolvedores que se sentem à vontade com código SQL ficam seriamente tentados a escrever código para fazer transformações e movimentação de dados ao invés de desenvolver interfaces.
Existem alguns problemas quanto à isso:

- Procedures contém código manual que precisa sofrer manutenções periódicas;
- Procedures não mantém referências com os outros artefatos ODI como datastores, modelos, etc., fazendo com que a manutenção seja muito mais complexa comparado às interfaces.

Procedures nunca devem ser utilizadas para mover ou transformar dados, essas operações devem ser feitas utilizando as interfaces.





Resumindo: Quando você começar a usar procedures para mover/transformar dados provavelmente você está fazendo uso inadequado das procedures e deveria usar interfaces no lugar delas.

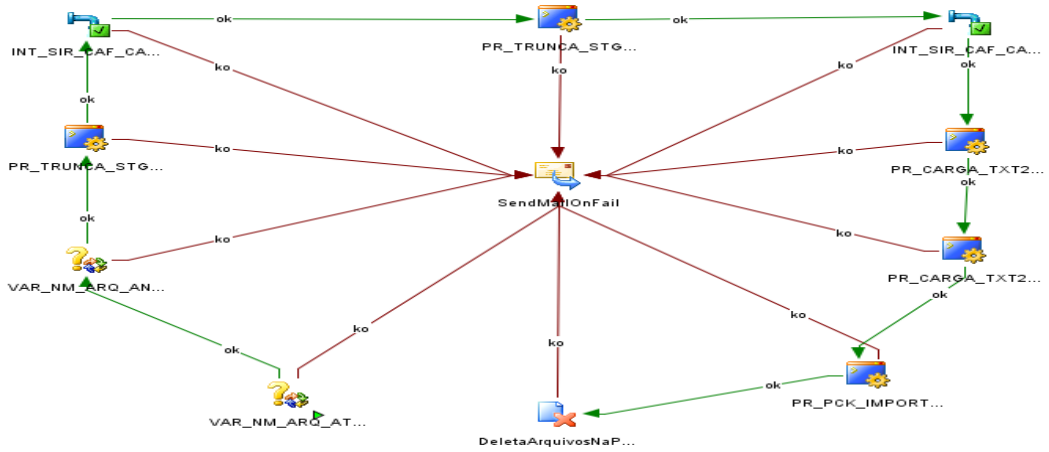
4 – Garantir Qualidade de Dados

Às vezes o líder de projeto de integração de dados não leva em conta a qualidade dos dados. Isso é um erro comum. O processo de integração pode estar movendo e transformando lixo e propagando esse lixo para outras aplicações.
O ODI permite que a qualidade dos dados seja garantida na fonte, (source) utilizando static checks ou então durante o processo de integração antes de gravar no destino (target) utilizando flow checks. Utilizando esses dois mecanismos de checagem de dados é possível garantir a qualidade dos dados.

Resumindo: Garanta a qualidade dos dados usando os dois métodos: static checks e flow checks. Qualidade de dados não é uma opção.

5 – Capturar Erros em Packages

Numa package é possível sequenciar passos de execução. Cada passo em uma package é passível de falha por qualquer razão (source ou target fora do ar, muitos registros rejeitados em uma interface, etc.).
É necessário sempre procurar prever os possíveis erros em cada passo da package.
As setas de "OK" (verdes) nas packages precisam existir e as setas "KO" (vermelhas) são o que tornam a sua package à prova de balas.



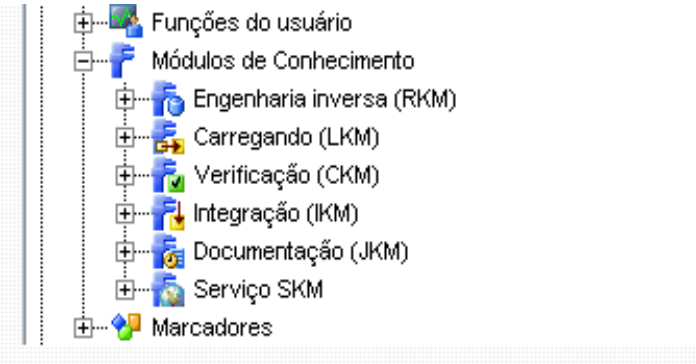
6 – Escolha o KM correto

A escolha do KM é crítica ao desenvolver uma interface, pois define quais as features estarão disponíveis e afeta também a performance de uma package.
Alguns erros comuns na escolha do KM:

- Começar com KM's complexos: Desenvolvedores iniciantes que ter suas interfaces rodando rapidamente mas às vezes não levam em conta todos os requisitos para utilizar um KM. Escolhendo, por exemplo, um LKM de uma tecnologia específica pode não funcionar por uma configuração ou instalação incorreta do loader. Uma ecolha mais segura seria começar usando KM's mais genéricos (como KM's de SQL) que funcionam na maioria dos casos. Depois de desenvolver suas primeiras interfaces com esses KM's é hora de mudar para KM's mais específicos (estude as especificações antes!).
- Interfaces com excesso de engenharia: KM's com features extras causam um custo extra de performance. Por exemplo, executar inserts é mais rápido do que executar updates incrementais. Se sua interface deleta os dados no destino antes da integração, utilizar update incremental é "excesso de engenharia" e causará perda de performance. Utilize o KM que se encaixa adequadamente à sua necessidade.
- De maneira similar, ativar ou desativar algumas fatures do KM pode adicionar passos extras degradando a performance. As opções default do KM são suficientes para executar o KM da forma como ele foi fornecido. Após executar o KM com as opções default é sempre bom revisar e checar se alguma opção pode ser alterada de acordo com a sua necessidade. A descrição do KM é sempre uma boa opção de documentação para entender e otimizar a utilização do KM.

Resumindo: Comece com os KM's mais simples, não se utilize de "excesso de engenharia" com KM's complexos ou ativando opções complexas e preste atenção às opções dos KM's.

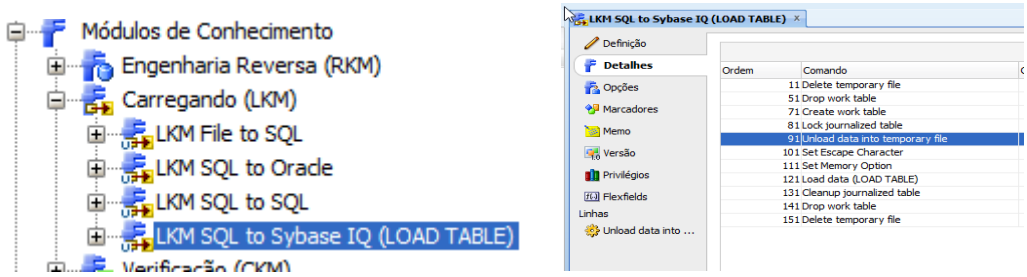
Para melhorar o entendimento vamos detalhar cada tipo de Módulo de Conhecimento (KM):

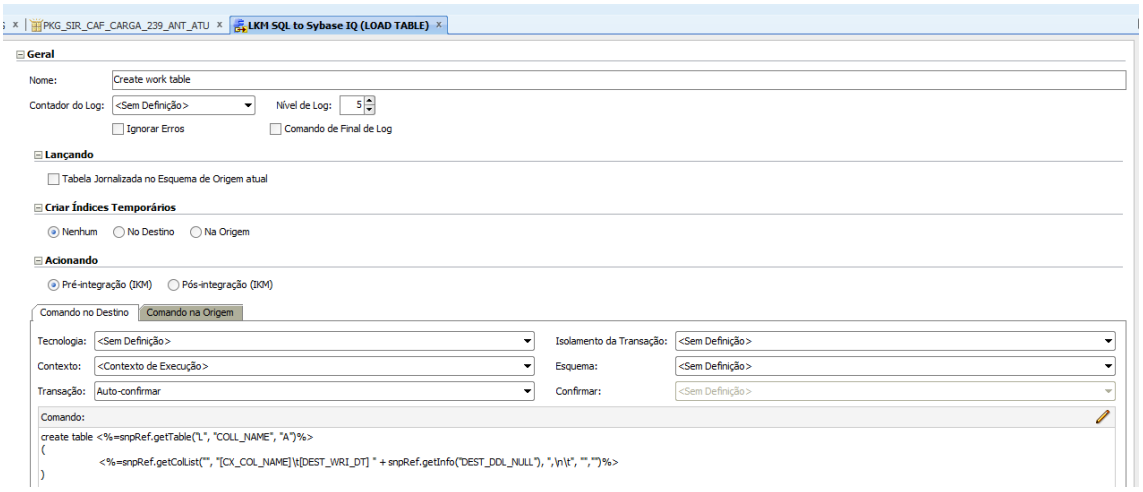


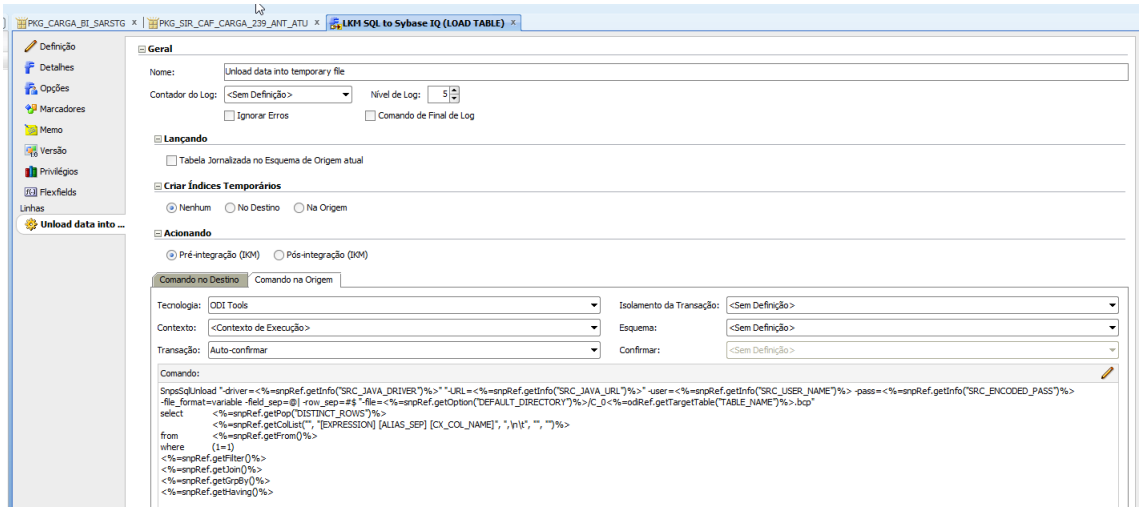
- ✓ **RKM** - Reverse Knowledge Module (Engenharia Reversa): é o responsável por fazer uma reversa "customizada" dos armazenamentos de dados no ODI. Por exemplo: se existir uma situação em que se necessite fazer algum tipo de procedimento extra ao reverter um modelo de dados, podemos utilizar RKMs específicos e não o padrão para esta tarefa. O ODI faz reversas de tabelas automaticamente, mas podemos customizar estas reversas com um RKM;
- ✓ **LKM** - Load Knowledge Module (Carga): é o responsável por carregar os dados das tabelas de origens no nosso processo de ETL quando estas tabelas se encontram em servidores de dados (Data Servers) diferentes;
- ✓ **CKM** - Check Knowledge Module (Verificação): é o responsável por realizar as validações dos dados no processo de ETL. No ODI podemos criar check constraints próprias contendo alguma regra de negócio (por exemplo, valor não pode ser negativo) ou podemos validar FKs de banco antes de inserir os dados na tabela de destino, ou ainda, durante o próprio processo de ETL, podemos verificar dados not null, etc. O CKM é o responsável por executar todas estas verificações;
- ✓ **IKM** - Integration Knowledge Module (Integração): é o responsável pela integração dos dados efetivamente no banco de destino. Ele resolve as regras do ETL descritas nas interfaces e insere os dados finais na tabela de destino;
- ✓ **JKM** - Journalizing Knowledge Module (Documentação): é o responsável por fazer a jornalização de dados quando se trabalha com este tipo de conceito. Pode ser usado, por exemplo, para se fazer replicação de bancos de dados;
- ✓ **SKM** - Service Knowledge Modules (Serviço): é utilizado para publicar dados utilizando Web Services. Pode ser utilizado para gerar e manipular dados via Web Services para arquiteturas SOA (Service Oriented Architecture – Arquitetura Orientada a Serviços);
- ✓ **Marcadores:** são utilizados para colocar marcadores nos objetos criados no ODI. Servem para a organização do projeto.

7 - Customize Seus KMs

Knowledge Modules (KMs) é um poderoso framework utilizado em cada ponto de integração no ODI. Um grande número de KM's está disponível para utilização. Eles suportam uma grande variedade de bancos de dados. Mesmo não sendo necessário na maioria dos casos, alguns projetos podem ter casos de uso ou requerimentos que solicitem uma customização de KM.
Então, qual deve ser o momento de customizar um KM? A resposta é: Tão logo seja detectada uma operação que precisa ser executada em várias interfaces, por exemplo, rodar um comando no target para otimização da execução.
Não é recomendado começar um KM à partir de uma folha em branco. O caminho recomendado é encontrar um KM que esteja o mais próximo possível do comportamento desejado e à partir dele, customizar de acordo com a necessidade.
Resumindo: Quando uma operação é necessária em muitas interfaces, não tenha medo de customizar um KM e criar o seu próprio KM baseado em algum já existente.



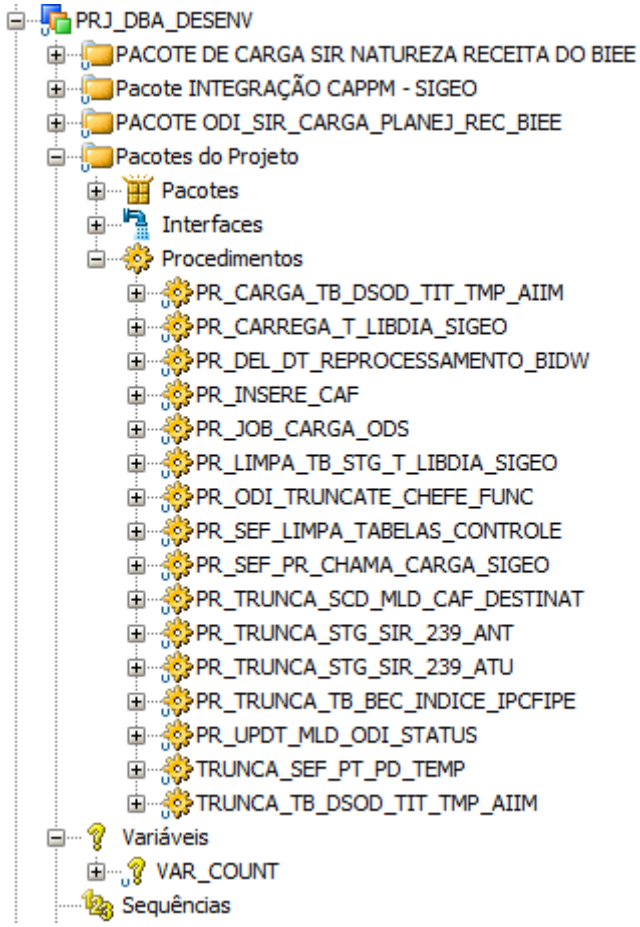


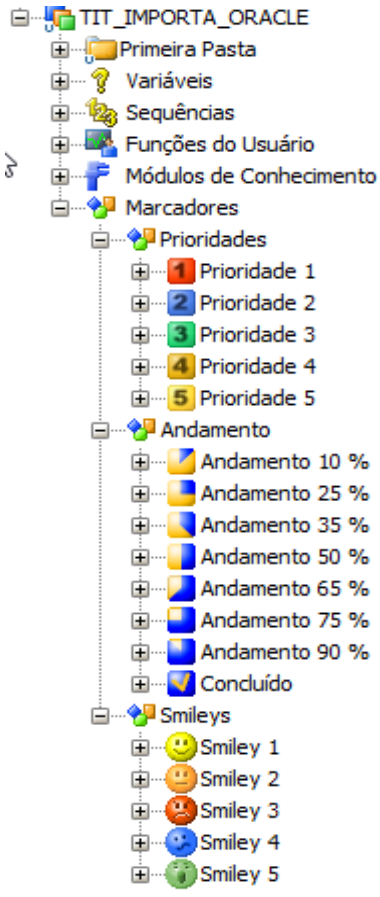


```
SnpsSqlUnload "-driver=<%=snpRef.getInfo("SRC_JAVA_DRIVER")%>" "-
URL=<%=snpRef.getInfo("SRC_JAVA_URL")%>" -user=<%=snpRef.getInfo("SRC_USER_NAME")%> -
pass=<%=snpRef.getInfo("SRC_ENCODED_PASS")%> -file_format=variable -field_sep=@| -row_sep=#$ "-
file=<%=snpRef.getOption("DEFAULT_DIRECTORY")%>/C_0<%=odiRef.getTargetTable("TABLE_NAME")%>.bcp
"
select      <%=snpRef.getPop("DISTINCT_ROWS")%>
            <%=snpRef.getColList("", "[EXPRESSION] [ALIAS_SEP] [CX_COL_NAME]", ",\n\t", "", "")%>
from        <%=snpRef.getFrom()%>
where       (1=1)
<%=snpRef.getFilter()%>
<%=snpRef.getJoin()%>
<%=snpRef.getGrpBy()%>
<%=snpRef.getHaving()%>
```

8 – Organize o Seu Projeto no Início

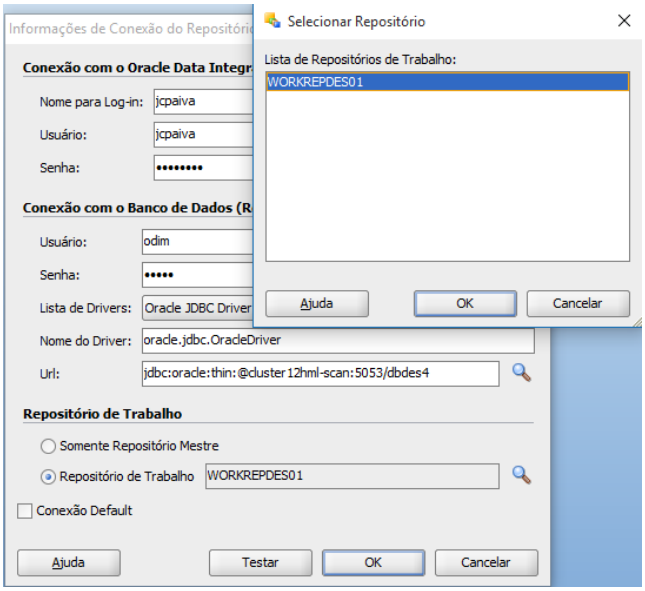
Gerenciamento e organização podem não parecer pontos críticos quando o assunto é integração de dados, mas são.
O ODI oferece muitas ferramentas que ajudam a organizar o desenvolvimento e o ciclo de vida do projeto: perfis de segurança, projetos de desenvolvimento, pastas, marcadores, versionamento, importação/exportação, impressão da documentação em PDF, etc.
Revise e utilize todas essas ferramentas e features para gerenciar o projeto. Defina a organização do projeto, a padronização de nomes e tudo o que pode evitar o caos depois que o projeto tiver iniciado. Faça isso desde o início do projeto.
Resumindo: Mantenha alta produtividade com o ODI, é melhor ter regras de organização baseadas nas features do ODI para evitar o desenvolvimento do caos.

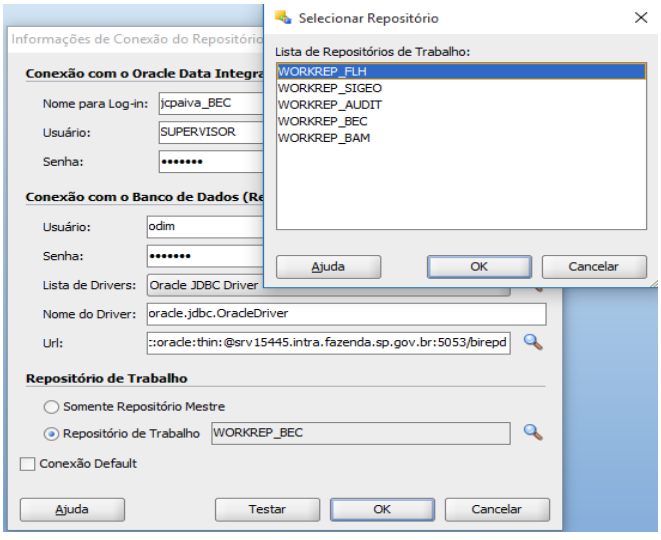




9 – Controle os Repositórios

No ODI, um repositório MASTER pode ter vários repositórios WORK. Também é possível ter vários repositórios MASTER, cada um com seu grupo de repositórios WORK. Cada repositório tem um ID definido na hora da criação do repositório.
Bem, um repositório não é um documento. É "a fonte da verdade", é a referência central entre os artefatos do ODI.
Além disso, todo objeto é identificado por um ID interno que depende do ID do repositório. Esse ID interno identifica unicamente um objeto e é utilizado pelo sistema de importação do ODI. Dois repositórios com o mesmo ID possivelmente contêm objetos com o mesmo ID interno, o que significa o mesmo objeto para o ODI. Transferir objetos entre esses repositórios é como copiar arquivos com o mesmo nome entre diretórios e pode fazer com que o objeto seja substituído.
Garanta que todos os repositórios sejam criados com ID's diferentes (mesmo em sites diferentes), e defina uma documentação para o processo de mover objetos entre repositórios utilizando import/export ou versionamento.
Resumindo: A multiplicação de repositórios deve ser feita sob estrito controle e planejamento (especialmente quanto à escolha do ID do repositório), e o gerenciamento de transferências de objetos utilizando import/export ou versionamento deve ser feito por vias formais.





10 – Cuidado Com o Conteúdo dos Repositórios

O ODI armazena todas as informações num repositório de metadados em um banco de dados relacional. Sabendo disso é muito tentador começar a explorar as tabelas do repositório para conseguir informações "mais rápido".
O repositório não implementa toda a lógica que existe na interface gráfica e também não implementa toda a lógica de negócios que existe no código Java. Construir, por exemplo, dashboards ou relatórios sobre o repositório é aceitável mas escrever dados ou alterar as informações do repositório é perigoso e deve ser deixado para operações do suporte técnico da Oracle.
Resumindo: Você faria isso no banco de dados do ERP da sua empresa? Também não faça com o repositório de metadados do ODI.

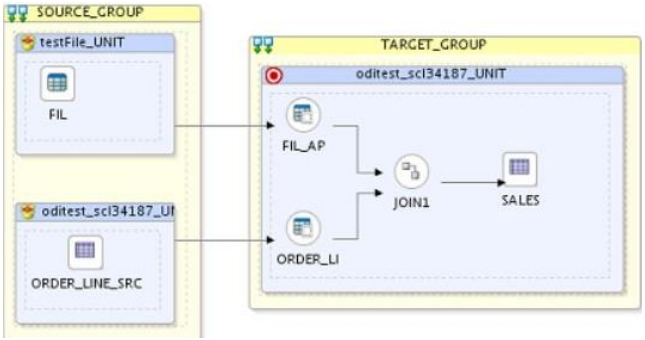
11 – Paralelismo e Controle de Sessões e Threads pelo ODI

O projeto que você desenvolver deve ser feito com olho na performance de sua infraestrutura.

Um tema a ser abordado é o **Paralelismo na Sessão de execução do ODI** e o outro é sobre o **Controle de Sessões e Threads pelo ODI**.

Com a nova versão do ODI 12c, a Oracle implementou uma camada de inteligência onde o **ODI** identifica automaticamente as sessões que podem ser executadas simultaneamente e gera o código pra sua **execução paralela**.

Por exemplo, na camada de ORIGEM (SOURCE) você precisa buscar dados de fontes totalmente distintas. Acontece que essas fontes irão alimentar o mesmo destino. **O que o ODI faz?** Ele junta as duas fontes (que estão em servidores diferentes) **na mesma sessão**, conforme imagem abaixo.

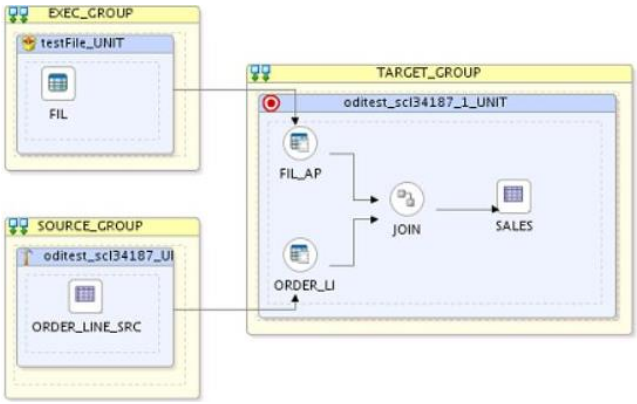


Desta forma, ele executa o processamento da carga fazendo **paralelismo**.

Bom, o que pra muitos isto pode parecer como algo muito bom já que tudo será executado ao mesmo tempo e pode parecer um processo otimizado, ***eu vejo como algo delicado.*** Por que o uso de paralelismo, **nessas condições onde as FONTES estão em lugares distintos**, é justamente um dos motivos que fazem com que sua performance caia.

Imagine se você tiver que fazer vários mapeamentos semelhantes ao que foi mostrado na imagem e além disso este mapeamento estiver em modo de execução automática (obedecendo alguma regra de negócio / schedule / cenário) que deve executá-lo de hora em hora. **O servidor vai ser onerado**.

Para solucionar isso, recomendo que (***como boa prática***), você separe as FONTES (Tabelas) para que cada uma tenha sua própria unidade na sessão, como na imagem abaixo.



Desta foram, o ODI irá entender o que você está querendo fazer. Ele irá executar os processos de forma sequencial analisando qual deve rodar primeiro. Assim, **seu processo ficará muito mais leve e rápido** sem degradar a performance de sua infraestrutura.

Além disso, **existe um ponto interessante a ser abordado** nesta questão do paralelismo que é o **"Gerenciamento das Conexões"**.

Por que estamos abordando que manter o paralelismo nos mapeamentos prejudica sua performance?

Cada mapeamento quando está em modo de execução paralela, assume que cada tarefa terá sua própria conexão a partir do Pool de Conexões. Por isso, se você tiver mapeamentos altamente paralelos, o tamanho do pool de conexões deve ser de acordo com o nível de paralelismo. Ou seja, **você irá ter mais de uma conexão** para executar o mesmo mapeamento **(carga de dados).**

Existem três pontos que podemos citar como candidatos ao Paralelismo (que você pode evitar) que são:

1 – Quando os dados de Origem são carregados na área de teste;

2 – Quando os dados estão em Origens distintas;
Sacada : Você pode criar uma Staging Area pra carregar os dados que precisa e executar a carga em uma estrutura só.

3 – Quando os dados precisam alimentar vários Destinos;

Acredito que adotando essas boas práticas você terá um excelente mapeamento e uma otimização de processamento muito satisfatória.

Conclusão

ODI é uma ferramenta de ETL com imenso potencial para ajudar no processo estratégico de carga de dados. No entanto, é preciso entender como utilizar e em qual situação aplicar os componentes disponíveis nessa ferramenta.
A combinação de recursos do banco de dados com os recursos do ODI para transferência ou transformação de dados pode trazer excelentes benefícios de performance em diversos cenários. Avalie o cenário existente e veja se a ODI é uma solução possível para o seu caso. Ao implementar a mesma, analise, teste, revise e siga as melhores práticas deste manual, de modo a obter a melhor performance, configuração e segurança do Oracle Data Integrator.