Fábricas de *Software* - Customização de *Software*

João Vilaverde

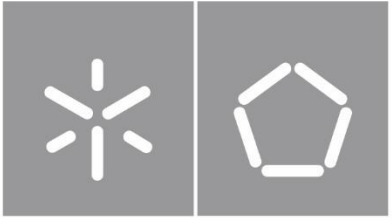
UMinho | 2021

Universidade do Minho
Escola de Engenharia

João Carlos Oliveira Vilaverde

Fábricas de *Software* - Customização de *Software*

janeiro de 2021



Universidade do Minho
Escola de Engenharia

João Carlos Oliveira Vilaverde

Fábricas de *Software* - Customização de *Software*

Dissertação de Mestrado
Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação do
Professor Doutor José Luís Mota Pereira

janeiro de 2021

Agradecimentos

Quero começar por agradecer aos meus pais e irmãos por todo o sacrifício que foi desde o primeiro dia em que entrei na universidade, até ao último dia. Foram incansáveis no apoio tanto financeiro como psicologico. Sempre me motivaram a terminar tudo quando tive momentos em que aquilo que mais queria era apenas fazer o suficiente e ir trabalhar porque queria retribuir toda a ajuda que sempre me deram.

Quero agradecer à minha namorada pela ajuda, pela disponibildiade de podermos conversar e trocar opiniões sobre alguns trabalhos quando precisava de mais uma validação nos documentos. Quero agradecer-lhe por me motivar todos os dias especialmente quando tinha coisas da universidade para tratar e não estava com disposição para fazer o que quer que fosse.

Quero agradecer aos meus colegas de curso, e amigos, Fábio, Ricardo e João, por termos criado o grupo que criamos e por termos levado esse grupo desde o 1º ano até ao ultimo ano do curso. Fomos um suporte uns dos outros, sempre dedicamos tudo o que tinhamos em prol do melhor rendimento dos nossos trabalhos. Foi incrivel ter a possibilidade de os conhecer e de fazer todos os trabalhos no curso com eles. O meu muito obrigado a eles pela amizade e pelo compromisso nos trabalhos!

Quero agradecer aos meus amigos por me terem ajudado sempre a descomprimir quando passava semanas complicadas, pelas futeboladas, pelo café que íamos tomar à noite para espairecer e falar um bocado de tudo menos do trabalho/universidade.

Quero agradecer à minha madrinha por sempre acreditar em mim e sentir o orgulho que sentiu quando tivemos a cerimónia de fim de curso. Sempre me incentivou a terminar este Mestrado em honra da família!

Quero agradecer aos meus avós, aos meus tios e tias, por todo o apoio que deram, por todo o miminho que recebia muitas vezes monetário que para mim sempre significou muito e me permitia poder aproveitar melhor os momentos, para além de ter de pagar todas as despesas que tinha.

Quero agradecer ao meu padrinho por ter sido sempre incansável no apoio. Sempre que precisei de alguma coisa nunca me disse que não. Ajudou-me imenso a tornar

possível poder entrar no curso que entrei no qual acabei por me formar durante estes últimos 7 anos.

Resta-me também agradecer ao meu orientador, professor doutor José Luís Mota Pereira, por tornar possível que eu pudesse terminar o curso e por toda a orientação que tem dado desde o início. O meu muito Obrigado!

Por fim, resta-me pedir desculpa a todos por todos os momentos que tive de abdicar de alguma coisa para poder dedicar-me a trabalhos ou a estudar para testes/exames.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

Este trabalho de dissertação ocorreu num momento em que havia sido concluído um estágio profissional de 9 meses de duração. Este estágio foi possivel através do Instituto do Emprego e Formação Profissional (IEFP), tendo contrato com a Aubay mas trabalhando como outsourcing para um cliente XPTO da Oracle. O trabalho correu no *Innovation and Technology Center* (ITC), em Leça do Balio, Porto. O ITC é considerado uma fábrica de *software*, isto é, é um conjunto estruturado de ativos relacionados com *software* que ajuda na produção de aplicações, ou componentes de *software*, de acordo com requisitos específicos dos clientes.

Como abordagem de gestão de projeto, para o cliente XPTO foi usada a abordagem Scrum utilizando metologias do AGILE. O projeto foi estruturado em *Program Increments* (PIs) divididos em *sprints* – PIs de 3 meses de duração cada, sendo que cada *sprint* tinha uma duração de 2 semanas – sendo que em todos os *sprints* havia sempre uma sessão de esclarecimento daquilo que seria para fazer e depois havia o tempo de desenvolvimento até ao momento da entrega no final do *sprint*. No final de cada PI, o cliente XPTO fazia, juntamente com a equipa de desenvolvimento, um balanço de todo o PI, identificando quantos bugs foram abertos, o que é que a equipa de desenvolvimento achava que poder-se-ia melhorar na abordagem de desenvolvimento do projeto, quais as dificuldades que estariam a enfrentar, etc.

A nível de tecnologia *Enterprise Resource Planning* (ERP), durante o estágio para o cliente XPTO da Oracle foi usado o Oracle *Retail Merchandising System* (RMS). Neste produto foram desenvolvidas, ao longo dos *sprints*, não só algumas costumizações de partes que já existiam, de forma a servir as necessidades do cliente, como também foram criadas um conjunto de APIs de forma a servir novas funcionalidades pretendidas pelo cliente. Este trabalho decorreu especificamente em módulos diferentes, tal como se poderá verificar nas tarefas que são demonstradas ao longo do trabalho de dissertação, tarefas essas que estão descritas e que apresentam nos anexos conteúdo, como por exemplo, algum código desenvolvido e ainda alguns testes efetuados que comprovam o bom funcionamento da funcionalidade imprementada.

Palavras-chave: AGILE, API, ERP, Fábrica de *Software,* RMS, SCRUM

Abstract

This dissertation work took place at a time when a 9-month professional internship was concluded. This internship was made possible by the Institute of Employment and Professional Training (IEFP), having a contract with Aubay but working as outsourcing to an XPTO client from Oracle. The work took place at the Innovation and Technology Center (ITC), in Leça do Balio, Porto. ITC is considered to be a software factory wich is a structured set of software-related assets that helps in the production of applications, or software components, according to specific customer requirements.

As a project management approach, for the XPTO client, the Scrum approach was used using AGILE methodologies. The project was structured into Program Increments (PIs) divided by sprints - 3 months in duration, each sprint lasting 2 weeks - and in all sprints there was always a session to clarify what it would be done and then there was development time until delivery at the end of the sprint. At the end of each PI, the XPTO client was made a balance of the entire PI, with the development team, identifying how many bugs were opened, what the development team thought could be improved in the development approach of the project, what difficulties they would be facing, etc.

In terms of Enterprise Resource Planning (ERP) technology, during the internship for Oracle's XPTO client, the Oracle Retail Merchandising System (RMS) was used. In this product, not only some customizations of parts that already exist to serve the customer's needs were developed, over the sprints, but also a set of new APIs were also created to serve the new features that the customer wanted. This work took place in different modules, as can be seen in the tasks that are demonstrated throughout the dissertation work, tasks that are described and, in the annexes, contain content, such as some developed code and some tests performed that prove the good functioning of the implemented functionality.

Keyword: *AGILE, API, ERP, RMS, SCRUM, Software Factory,*

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API – *Application Programming Interface*

BPR – *Business Process Reengineering*

CFA – *Custom Flex Attribute*

CRM – Customer Relationship Management

CSFs – *Critical success factors*

ERP – *Enterprise Resource Planning*

IEFP – Instituto do Emprego e Formação Profissional

ITC – *Innovation and Technology Center*

MRP – *Material Requirements Planning*

MRP II – *Manufacturing Resource Planning*

PIs – *Program Increments*

ReIM – *Retail Invoice Matching*

ReSA – *Retail Sales Audit*

RIB – *Retail Integration Bus*

RH – Recursos Humanos

RMS – *Retail Merchandising System*

RPM – *Retail Price Management*

RTM – *Retail Trade Management*

UDA – *User Defined Attribute*

1 - Introdução

1.1 – Enquadramento

Este trabalho encontra-se inserido no âmbito de um Estágio Profissional, através do Instituto do Emprego e Formação Profissional (IEFP), na empresa Aubay, que ocorreu entre o dia 1 de julho de 2019 e o dia 31 de março de 2020, tendo como cliente final XPTO, que por razões de confidencialidade será o nome utilizado ao longo do documento.

A Aubay é uma empresa multinacional Francesa, que se encontra em Portugal desde 2007, especializada em consultoria em Gestão, Implementação, Desenvolvimento e Manutenção de Sistemas de Informação. Atualmente tem no ativo mais de 130 parceiros e opera em setores como bancos, seguros, telecomunicações, serviços, energia e transportes. A empresa está sediada na cidade de Lisboa tendo ainda outro escritório na cidade do Porto. Atualmente esta tem uma parceria em vigor com a Oracle (outsourcing) fazendo com que a Aubay forneça funcionários à Oracle, sendo este o cliente onde foi realizado o estágio.

A Oracle é uma multinacional de tecnologia dos Estados Unidos, especializada no desenvolvimento e comercialização de hardware e *software*, em particular de bases de dados. Recentemente, a Oracle abriu um *Innovation and Technology Center* (ITC) em Portugal, mais concretamente em Leça do Balio, no Porto, tendo sido este o lugar onde ocorreu todo o estágio. O ITC funciona como uma Fábrica de *Software*, isto é, é um conjunto estruturado de ativos relacionados com *software* que ajuda na produção de aplicações, ou componentes de *software*, de acordo com requisitos específicos dos utilizadores finais. Para além disso, as fábricas de *software* aplicam princípios e técnicas para desenvolvimento de *software* de forma a imitar os benefícios de uma fábrica tradicional. Estas fábricas estão habitualmente relacionadas com criação de *software outsourced*. Cada fábrica de *software* é projetada para ajudar a criar aplicações que compartilham uma arquitetura e um conjunto de recursos, fornecendo ainda modelos e ferramentas para ajudar as equipas de desenvolvimento a iniciar rapidamente o desenvolvimento de novas aplicações. Estas fábricas ajudam os *developers* fornecendo-

lhes a orientação e automação das atividades de desenvolvimento prescritas ao longo do ciclo de vida do desenvolvimento de aplicações.

Uma característica fundamental de uma fábrica de *software* é que os arquitetos e os *developers* podem customizar, estender, e ajustar o ciclo de vida do desenvolvimento de forma a atender às necessidades exclusivas de uma equipa de projeto ou de uma organização. Normalmente, um arquiteto desenha a customização e, de seguida, redistribui para a fábrica de *software* que executará a customização e a irá devolver à equipa de projeto para entrega ao cliente.

1.2 – Objetivos a atingir

Como principais objetivos a atingir, este trabalho visa:

- Contacto e participação em projetos de desenvolvimento de *software* desde o desenho à implementação;
- Contacto e envolvimento com metodologias de gestão de projeto e desenvolvimento de *software*;
- Aquisição de competências na análise e desenho de requisitos de desenvolvimento de *software*;
- Aquisição de competências na implementação de sistemas ERP;
- Aplicar conhecimentos de linguagem PL/SQL, e ainda linguagem C, para o desenvolvimento da customização de *software*, usando o RMS, para o cliente XPTO.

1.3 – Estrutura do documento

Este documento está dividido em seis capítulos.

No primeiro capítulo faz-se um enquadramento do trabalho, incluindo uma breve apresentação do contexto do contexto em que decorreu, qual a empresa promotora do mesmo, qual o cliente onde se realizou o trabalho, e por fim uma breve definição do conceito de fábrica de *software* que se enquadra com aquilo que é o local onde se realizou o estágio.

O segundo capítulo intitulado de Referencial Teórico, sintetiza os conceitos gerais em torno dos ERPs (*Enterprise Resource Planning)*, servindo como introdução para os capítulos três e quatro. Neste segundo capítulo consta uma breve revisão histórica da origem dos ERPs, a definição propriamente dita de ERP, alguns benefícios e os fatores críticos de sucesso da sua implementação.

No terceiro capítulo, intitulado Abordagem de Desenvolvimento, apresenta-se a abordagem usada no decorrer de um dos projetos que serviu de base para todo o desenvolvimento do trabalho prático.

No quarto capítulo, intitulado Oracle RMS, é apresentado o sistema ERP desenvolvido pela Oracle na área do Retalho, desde a sua definição até à especificação dos módulos existentes no mesmo.

No quinto capítulo, intitulado Tarefas de Customização de *Software* para o Cliente XPTO, consta uma breve apresentação do cliente XPTO, para quem se destina o trabalho, bem como o detalhe de cada tarefa realizada no período compreendido entre janeiro 2019 e janeiro 2020.

No sexto capítulo, intitulado Conclusão, constam todas as reflexões e descrevem-se as dificuldades encontradas ao longo do período de desenvolvimento do trabalho.

Na sétima secção, irão constar as referências usadas para o desenvolvimento da parte teórica deste documento.

Por fim, o documento termina com um conjunto de anexos contendo o planeamento do trabalho, assim como todos os desenvolvimentos de cada Tarefa mencionada no quinto capítulo deste documento.

2 – Referencial Teórico

2.1 – Enquadramento histórico dos sistemas ERP

Historicamente, os conceitos primordiais de ERPs surgiram há cerca de 100 anos atrás, numa altura em que o papel era predominante no dia a dia das organizações. Em 1913, o engenheiro Ford Whitman Harris desenvolveu o modelo de quantidade económica de pedidos (EOQ), um sistema de fabricação em papel para programar a produção. Em 1964, a organização Toolmaker Black and Decker adotou uma solução de *Material Requirements Planning* (MRP) que consistia numa combinação entre aquilo que era conhecido como EOQ, com um *mainframe computer*. Este conceito de MRP permaneceu como padrão até um novo conceito ter aparecido em 1983, chamado *Manufacturing Resource Planning* (MRP II). O MRP II apresentou os "módulos" como um componente essencial da arquitetura de *software*, e componentes de manufatura integrados, incluindo compras, listas de materiais, programação e gestão de contratos. Com este novo conceito tornou-se possível integrar diferentes tarefas de manufatura num sistema comum. Para além disso, o MRP II forneceu uma nova visão de como as organizações poderiam aproveitar o *software* para partilhar e integrar dados corporativos e aumentar a eficiência operacional com melhor planeamento de produção, stock reduzido e menor desperdício. À medida que a tecnologia de computadores evoluiu ao longo do tempo, entre 1970 e 1980, foram aparecendo conceitos similares ao do MRP II para lidar com as atividades do negócio além das atividades da manufatura, incluindo finanças, *Customer Relationship Management* (CRM) e ainda Recursos Humanos (RH). Em 1990, os analistas da tecnologia da época tinham um nome para essa nova categoria de *software* de gestão de negócios, *Enterprise Resource Planning* (ERP).

2.2 – Considerações gerais sobre os sistemas ERP

Enterprise Resource Planning (ERP) é um tipo de *software* usado pelas organizações para gerir, dia a dia, aquilo que são as atividades do negócio, tais como contabilidade, gestão de projeto, gestão de risco, compras, entre outros. Para além disso, um sistema

ERP inclui também gestão de performance organizacional, *software* que ajuda a planear, orçamentar, prever e que permite obter *dashboards* dos resultados financeiros de uma organização.

Os ERPs são desenhados em torno de um modelo de dados bem definido que tipicamente tem em comum uma base de dados, assegurando assim que a informação usada na organização é normalizada e baseada em definições comuns e em utilizadores experientes. Ou seja, o ERP é um meio para integrar não só as pessoas, como também os processos e tecnologias numa organização moderna.

2.3 – Benefícios dos sistemas ERP

Com a adoção de ERP as organizações podem beneficiar do seguinte:

- Visão alargada dos negócios a partir de informações em tempo real que são geradas por relatórios;
- Reduz os custos operacionais através de processos de negócio otimizados e implementação de melhores práticas;
- Melhora a colaboração de utilizadores que compartilham dados em contratos, requisições e pedidos de compra;
- Maior eficiência através de uma experiência comum dos utilizadores em enumeras funções e processos de negócio bem definidos;
- Infraestrutura consistente desde o *back office* ao *front office*, com todas as atividades de negócio com a mesma aparência;
- Risco reduzido através da melhoria da integridade dos dados e controlo financeiro;
- Reduzidos custos operacionais e de gestão através de sistemas uniformes e integrados.

2.4 – Fatores Críticos de Sucesso

Baseado na tese de mestrado *Critical Sucess Factors* (CSFs*) in ERP Implementation*, de 2017, tendo como autores Jonathan Arvidsson & Daniel Kojic, os fatores críticos de sucesso foram definidos examinando a frequência com que foram referidos após uma

revisão de literatura de vários outros autores. Assim sendo, foram categorizados 14 fatores críticos de sucesso, sendo eles:

2.4.1 – *Top Management Commitment*

Este conceito refere-se à necessidade de se ter uma liderança comprometida com o projeto de implementação e utilização do ERP por parte da gestão de topo. Uma implementação bem-sucedida de um ERP depende e muito do constante envolvimento da gestão de topo, sendo crucial que a gestão de topo dê suporte nos mais variados níveis da organização.

2.4.2 – *Implementation Strategy*

Este conceito é dos mais importantes para uma implementação bem-sucedida de um ERP. É importante que sejam feitas escolhas para se formar uma estratégia de implementação que funcione bem. Perguntas como: quais são as informações especificas necessárias no nível operacional e de gestão; como é que o ERP irá integrar com sistemas existentes e qual é o plano para a implementação. Respondendo a esse conjunto de questões será possível desenvolver um plano que aumenta as possibilidades de sucesso comparando com as organizações sem plano.

2.4.3 – *Communication*

Não conseguir uma comunicação aberta e fluente entre a gestão de topo e o utilizador do sistema é uma das principais causas da falha na implementação do ERP. Uma organização que incentiva os seus funcionários a participar ativamente na implementação é mais bem-sucedida.

2.4.4 – *Training and Education*

Planear programas de *training and education,* ou seja, formação no futuro sistema e processos definidos, são algumas das variáveis chave no planeamento de novos sistemas

que em conjunto com outras variáveis se tornam num ingrediente importante para uma mudança bem-sucedida na implementação dos sistemas ERP, bem como para as atualizações em curso relacionadas à mudança.

2.4.5 – *Implementation Team*

A necessidade de uma equipa principal competente de funcionários dedicados e capazes, especialmente no início da implementação é também importante para uma implementação funcionar sem problemas. Essa equipa deverá ser capaz de liderar o caminho, usando o talento para explorar detalhes ao executar a fase de planeamento.

2.4.6 – *Change Management*

A estrutura organizacional preexistente e os processos encontrados na maioria das organizações tipicamente não são compatíveis com a nova estrutura, ferramentas e informações fornecidas pelos sistemas ERP. Assim para lidar com este facto, uma organização pode forçar a reengenharia dos principais processos de negócios ou desenvolver novos processos de negócio para apoiar os objetivos da organização. Consequentemente a isto, os utilizadores do sistema podem resistir à mudança para o novo sistema. Daí ser crucial dedicar especial atenção a esta área de gestão da mudança de forma a assegurar que toda a organização acompanhe, adote e aceite os novos processos e ferramentas.

2.4.7 – *User Involvement*

Além do fator descrito acima, *training and education*, é importante não só envolver os utilizadores durante o desenvolvimento do sistema, como também obter conhecimento existente em áreas onde a equipa de implementação não possui experiência suficiente. Posto isto, o envolvimento e participação de utilizadores finais resultarão num melhor ajuste dos requisitos e também na melhoria da qualidade, do uso e da aceitação do novo sistema.

2.4.8 – *Business Process Reengineering (BPR)*

Um projeto ERP incentiva as organizações a reverem os seus processos de negócio e a descobrir maneiras de fazer as coisas em relação às melhores práticas já incorporadas no sistema. BPR é o repensar fundamental e o redesenho radical dos processos de negócio para alcançar melhorias dramáticas nas medidas de desempenho, como custo, qualidade, serviço e velocidade. BPR também envolve o alinhamento dos negócios com o novo sistema: adoção de processos, novos processos *standard*, novos processos de negócio e *redesign* de tarefas. Há considerações que se devem ter em conta durante esta etapa, como o aprimorar da qualidade da interface do ERP, bem como a necessidade de planear a infraestrutura da tecnologia. Um problema do *packaged software*, *software* pré-definido com processos e práticas embebidas, é o risco de conflito entre as necessidades da organização e os processos de negócio pré-existentes. Posto isto, é importante identificar os processos de negócio principais antes da reengenharia por forma a que estes sejam analisados com mais enfase e que a sua substituição ou manutenção seja devidamente ponderada.

2.4.9 – *Use of Consultants*

Os consultores são usados para facilitar o processo de implementação, pois podem estar familiarizados com indústrias especificas, possuir conhecimentos sobre os módulos e ter competências adequadas para determinar qual será o conjunto de *software* mais indicado para uma determinada organização. As organizações tendem a usar consultores externos quando configuram, instalam e personalizam o seu *software*, sendo que esse uso de partes externas desempenha um papel essencial no início do projeto, mas diminui durante as últimas etapas da implementação quando o sistema está em execução. Além disso, um apoio adequado por parte dos fornecedores e a participação dos consultores externos, mesmo após o término da implementação, são um dos fatores subjacentes ao sucesso dos projetos ERP.

2.4.10 – *Project Support*

Uma das formas de fornecer este fator crítico de sucesso é através de outro fator crítico de sucesso, já descrito acima, que é o *training and education*. No entanto, caso surjam problemas e for necessária ajuda externa, deverá haver um sistema de suporte pronto para ser acionado. Este sistema de suporte não deve estar ativo apenas na eventualidade de ocorrerem problemas. Deve estar sempre ativo. A principal função do suporte ao projeto é a manutenção e até a evolução do sistema. Este recurso é normalmente ativado na última fase do projeto de implementação, a etapa das operações. A manutenção é então vital para que a implementação seja bem-sucedida. É nessa etapa que são instaladas atualizações contínuas, onde há formação adicional que pode ser necessário e é realizada uma procura constante de problemas para que a organização preveja futuras áreas problemáticas. Quantas mais personalizações forem feitas num ERP maior é o risco de precisar de ter um sistema de suporte mais pesado, que é caro e, a longo prazo, se torna altamente ineficiente.

2.4.11 – *ERP Selection*

Selecionar o Sistema ERP correto desde o início é vital para o sucesso de um projeto de implementação. Para que uma organização possa escolher o sistema ERP correto, as fases de implementação precisam de ser cuidadosamente trabalhadas. O sistema ERP deve corresponder aos processos de negócio da organização para que seja implementado com sucesso. O ajuste entre o sistema de escolha do ERP e a organização é fundamental para o sucesso do processo. Uma escolha errada de um sistema ERP que não se enquadre nos processos de negócio levará a organização a lutar para se adaptar, gastar recursos preciosos e, por fim, ter de abandonar a conversão como um todo e tentar novamente. Sendo este um processo muito caro e demorado, a organização não deve considerar a escolha de um sistema ERP de forma descuidada. Quando se fala em escolher um sistema ERP, está também associado escolher o fornecedor certo para implementar o sistema. A organização deverá basear a sua decisão do fornecedor em diversos fatores, sendo eles, características do fornecedor (ou seja, reputação), o relacionamento entre a organização e o fornecedor, o suporte que o fornecedor pode oferecer e quais ferramentas o fornecedor possui em seu nome.

2.4.12 – *Project Management*

A abordagem da gestão de projetos sugere que o planeamento e o controlo do projeto estão em correlação com as características do projeto, tais como o tamanho do projeto, as experiências com tecnologia e a estrutura do projeto. Quando a equipa do projeto é formalmente estabelecida, posteriormente, a equipa deve ser definida em termos dos seus *milestones*. Isso inclui determinar caminhos críticos do projeto, decidir a durabilidade do projeto e o esforço necessário à tomada de decisão. É preciso ter em conta que um objetivo de projeto muito amplo e ambicioso pode causar imensos problemas. Assim sendo, o âmbito do projeto deve ser claramente definido e limitado.

2.4.13 – *Quality Management*

A gestão da qualidade refere-se à integração de dados anteriores e a precisão que garante que a qualidade dos dados atenda aos requisitos do novo sistema. Além disso, grande parte do sucesso do processo de implementação e, em última instância, do sucesso total do sistema, depende da capacidade da equipa em garantir a precisão dos dados ao convertê-los no novo sistema. O processo de conversão da implementação pode envolver uma limpeza de dados considerados suspeitos que não serão necessários no novo sistema ERP. Esse mesmo processo deve ser pensado de forma a que haja uma menor probabilidade de perda de dados durante a migração. Essa parte da implementação também envolve garantir a confidencialidade, a integridade, a estabilidade e a compatibilidade do *software*. Deve-se avaliar se o sistema ERP é percebido como complexo ou não pelos utilizadores e se o ERP se encaixa nos processos de negócio da organização. Antes de o *software* entrar em *live*, ou seja, ser usado oficialmente pelo cliente, a organização deve executar uma variedade de testes para que o sistema seja estável, detetando falhas e erros antes da introdução. Envolver os utilizadores nos testes do sistema é vital, para que o utilizador possa verificar a robustez do sistema, além de controlar se ele funciona corretamente no ambiente operacional.

2.4.14 – *Risk Management*

A Gestão de Riscos envolve o desenvolvimento de ferramentas apropriadas à resolução de problemas, habilidades e técnicas adequadas e, em relação ao fator critico de sucesso, já descrito anteriormente, *Use of Consultants,* trabalhar em estreita colaboração com fornecedores e consultores quando alguma coisa estiver errada no sistema. A resolução de problemas é um fator crítico ao implementar um ERP e o relacionamento com fornecedores e consultores para resolver problemas de *software* deverá funcionar bem. Existem dois fatores de risco que influenciam os resultados desfavoráveis do projeto:

- Externo – por exemplo, modelos de negócio e participantes;
- Interno – por exemplo, tamanho, duração, complexidade e *outsourcing* do projeto;

Esses aspetos representam um risco para o projeto e são importantes para que a gestão de topo do projeto considere, na implementação, desenvolver um plano de gestão de risco apropriado para prever falhas no projeto.

A gestão de risco pode ser dividida em diversas fases, sendo elas:

- Análise de contexto;
- Identificação de Riscos;
- Análise de Riscos;
- Avaliação de Riscos;
- Tratamentos de Riscos;
- Monitorização e Revisão;
- Comunicação e Consultoria;

Apesar de haver uma estratégia de gestão de risco bem definida, os utilizadores devem ter conhecimento sobre qualquer plano de contingência se ocorrerem erros. Se eles não tiverem conhecimento sobre isso, poderão depender excessivamente do fornecedor para resolver consultas técnicas. Além disso, se o utilizador não tiver a educação e conhecimento apropriados aos quais está exposto, será difícil fornecer detalhes precisos das deficiências e levar a atrasos dos fornecedores para resolver o problema.

Para além disso, os autores disponibilizam uma tabela, tabela 1, onde é possível ver os Fatores Críticos de Sucesso que foram por eles propostos e os Fatores Críticos de Sucesso Originais que foram encontrados ao longo de uma revisão de literatura de vários autores.

Proposed CSFs	***Original CSFs***
Top management commitment	*Top management commitment and support* (Finney & Corbett, 2007), *top management involvement* (Moon, 2007), *top management support and commitment* (Dezdar & Suleiman, 2009), *support of top management* (Shaul & Tauber, 2013), *top management support and commitment* (Saade & Nijher, 2015)
Implementation strategy	*Implementation strategy and timeframe* (Finney & Corbett, 2007), *software implementation process* (Moon, 2007), *contingency plans* (Saade & Nijher, 2015), *implementation strategy* (Shaul & Tauber, 2013)
Communication	*Communication plan* (Finney & Corbett, 2007), *communication* (Moon, 2007), *enterprise-wide communication and cooperation* (Dezdar & Suleiman, 2009), *open and transparent communication* (Saade & Nijher, 2015)
Training and education	*User training and education* (Dezdar & Suleiman, 2009), *education and training* (Moon, 2007), *education and training* (Shaul & Tauber, 2013)
Implementation team	*Balanced team* (Finney & Corbett, 2007), *ERP team composition, competence and compensation* (Dezdar & Suleiman, 2009), *project team competence* (Shaul & Tauber, 2013), *project champion* (Finney & Corbett, 2007), *empowered decision makers* (Finney & Corbett, 2007), *small internal team of best employees* (Saade & Nijher, 2015)
Change management	*Managing cultural change* (Finney & Corbett, 2007), *stable and successful business setting* (Moon, 2007), *organisational change management* (Moon, 2007), *change management programme* (Dezdar & Suleiman, 2009), *organisational experience of major*

	change (Shaul & Tauber, 2013), *cultural change readiness* (Saade & Nijher, 2015)
User involvement	*User feedback usage* (Saade & Nijher, 2015), *user involvement* (Dezdar & Suleiman, 2009), *user involvement* (Shaul & Tauber, 2013)
Business process reengineering	*BPR and software configuration* (Finney & Corbett, 2007), *business process alignment* (Moon, 2007), *BPR and minimum customization* (Dezdar & Suleiman, 2009), *BPR* (Saade & Nijher, 2015)
Use of consultants	*Consultant selection and relationship* (Finney & Corbett, 2007), *legacy systems support* (Saade & Nijher, 2015), *use of consultant* (Dezdar & Suleiman, 2009)
Project support	*Project support* (Moon, 2007), *vendor support* (Dezdar & Suleiman, 2009), *local vendor* partnership (Saade & Nijher, 2015)
ERP selection	*Selection of ERP* (Finney & Corbett, 2007), *careful selection of ERP software* (Dezdar & Suleiman, 2009), *enterprise system selection process* (Shaul & Tauber, 2013), *ERP fit with the organisation* (Saade & Nijher, 2015)
Project management	*Project management and evaluation* (Dezdar & Suleiman, 2009), *project tracking* (Shaul & Tauber, 2013), *project management* (Moon, 2007), *project management* (Finney & Corbett, 2007)
Quality management	*System testing* (Finney & Corbett, 2007), *system quality* (Dezdar & Suleiman, 2009), *quality management* (Saade & Nijher, 2015)
Risk management	*Troubleshooting and crisis management* (Finney & Corbett, 2007), *software analysis, testing and troubleshooting* (Dezdar & Suleiman, 2009), *software* maintenance (Shaul & Tauber, 2013), *risk management* (Saade & Nijher, 2015)

Tabela 1 – Fatores Críticos de Sucesso

3 – Abordagem de Desenvolvimento

A abordagem de desenvolvimento escolhida pelo ITC, no âmbito do projeto em que ocorreu o estágio, foi a Scrum. Esta abordagem oferece um conjunto de vantagens que o ITC considerou serem adequadas para o projeto em questão, vantagens essas como a melhoria contínua com entregas semanais, sprint a sprint, sendo que há tempo dedicado à correção de problemas de entregas anteriores, como também ao desenvolvimento de novas funcionalidades-, o feedback contínuo, dado por reuniões diárias em que cada um dos elementos transmite à equipa como está o estado do seu trabalho, quais os seus bloqueios de forma a se poder ter conhecimento alargado do estado das tarefas e se poder partilhar soluções quando surgem problemas.

3.1 – Metodologia Scrum

Scrum é uma *framework* ágil de desenvolvimento de produtos, que apareceu em 1993 quando Jeff Sutherland criou o processo Scrum. Jeff atribuiu o nome "Scrum" depois de ver uma analogia num estudo de 1986 de Takeuchi e Nonaka que foi publicado na *Harvard Business Review*. Nesse estudo os autores comprovaram o alto desempenho de equipas multifuncionais com a formação Scrum usada por equipas de Rugby. Por se tratar de uma abordagem "ágil", esta torna-se flexível de forma a que os produtos sejam desenvolvidos e melhorados de forma iterativa e incremental. Para além disso, permite que as equipas de trabalho partilhem o mesmo espaço físico de forma a manter uma comunicação diária entre todos os elementos.

Desta metodologia, como princípio-chave tem-se que os clientes podem mudar de ideias, isto é, os clientes podem alterar as suas ideias sobre o que querem, como querem e até mesmo quando querem. Sendo essas mudanças possíveis significa que os requisitos não podem ser facilmente abordados de forma planeada, ou seja, o objetivo passa por tentar maximizar a capacidade da equipa para entregar rapidamente.

A metodologia Scrum é composta por três papéis: *Scrum Master*, *Product Owner* e equipas de desenvolvimento.

3.1.1 – *Scrum Master*

Scrum Master é o responsável por gerir todo o processo Scrum mas não a equipa de desenvolvimento. O *Scrum Master* é o líder, o responsável por remover qualquer barreira e impedimento que possa surgir durante o projeto e também, quando necessário, ajudará outras partes da organização, sem ser a equipa de desenvolvimento, a entender as práticas do Agile. O *Scrum Master* não lidera as equipas de desenvolvimento porque é da responsabilidade desta serem auto-organizadas o suficiente para fornecer o que está a ser exigido pelo negócio, fazendo com que o *Scrum Master* seja apenas responsável por orientar a equipa.

Ademar Albertin, no seu artigo online *Agile Principles & Culture*, enumera um conjunto de *características* que são importantes para se ser um bom *Scrum Master*, sendo elas:

- ***Service Level Management*** – Deve ser capaz de perceber como orientar a equipa de desenvolvimento de forma a alcançar os objetivos que vão sendo definidos durante o planeamento do *Sprint*;

- ***Architecture Design*** – A equipa de desenvolvimento deve ser orientada, de forma a obter o conhecimento necessário sobre determinadas componentes, para desenvolver soluções que respondam às necessidades do cliente e do negócio;

- ***Component Integration*** – Segue o mesmo princípio de que o Design da Arquitetura, ou seja é necessário ter um conhecimento mais abrangente sobre as componentes a serem desenvolvidas de forma a fornecer os melhores *inputs* às equipas de desenvolvimento;

- ***Testing*** – É muito frequente nas abordagens Ágeis, os produtos e serviços serem entregues com qualidade e eficiência, o que indica que todas as componentes previamente alteradas ou construídas de raiz sejam completamente testadas antes da entrega ao cliente. O *Scrum Master* pode

orientar a equipa de desenvolvimento a saber que tipo de teste deverá ser efetuado, e como fazê-lo, para determinada componente;

- ***Education and Training Provision*** – É responsável por treinar e orientar a equipa de desenvolvimento tendo por base as práticas do Agile, sendo possível orientar outras partes da organização. Esta *skill* permitirá ao *Scrum Master* operar num nível superior, podendo não só dar *training*, como também educar os membros da equipa com as práticas Agile dentro da organização;

- ***Personnel Development*** – Alguns recursos das equipas de desenvolvimento podem exigir mais atenção, isto porque alguns podem ser novos, e é da responsabilidade do *Scrum Master* que estes sejam treinados e desenvolvidos para acompanhar o desenvolvimento da forma mais apropriada;

- ***Project and Portfolio Management*** – Deverá ter *skills* de gestão de projetos e portfólio para facilitar o entendimento das equipas sobre como gerir as atividades necessárias para concluir os *Sprints*, ou melhor dizendo, para concluir as diferentes fases do projeto;

- ***Risk Management*** – Durante o desenrolar do projeto, são tomadas várias decisões incluindo como é que a equipa de desenvolvimento deve criar uma determinada componente. O *Scrum Master* precisa de entender os riscos de se desenvolver uma componente de determinada forma, em vez de outra. Quando situações destas acontecem, é necessária uma boa análise do risco para definir a maneira mais apropriada de desenvolver a componente;

- ***Process Improvement*** – a própria metodologia *Agile* já contempla melhoria contínua, incluindo o processo. Através das revisões no final da entrega, o *Scrum Master* pode orientar a equipa a encontrar melhorias que ajudarão a facilitar o próximo desenvolvimento;

3.1.2 – *Product Owner*

Cada equipa deve ter um *Product Owner* para fornecer visão e os requisitos de negócios do serviço e/ou produto. O *Product Owner* irá preencher a ponte entre as equipas comercias e técnicas ou, noutras palavras, as equipas Scrum. Existe um, e só um, *Product Owner* do produto para cada produto ou serviço. Ele representará a voz do cliente e ajudará a maximizar o valor do projeto, produto ou serviço. O papel do *Product Owner* não requer necessariamente conhecimento técnico, mas deverá ser competente em várias outras áreas, como por exemplo:

- ***Service Level Management*** – a gestão de datas-alvo e a compreensão dos melhores métodos de abordagem ajudarão o *Product Owner*. Toda a equipa é responsável por fornecer os recursos de acordo com a data prevista. Mas é o *Product Owner* que realizará as negociações com os clientes;

- ***Product or Service Planning*** – A gestão do *backlog* do produto[1] é uma habilidade básica que deve ser desenvolvida para quem deseja tornar-se um *Product Owner*. O *Scrum Master* e a Equipa de Desenvolvimento poderão avaliar e participar no planeamento, mas a responsabilidade de manter o *backlog* do produto ainda pertence ao *Product Owner*.

- ***Architecture Design and Application Design*** – Quando uma pessoa de negócios possui pelo menos um conhecimento técnico vago, torna-se um ajuda a definir melhores requisitos e metas.

- ***Needs Identification*** – O *Product Owner* será responsável por entender os requisitos do produto e/ou serviço, bem como a tradução dessas necessidades para a equipa de Desenvolvimento e para o *Scrum Master*. Um bom *Product Owner* definitivamente precisa de saber como identificar as necessidades da organização.

[1] Backlog do Produto – conjunto de requisitos ou funcionalidades ainda por desenvolver e implementar.

- ***Project and Portfolio management*** – No *Agile* Scrum não há a função Gestor de Projeto, mas o *Product Owner* deve gerir o *backlog* do produto.

- ***Relationship Management*** – No *Agile* Scrum, não há *Line Manager*, porque as equipas são auto-organizadas. Isso significa que a gestão e o estabelecer de um relacionamento são muito importantes para equilibrar melhor a maneira como as atribuições são dadas e também gerir o relacionamento com os negócios e os clientes.

- ***Business Change management*** – Alterações no projeto são sempre bem-vindas nas práticas do *Agile* Scrum. Porém, mais do que isso, às vezes os negócios podem mudar não apenas o produto e/ou serviço, mas também o próprio negócio e os seus *stakeholders*, e o *Product Owner* deve estar pronto para gerir e realizar as alterações necessárias.

Um *Product Owner* possui como responsabilidades:

- Desenvolver a direção e estratégia para os produtos e serviços, incluindo as metas de curto e longo prazo;
- Fornecer ou ter acesso ao conhecimento sobre o produto ou serviço;
- Compreender e explicar as necessidades do cliente para a equipa de desenvolvimento;
- Reunir, dar prioridade e gerir os requisitos de produto ou serviço;
- Assumir quaisquer responsabilidades relacionadas ao orçamento de produtos ou serviços, incluindo a sua lucratividade;
- Determinar a data de lançamento dos recursos do produto ou serviço;
- Trabalhar em conjunto com a equipa de desenvolvimento diariamente para responder a perguntas e tomar decisões;
- Aceitar ou rejeitar os recursos completos relacionados aos Sprints;
- Mostrar as principais realizações da equipa de desenvolvimento no final de cada Sprint;
- Responsável pelo *backlog* do produto;

3.1.3 – Equipas de desenvolvimento

A equipa de desenvolvimento é formada por 3 a 9 pessoas que devem satisfazer todas as necessidades técnicas para entregar o produto ou serviço. Eles serão guiados diretamente pelo *Scrum Master*, mas não serão geridos diretamente. Eles devem ser auto-organizados, versáteis e responsáveis o suficiente para concluir todas as tarefas necessárias.

Geralmente, quando uma nova equipa começa a trabalhar, leva tempo até que tenham afinidade suficiente para combinar os seus conhecimentos. Isso é completamente normal, e eles enfrentarão uma curva de aprendizado que será facilitada diariamente pelo *Scrum Master*. Não é recomendável alterar as componentes da equipa com muita frequência, a fim de evitar qualquer tipo de redução de produtividade devido a alterações pessoais.

As equipas de desenvolvimento possuem como responsabilidades:

- Criar e entregar os produtos ou serviços;
- Ser auto-organizado e ser capaz de se gerir. As equipas devem poder determinar as suas próprias tarefas e como elas serão executadas;
- As equipas não são combinadas com uma única *skill*, mas sim com várias e diferentes;
- São dedicados a um produto ou serviço;
- Recomendação para trabalhar no mesmo espaço de trabalho;

3.2 – Manifesto Agile

O Manifesto Agile foi um documento criado em 2001 por um conjunto de 17 pessoas onde estabeleceram os valores e princípios fundamentais para as metodologias ágeis. Assim sendo, como valores do manifesto Agile temos:

- Os indivíduos e as suas interações são mais importantes que os procedimentos e ferramentas;
- *Software* a funcionar é mais importante que ter a documentação completa;
- A colaboração com o cliente é mais importante que a negociação de contratos;

- A capacidade de responder a alterações é mais importante que seguir um plano.

Com estes valores a ideia não era retirar a importância dos procedimentos, ferramentas, documentação, contratos, planos, etc. A ideia passava apenas por estabelecer prioridades de forma a que nos projetos existisse mais flexibilidade.

Para além dos valores, o manifesto Agile estabeleceu 12 princípios, sendo eles:

- Garantir a satisfação do cliente, através da entrega rápida e continua de *software* funcional;
- Até mesmo as mudanças tardias ao âmbito do projeto são bem-vindas;
- *Software* funcional é entregue frequentemente;
- Cooperação constante entre as pessoas que entendem do negócio e os *developers*;
- Os projetos devem ser criados em torno de indivíduos motivados. Dê-lhes o ambiente e o apoio que necessitam, e confie-os para começar o trabalho feito;
- O método mais eficiente e eficaz de transmitir informações para e dentro de uma equipa de desenvolvimento é a conversa cara a cara;
- *Software* funcional é a principal medida de progresso do projeto;
- Os processos ágeis promovem o desenvolvimento sustentável. Os *sponsors*, *developers* e utilizadores devem ser capazes de manter um ritmo constante indefinidamente;
- A atenção contínua à excelência técnica e ao bom design aumenta a agilidade;
- Simplicidade – a arte de maximizar a quantidade de trabalho não feito – é essencial;
- As melhores arquiteturas, requisitos e projetos emergem de equipas auto-organizadas;
- Em intervalos regulares, a equipa reflete sobre como tornar-se mais eficaz, ajustando o seu comportamento em conformidade.

3.3 – Agile Sprint

Agile Sprint é um termo que faz referência ao ciclo de trabalho, de duração fixa, onde uma equipa de trabalho entrega componentes de um determinado produto. Cada *sprint* inclui uma pequena sessão de planeamento seguida de um período de desenvolvimento do produto. Os *sprints* terminam com uma sessão de retrospetiva para identificar os ajustamentos necessários ao produto e processo. Em termos de *release*, um ou mais *sprints* podem ser combinados de forma a gerar-se um *package* de instalação. Relativamente à duração, cada um dos *sprints* pode ter duração entre uma a quatro semanas.

Vários projetos podem adotar abordagens diferentes no que diz respeito à metodologia Scrum. Para isso, os responsáveis podem adotar a abordagem "Scope driven", onde o ciclo se repete até que todos os elementos desejados sejam implementados, sendo que nestes casos o número de sprints estimados para completar o projeto pode variar. Além dessa, os responsáveis podem adotar uma abordagem *schedule driven*, isto é, completam-se todos os elementos que sejam possíveis num número pré-definido de ciclos, sendo que nestes casos o número de *sprints* é fixo desde o início.

3.4 – *Kanban Board*

O *Kanban Board* é uma ferramenta de organização do trabalho em curso. Este trabalho pode ser representado por cartões*, post its*, etc. Cada um desses objetos irá representar uma componente do trabalho sendo depois organizados em listas que representam o processo de desenvolvimento do produto. Um exemplo dessas listas é a utilização dos estados "Não iniciado", "Em progresso" e "Concluído". Na metodologia Scrum, um exemplo de listas para o Kanban poderá ser "Backlog", "Em Desenvolvimento", "Concluído", "Aceite" e "Entregue ao Utilizador". A equipa de trabalho, à medida que vai progredindo, vai movendo as componentes em que se encontra a trabalhar para os devidos lugares que espelham o desenvolvimento dessa componente. Habitualmente os *Kanban Boards* são usados para organizar diversas áreas de uma organização, tais como, desenvolvimento de *software*, equipas de vendas, equipas comerciais e ainda

gestão de tarefas pessoais. Para além disso, os *Kanban Boards* ajudam a manter o controlo sobre o projeto e aumentam a produtividade, sendo que esta ferramenta é, não só capaz de criar um sentido de *ownership* sobre o projeto, como também permite a visibilidade sobre o trabalho desenvolvido por cada elemento da equipa de trabalho ao longo dos *Sprints*.

4 – Oracle Retail Merchandising System (RMS)

4.1 – Contexto Teórico

Como explicado no capítulo introdutório deste relatório o trabalho de estágio foi realizado para e nas instalações da Oracle que é uma multinacional tecnológica possuindo diversas soluções e produtos endereçando diversas áreas de mercado. Com efeito, e mais concretamente, o trabalho foi realizado para uma divisão concreta da Oracle, a RGBU – Retail global Business Unit. A RGBU é uma divisão da Oracle dedicada à área do Retalho e fornece por isso produtos e serviços dedicados a esse mercado.

Nesse contexto, o *Retail Merchandising System* (RMS) é o Sistema responsável por armazenar e controlar todos os dados relacionados com os processos de negócio do retalho (seja ele alimentar, têxtil, basar ou qualquer outro tipo), garantindo a integridade dos mesmos e servindo de *hub* central para a sua integração nos restantes sistemas de uma organização. O RMS inclui funções chave tais como manutenção de artigos, gestão de preços de custo e preços de venda, gestão de stock e reaprovisionamento. Essas funcionalidades fornecem um acesso fácil às informações cruciais para as atividades diárias de *merchandising* dentro de uma organização do setor do retalho, oferecendo aos seus utilizadores a capacidade de se focarem nas principais decisões que ajudam a alcançar metas de vendas e lucros. O RMS surgiu na década de 90, sendo um sistema legado com história (conta já com 19 versões desde a sua criação), concebido através da utilização de tecnologias Oracle, entre as quais o PL/SQL, java, ADF e JET, com intuito de simplificar as práticas comerciais e unir os sistemas comerciais dos retalhistas de forma a entender melhor o cliente. Como o RMS foi desenvolvido de forma escalável e baseado na Web, consegue suportar grandes volumes de dados

encontrados no retalho o que permite aos retalhistas terem mais tempo para se focarem nos resultados finais.

Para o âmbito deste trabalho, o RMS foi um dos produtos em curso de implementação no cliente e seria alvo de customizações ou extensões, visto ser um sistema base passível de ser adaptado aos requisitos de cada cliente, uma vez que cada um poderá ter diferentes práticas de negócio ou requisitos específicos.

4.2 – Módulos RMS

Este subcapítulo fornecerá uma visão geral da informação que o RMS é capaz de manter, quais os processos que suporta e as capacidades de integração com outras aplicações.

4.2.1 – *Seed Data*

O RMS contém dados que devem ser carregados no momento da instalação. As tabelas CODE_HEAD e CODE_DETAIL são exemplos de tabelas com valores exigidos pelo sistema que deverão ser carregadas no momento da instalação, não impedindo que futuramente sejam inseridos novos códigos através de scripts ou então através dos formulários online.

Além disso, algumas tabelas de configuração devem ser também carregadas para que a aplicação abra e funcione corretamente, mesmo que os valores de configuração sejam alterados posteriormente. Essas configurações são mantidas num conjunto de tabelas de SYSTEM_OPTIONS que são necessárias para a configuração inicial, sendo que poderão ser atualizadas antes da implementação para refletir as configurações finais.

4.2.2 – *Foundation Data*

Foundation Data é a base para toda a informação futura na qual o RMS se baseia. Essa informação precisa de estar presente antes de se começar a usar o sistema. A maioria da *Foundation data* pode ser configurada online, no entanto, o mais comum a ser feito é ser executado um processo de *conversion* que irá preencher as devidas tabelas.

Foundation Data consiste, principalmente, em três tipos de informação:

- *Organizational Hierarchy;*
- *Merchandise Hierarchy;*
- *Supplier and Partner Management;*

Organizational Hierarchy

A Hierarquia Organizacional permite criar no máximo seis níveis de relacionamentos hierárquicos de forma a suportar a estrutura operacional da organização.

Numa loja de retalho, um cliente vai à loja e compra, os retalhistas adquirem mercadorias dos fornecedores e o retalhista funciona como cliente. Já numa loja franchisada, o retalhista fornece mercadorias para os clientes franchisados, e neste caso o retalhista funciona como fornecedor e faz acompanhamento das vendas.

A Hierarquia Organizacional apenas suporta dois tipos de lojas de forma a satisfazer os requisitos franchise:

- *Franchise*
- *Company*

Uma loja do tipo *Company* funciona como uma loja de retalho no RMS.

Uma loja do tipo *Franchise* é usada para suportar o modelo de negócio *Franchise*. Outras aplicações vêm as lojas *Franchise* da mesma forma que o RMS vê outras lojas, no entanto, o RMS executa um processamento especial tendo por base o tipo da loja em questão.

Merchandise Hierarchy

A Hierarquia de Mercadorias permite criar no máximo seis níveis de relacionamentos hierárquicos de forma a suportar a estrutura de gestão de produtos da organização. É possível atribuir um comprador e um comerciante nos níveis de DIVISION, GROUP e DEPARTMENT da hierarquia de mercadorias. É possível ainda estabelecer uma ligação entre o nível mais baixo com o nível seguinte da hierarquia, por exemplo, pode-se

indicar a qual GROUP um DEPARTMENT pertence, ou a qual DIVISION um GROUP pertence.

Supplier and Partner Management

Um fornecedor fornece a mercadoria e um *partner* está envolvido noutras transações financeiras que resultam em faturas. A gestão de fornecedores e *partners* fornece as funcionalidades de criar e armazenar fornecedores e *partners* de mercadorias válidas. É possível manter a variedade de informação sobre os fornecedores, como acordos financeiros, parâmetros de gestão de stock, tipos de transações EDI (*Eletronic Data Interchange*) e atributos de correspondência a uma fatura. Os fornecedores são criados normalmente no sistema financeiro e conectados ao RMS. Tudo o que seja enriquecimento de atributos sobre o fornecedor pode ser mantido no RMS.

4.2.3 – *Item Maintenance*

O RMS é responsável pela criação e manutenção de todos os artigos ou itens na terminologia do produto. Para isso, usa uma hierarquia de dados flexível para um item, com níveis que permitem modelar itens da maneira pretendida. A hierarquia consiste em três níveis, do mais alto (nível 1) ao mais baixo (nível 3). Dentro dos níveis definidos para uma família de itens, um nível é considerado como o nível da transação, ou seja, é onde todo o stock e transação de vendas ocorre. Este modelo dá aos retalhistas uma flexibilidade para criar famílias de itens que partilham características comuns.

O RMS cria vários tipos de itens, como itens regulares, itens de depósito, packs, itens de concessão, itens de consignação e itens transformáveis.

Através da manutenção de itens, o RMS também mantém os relacionamentos dos itens com outras entidades, como fornecedores, localizações e atributos.

Com a manutenção de itens existem 4 módulos que estão diretamente relacionados com esta temática, sendo eles:

- *Purchasing*
- *Contracts*

- *Deals*
- *Cost Management*

Purchasing

Este módulo permite criar e manter pedidos de compra (Purchase Orders ou PO's) de variadíssimas formas. Fornece compromissos com os fornecedores para entregar produtos em quantidades especificas e em localizações distintas. Os pedidos de compra são criados manualmente ou automaticamente através do reabastecimento ou de sistemas externos. Os pedidos podem ser criados com base em contratos ou transações, ou diretamente através da entrega direta na loja, ou *Vendor Managed Inventory* (VMI). O RMS fornece também a capacidade de manter os itens, locais e quantidades pedidas para os pedidos de compra.

Contracts

Um contrato é um acordo juridicamente vinculativo com um fornecedor para fornecer itens a um custo negociado.

No RMS, as funções dos contratos encaixam nas funções de reaprovisionamento e pedidos de compra, sendo que as principais funções dos contratos são reservar o tempo de manufatura, seguir a disponibilidade e os compromissos do fornecedor e combiná-los com as necessidades do negócio. O principal benefício comercial dos contratos é ter o envolvimento do fornecedor durante a fase de planeamento dos negócios de um retalhista.

Deals

Um *deal* é um acordo negociado entre o retalhista e o fornecedor que estabelece condições mais vantajosas de aquisição por parte do retalhista mediante o cumprimento de condições estabelecidas na negociação do próprio *deal*.

A gestão de *deals* permite criar e manter *deals* com os *partners* e/ou fornecedores. Os *deal partners* podem ser fornecedores, *distributors* e *manufacturers*. Dentro do *deal*, os

clientes criam componentes desse deal, especificam os itens para cada componente desse *deal*, e definem *thresholds*.

Os componentes são *deals* ou partes de *deals* que um retalhista recebe de um fornecedor. Pode haver múltiplos componentes num único *deal*. Têm de se definir *thresholds* para definir a quantidade e o valor que deve ser comprado ou vendido para receber o *deal*. Os componentes do RMS incluem *deals* não faturados, descontos, promoções financiadas pelo fornecedor, descontos financiados pelo fornecedor e *deals* fixos.

Como funcionalidades, pode-se definir os itens e as localizações para os quais possa ser aplicado o *deal*. Pode-se também escolher incluir, ou excluir, localizações, caso seja necessário.

A funcionalidade *deal pass-through* permite que uma percentagem de um *deal* seja transferida do retalhista para um cliente *Franchise*, sendo que essa funcionalidade apenas e só se aplica às lojas franchisadas.

Cost Management

Uma *cost change* é um ajuste no custo do fornecedor de um item, seja para cima ou para baixo. Antes de criar uma *cost change*, deve-se criar uma lista de razões de alterações de custo definidas pelo utilizador e, de seguida, aplicar uma razão a cada *cost change*.

O custo inicial de um item é estabelecido na configuração do item. O custo do item é ajustado no registo do item até que o estado do item seja *Approved*. Após a provação do item, qualquer alteração de custo deve ser tratada no ecrã das *cost changes*.

Para *cost changes*, o ecrã fornece a capacidade de:

- Aceitar *cost changes* recebidas através de EDI (*flat files*);
- Criar *cost changes*;
- Editar uma *cost change*;
- Visualizar uma *cost change*.

4.2.4 – *Multiple Set of Books*

O suporte a *multiple set of books* fornece uma melhor integração com os sistemas financeiros que suportam *multiple set of books* numa única instalação. Essa opção está ativada por *default* no RMS e o cliente precisará de configurar localizações especificas *Foundation data*, incluindo:

- *Organization units;*
- *Transfer entities;*
- *Set of books Ids.*

4.2.5 – *Inventory Control*

A funcionalidade de Controlo de stock no RMS é o *core* da aplicação. Este é seguido perpetuamente e financeiramente no RMS.

O RMS obtém controlo de inventário através de funções que incluem *transfers, Return to Vendor (RTV), Inventory Adjustments, Sales Upload, Purchase Order Receipts (shipments), Stock Counts, Allocations, Franchise Orders and Returns, and Customer Orders.*

Transfers

No RMS, as *transfers* fornecem uma *framework* organizada para monitorizar o movimento do *stock*. O RMS cria e mantem *transfers*, no entanto, é possível introduzir informações de *transfers* no RMS através de outros sistemas.

O RMS suporta um conjunto de diferentes tipos de *transfers como intercompany transfers, book transfers, Purchase Order-linked transfers, externally generated transfers, customer orders and francise order*. As funções das *transfers* também suportam o movimento de um ou mais itens entre duas *locations* internas do RMS, e *multi-leg transfers* nas quais o entreposto é considerado *finisher location*. *Finishers* são *locations* onde é realizado trabalho de valor acrescentado às mercadorias, como por exemplo a anexação de etiquetas ou outro tipo de transformação.

Mass return transfers são usadas para realocar mercadoria nas *locations* ou para devolver mercadoria aos fornecedores.

Returns to Vendor

As transações *Return to Vendor* (RTV) são usadas para enviar mercadoria de volta para o fornecedor. No RMS, estas transações permitem que um ou mais itens possam ser devolvidos a um fornecedor. Para cada transação, os itens, quantidades, e custos são especificados. Após o envio para fora da *location*, o *stock* é removido do *stock on hand*, isto é, é removido o *stock* disponível.

As RTVs são criadas manualmente no RMS ou então podem ser importadas de sistemas externos. O RMS também fornece a capacidade de manter RTVs. As RTVs enviadas criam uma solicitação de nota de crédito ou uma nota de crédito (com base na configuração do fornecedor) na tabela de *staging* da conferencia de faturas (*invoice matching)* no RMS, para exportação para o *Oracle Retail Invoice Matching* (ReIM).

Inventory Adjustments

Inventory Adjustments são usadas para aumentar ou diminuir *o stock* para contabilizar eventos que ocorram fora do curso normal do negócio, tal como receções, vendas, contagem de *stock*. Esses ajustes são criados no RMS ou importados de um sistema externo. O RMS suporta dois tipos de ajustes sendo eles *stock on hand* ou *unavailable inventory*. Os ajustes podem ser também criados por *locations* para vários itens, por item para várias *locations*, ou através de transformação de produto para uma localização específica.

Purchase Orders Receipts (Shipments)

Purchase Orders Receipts (shipments) registam o aumento disponível quando as mercadorias são recebidas de um fornecedor. O custo médio ponderado (WAC) é recalculado no momento da receção usando o custo inicial da PO (*Purchase Order*). Os registos de auditoria da transação são criados para auditoria financeira e a transação referente à receção fica disponível para *invoice matching.*

Stock Counts

Stock Counts são os processos para os quais o inventário é contado na loja e é comparado com o que se encontra no nível de inventário do sistema para discrepâncias.

4.2.6 – *Replenishment*

O reaprovisionamento automático monitoriza constantemente os níveis do *stock*. Com base nisso, as POs ou *transfers* são criadas para atender à procura do consumidor.

Os itens podem ser configurados para reaprovisionamento automático através do ecrã de *item maintenance*, individualmente ou através de listas de itens.

O reaprovisionamento automático também suporta métodos diferentes para determinar se as POs são criadas e as quantidades solicitadas.

4.2.7 – *Stock Ledger*

O *Stock Ledger* no RMS regista os resultados financeiros dos processos de *merchandising*, como compras, vendas, *price changes*, e *transfers*. Todas essas transações são registadas no RMS *Stock Ledger* e acumuladas no nível de *subclasse/location* por dias, semanas e meses, dependendo das configurações do calendário. Os níveis agregados do *stock ledger* são usados para medir valores do *inventory* e rentabilidade das mercadorias. O *stock ledger* é usado principalmente para fins de *reporting*, no entanto também há alguma visibilidade *online*.

Stock Ledger suporta várias moedas. Todas as informações no nível da transação são armazenadas na moeda local da loja ou do entreposto em que a transação ocorre. À medida que as informações no nível de transação são acumuladas até aos níveis agregados do *stock ledger*, os registos são mantidos na moeda local e convertidos em moeda principal. Isso permite que os *reports* corporativos sejam executados na moeda principal da organização, enquanto ainda fornecem visibilidade por *location* dá rentabilidade na moeda local.

4.3 – Integração do RMS com outras Aplicações

O RMS fornece informações essenciais para todas as aplicações *Oracle Retail Merchandising Operations* Management (ReSA – Auditoria de Vendas, RTM – Gestão de processo de Importação, RPM – Gestão de preço de venda, ReIM – Conferência de Faturas, Allocations) e interage com todos eles ou com sistemas externos com

funcionalidade ou objetivos de negócio equivalente. O RMS existe no mesmo esquema de base de dados de todas as outras aplicações, tal como se verifica na figura 1, o que fornece flexibilidade na maneira como as informações são compartilhadas entre o RMS e as outras aplicações *Oracle Retail Merchandising Operations*.

As informações são compartilhadas com outras aplicações através de leituras diretas das tabelas de aplicações *Oracle Retail Merchandising Operations Management*, através da chamada de *packages* dessas mesmas aplicações, através de *batch processes*, e também através de *Oracle Retail Integration* Bus (RIB) se o cliente estiver a usar essa opção.

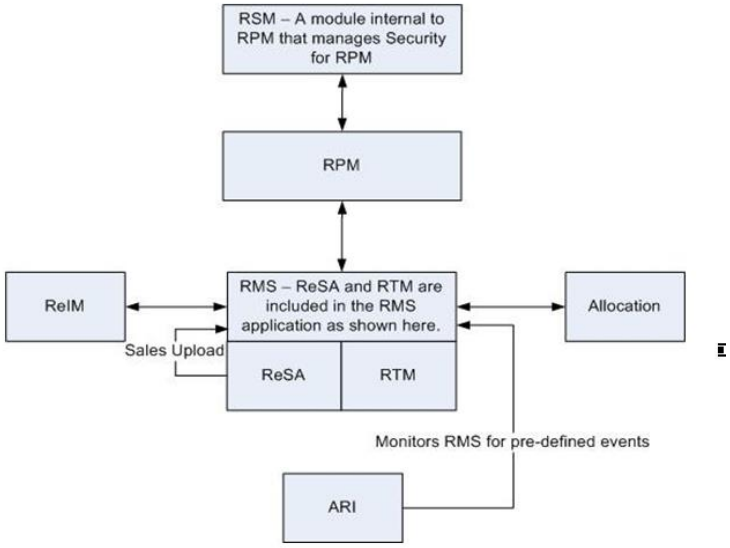


Figura 1- Localização do RMS no esquema de *Merchandising*

4.3.1 – *RMS and RTM*

Oracle Retail Trade Management (RTM) e o RMS partilham a mesma instância de base de dados. Quando o RTM é ativado na instância do RMS, é necessária manutenção de dados ao nível do *country,* fornecedor*, partners e itens*. Esses dados são atualizados diretamente na base de dados do RMS e subsequentemente usados no RTM.

4.3.2 – *RMS and ReSA*

Oracle Retail Sales Audit (ReSA) e o RMS partilham a mesma base de dados. O ReSA partilha alguns dos seus dados mestre com o RMS. *Foundation data*, como *stores*

company/ location *close dates, location traits, bank setup* são mantidas no RMS e usadas no ReSA.

4.3.3 – *RMS and RPM*

Oracle Retail Price Management (RPM) existe no mesmo esquema de base de dados que o RMS, o que permite que as informações sejam partilhadas entre aplicações por meio de leituras diretas à base de dados, chamadas de *packages* e *batch processes*. O RPM usa APIs para facilitar a troca de informações com o RMS. Ao nível do *Foundation Data*, o RPM precisa de conhecer a hierarquia de *merchandise*, a hierarquia organizacional, fornecedores, etc, para poder configurar com sucesso eventos de alteração de preço.

4.3.4 – *RMS and Allocation*

Allocation existe no mesmo esquema de base de dados que o RMS, o que permite que as informações sejam partilhadas entre aplicações por meio de leituras diretas à base de dados, chamadas de *packages* e *batch processes*. Assim sendo, por exemplo, no nível de *Foundation Data*, o RMS fornece informações relativas às *locations* que são válidas para se poder alocar de um local para outro. Ao nível dos Itens, as *allocations* são geradas no nível *item/location* logo é necessário que o *Allocation* perceba quais os *itens* e *item/locations* que são elegíveis no sistema.

4.3.5 – *RMS and ReIM*

Oracle Retail Invoice Matching (ReIM) existe no mesmo esquema de base de dados que o RMS, o que permite que as informações sejam partilhadas entre aplicações por meio de leituras diretas à base de dados, chamadas de *packages* e *batch processes*. Assim sendo, por exemplo, no nível de *Foundation Data* o RMS fornece informações relativas às *locations* válidas para executar *invoices*, fornecedores validos para poderem receber *invoices*, etc.

5 – Tarefas de Customização de *Software* para o Cliente XPTO

5.1 – Apresentação do cliente XPTO

No seguimento de respeitar as regras impostas pela empresa que disponibiliza os seus recursos ao serviço desta dissertação, a ORACLE, não será usado o verdadeiro nome do cliente para quem se destinou todo o trabalho a ser demonstrado. Como tal, usar-se-á um nome fictício, XPTO, para nomear o cliente ao longo de toda a dissertação.

XPTO é a divisão de gestão de sistemas de informação de uma empresa francesa de consultoria de Tecnologias de Informação, pertencente ao setor do Retalho que opera neste ramo já há vários anos. XPTO tem na sua carteira de clientes a representação de 15 marcas, conta com cerca de 92 mil colaboradores e serve em média cerca de 810 mil clientes por dia.

De forma a encontrar uma melhor solução para o seu negócio, XPTO, em 2015, escolheu a ORACLE para realizar um projeto com o objetivo de usar o ERP da mesma. Em 2016, o projeto iniciou-se e encontrou-se em atividade até ao momento em que se iniciou a pandemia do COVID-19, sendo que o plano previa trabalho até 2021. Para efeitos desta dissertação apenas será considerado o período diferido entre junho 2019 e janeiro 2020.

5.2 – Apresentação das tarefas de customização

Na figura abaixo, Figura 2, está representada numa linha temporal, todas as tarefas que foram desenvolvidas em diferentes momentos do projeto. O projeto foi dividido em PIs – *Program Increments*, que agrupam um conjunto de *sprints* – de 3 meses de duração cada, sendo que cada *sprint* tinha uma duração de 2 semanas (salvo algumas exceções devido a férias, por exemplo PI2, devido ao mês de agosto):

- PI1: 13 de março de 2019 a 15 de julho de 2019;
- PI2: 22 de julho de 2019 a 4 de novembro de 2019;
- PI3: 11 de novembro de 2019 a 27 de janeiro de 2020;
- PI4: 19 de janeiro de 2020 a 25 de março de 2020;

Assim sendo, cada tarefa demonstrada nesta dissertação pertence a um determinado PI.

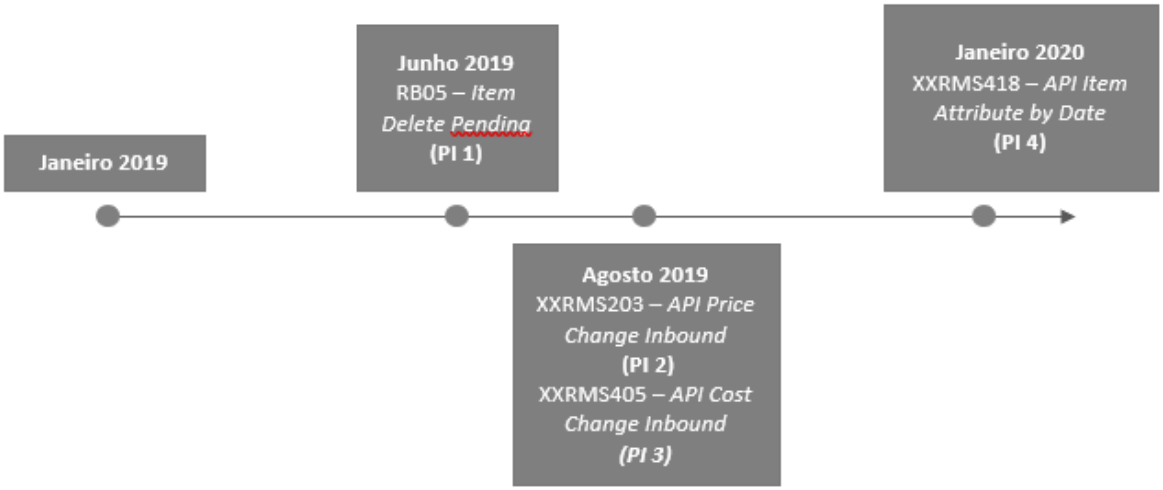


Figura 2- Distribuição Temporal das tarefas por PI

5.2.1 – Junho 2019: *RB05 Item Delete Pending (PI1)*

O RB05 consiste num processo de determinação de necessidade de purga de um artigo, que irá automaticamente avaliar o estado de um determinado atributo ao nível dos artigos (um "UDA"), e se esse atributo tiver estado *Supprimée* (deixar de existir), os *itens* serão movidos para uma tabela de purga do RMS ficando à espera de ser apagados.

Para esta tarefa foi preciso ter em consideração que um *item* tem vários estados ao longo do seu ciclo de vida, sendo que para o efeito desta tarefa apenas o estado *Supprimée* teve grande preponderância. Tipicamente, *itens* que terão este tipo de estado serão *itens* personalizados que irão ter um curto ciclo de vida. Estes *itens* são criados para satisfazer as necessidades de uma encomenda de cliente final e, após isso, são listados para purga o mais rápido possível para reduzir o volume desnecessário de dados no RMS. Assim sendo, como parte final do ciclo de vida de um item, XPTO pretende garantir que os itens sejam fisicamente apagados do RMS. Para isso, para todos os itens identificados é necessário verificar qual é o *item level* a que corresponde, porque em caso de serem *item parents*, ou seja, *item level* < *Transactional Level*, deverão ser marcados com 2 na ordem de *delete*, enquanto que os outros *itens* deverão ter ordem 1, de forma a que primeiro sejam sempre apagados primeiro os SKUs e só após

isso sejam apagados os *item parents* para que se possa eliminar também as relações *item locations*.

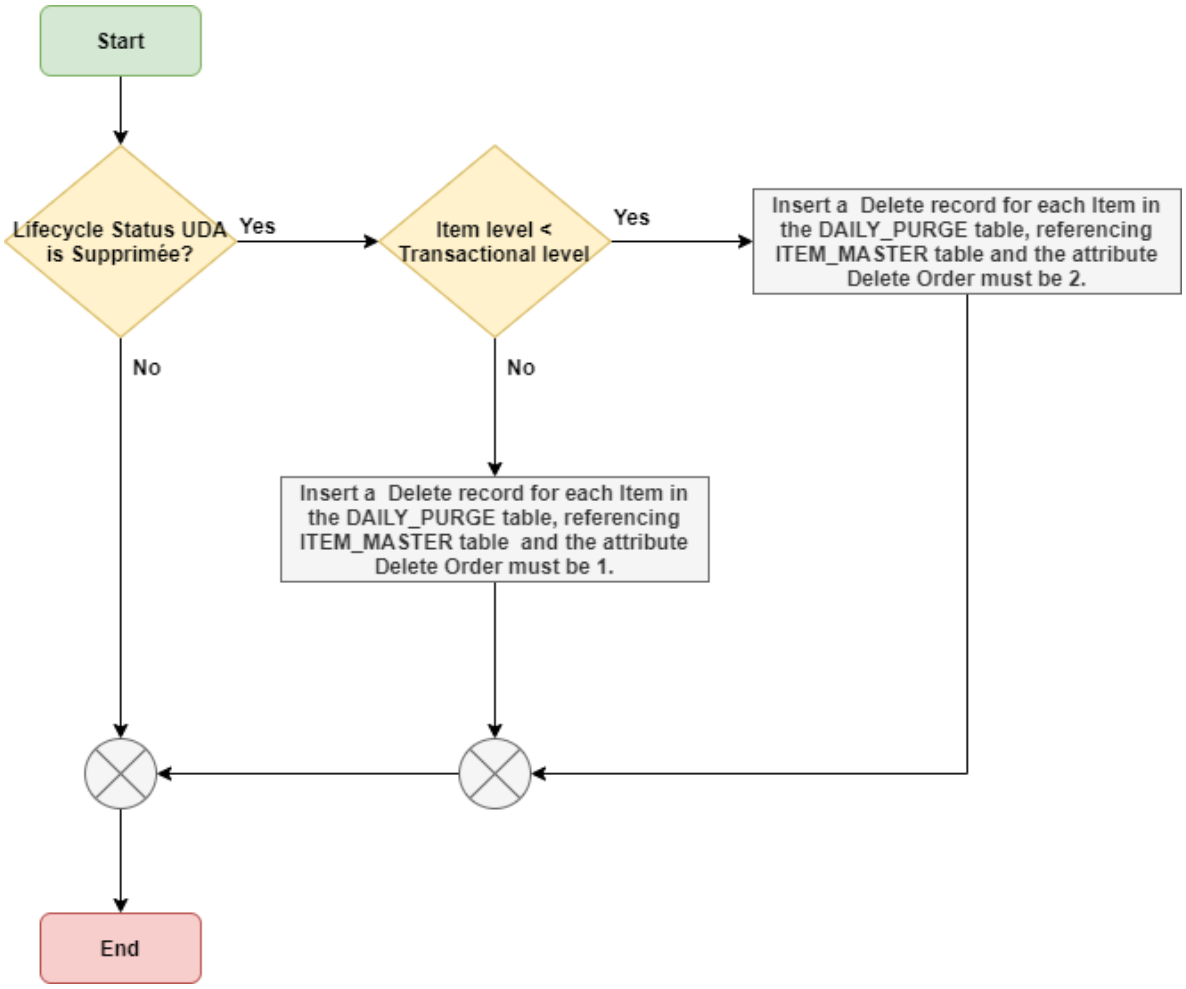


Figura 3- Diagrama para o RB05

De forma a implementar os acontecimentos descritos acima, foi necessário criar um novo *batch* (*XPTO_ITEM_DELPEND.pc*) bem como um novo *package* de base de dados (*XPTO_ITEM_DELPEND*), que se encontram na secção Anexos, concretamente no Anexo B deste documento, que irá conter toda a logica de processamento desta tarefa bem como o resultado de um teste. A figura 3 ilustra o que foi implementado para a construção do *package* de base dados. Quanto ao *batch*, este apenas servirá para fazer uma chamada do *package XPTO_ITEM_DELPEND* para que seja feita a purga dos *itens*

quando todas as condições estejam cumpridas. Este *batch* irá constar numa lista diária de *batchs* que correm de forma automática durante a noite (*nightly batches*).

5.2.2 – Agosto 2019: XXRMS203 *API Price Change Inbound* (PI2)

A interface XXRMS203 consiste numa API de *inbound* que tem como função fazer a comunicação com os restantes sistemas externos acerca de *price changes* que possam surgir, e irá comunicá-las ao RPM, onde ficarão alocados numa área de *staging*. Posteriormente, esses registos que irão constar nas tabelas de *staging* do RPM, após a inserção via *API price change*, irão ser consumidos por um *batch* que irá processar cada um dos registos e irá criar as respetivas *price changes* no RPM, se for apropriado.

Nota: O *batch* que irá processar os registos da *staging* do RPM não faz parte do objetivo desta API, mas para efeitos de melhor entendimento do processo foi mencionado.

Esta nova API para o *upload* de *price changes* deverá receber como *input* um conjunto de registo de *price changes* para inserir na tabela *staging* do RPM e é necessário retornar como *output* o mesmo conjunto de registos, onde cada um será enriquecido com um estado do resultado, seja ele um erro ou não. Esta API irá retornar um estado e uma mensagem de erro globais, tabela 2. A ideia é fazer chegar a quem invocou a API um estado geral sobre o processamento da API, sendo que em caso de erros críticos na execução, a mensagem de erro global irá preenchida com informação relevante.

Para o estado global, foram definidos alguns códigos de erro:

Código do Estado Global	Descrição	Significado
0	**Sucesso**	Toda a operação foi executada com sucesso.
1	Excedeu o número máximo permitido de registos	O número de registos excedeu o limite máximo estabelecido na configuração da API.

2	Processado com erros	A operação foi completada, mas alguns registos falharam e não foram inseridos na tabela de *staging* do RPM.
255	Erro	Toda a operação falhou sendo que nenhum registo foi inserido na tabela de *staging* do RPM.

Tabela 2 - Códigos de Erro do Estado Global: API

Como se pode receber dos sistemas externos mais do que uma *price change*, foi necessário criar códigos de estado ao nível do detalhe, tabela 3. Esta necessidade de criar estes códigos para o detalhe surge no sentido em que, como podemos receber mais do que uma *price change* para processar, a API vai tratando uma a uma e surge a necessidade de em cada processar de cada um dos registos, atribuir um estado ao nível do detalhe para que, no final de todos os registos serem processados, tendo em conta os estados do detalhe, se possa atribuir um estado global (detalhados na tabela 2). Assim sendo, como possível retorno de estados para o detalhe, pode ter-se:

Código do Estado Detalhe	Descrição	Significado
0	Sucesso	Toda a operação foi executada com sucesso.
1	Dados Inválidos	Dados inválidos foram requisitados para o registo processado, resultando num erro específico de falha nos dados. Isto significa que o registo não foi inserido na tabela de *staging* do RPM, mas não impede que os outros registos sejam processados.
255	Erro	Um erro ocorreu durante o processar do registo para a tabela de *staging* do RPM. Uma descrição significativa do erro será escrita neste campo.

Tabela 3 - Códigos de Erro do Detalhe: API Price Change

Relativamente à logica de processamento, cada vez que a operação de upload é invocada, a seguinte lógica terá de acontecer:

- Realizar validações iniciais (figura 4):
    - Verificar se o número máximo de registos para *upload* está ou não configurado;
    - Verificar se o número de registos para processar não é excedido;
    - Se alguma das validações acima falhar, retornar o estado/mensagem de erro apropriada – nenhum registo irá ser processado;

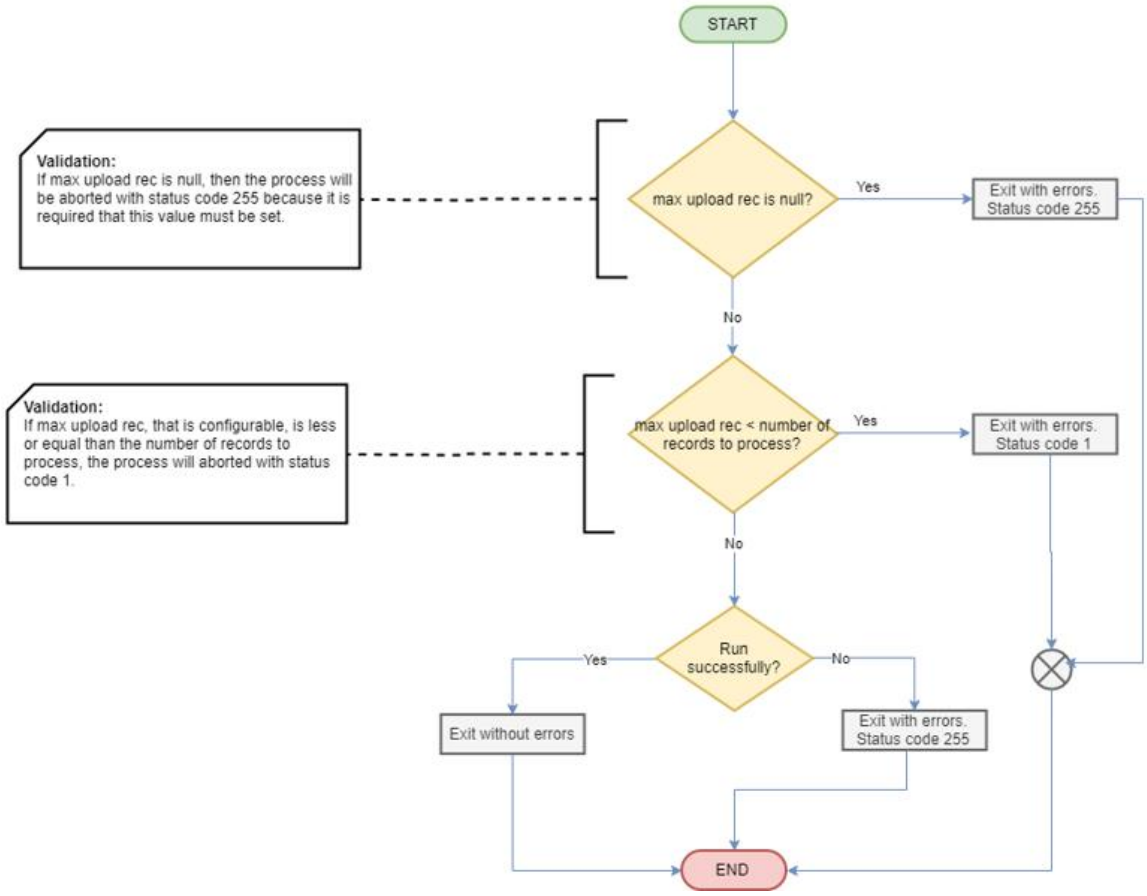


Figura 4- Diagrama da função das Validações Iniciais

- Para cada registo presente na coleção de *price changes* recebida é necessário:
    - Identificar o tipo de evento:
        - ***Create*** (figura 5) – A comunicação de um evento deste tipo assume que a fonte de dados tem conhecimento de todas as *price change* existentes no sistema e identifica o registo da *price change* como algo completamente novo para o RPM (para um

específico r*eason code*). Neste caso, o registo deve ser criado na área de *staging* com estado "N".

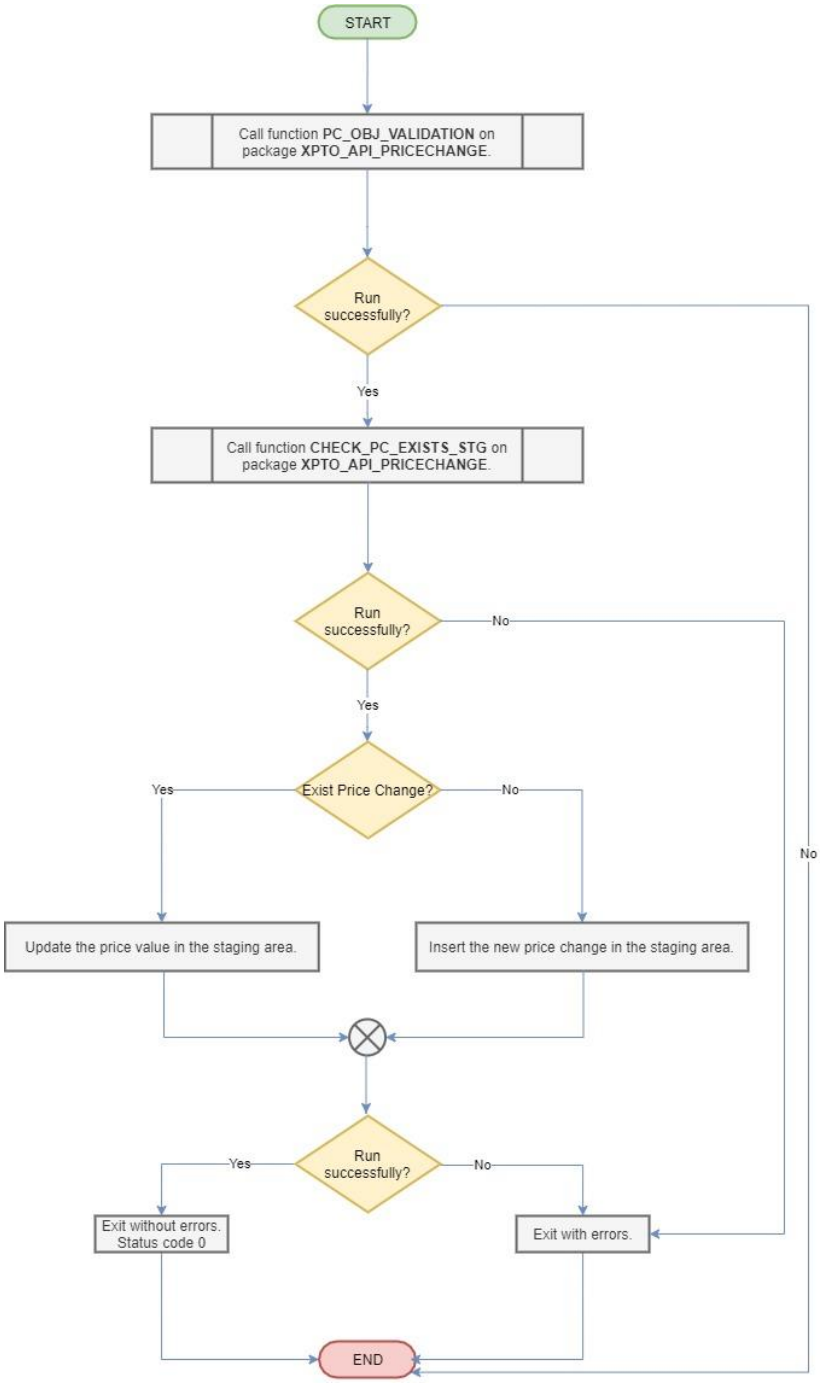


Figura 5- Diagrama da função *Create*

- ***Update*** (figura 6) – A comunicação de um evento deste tipo assume que a fonte de dados tem conhecimento de todas as *price changes* existentes no sistema e identifica o registo da

price change como algo que já existe. Neste caso o registo deverá ser criado na área de *staging* com estado “U”.

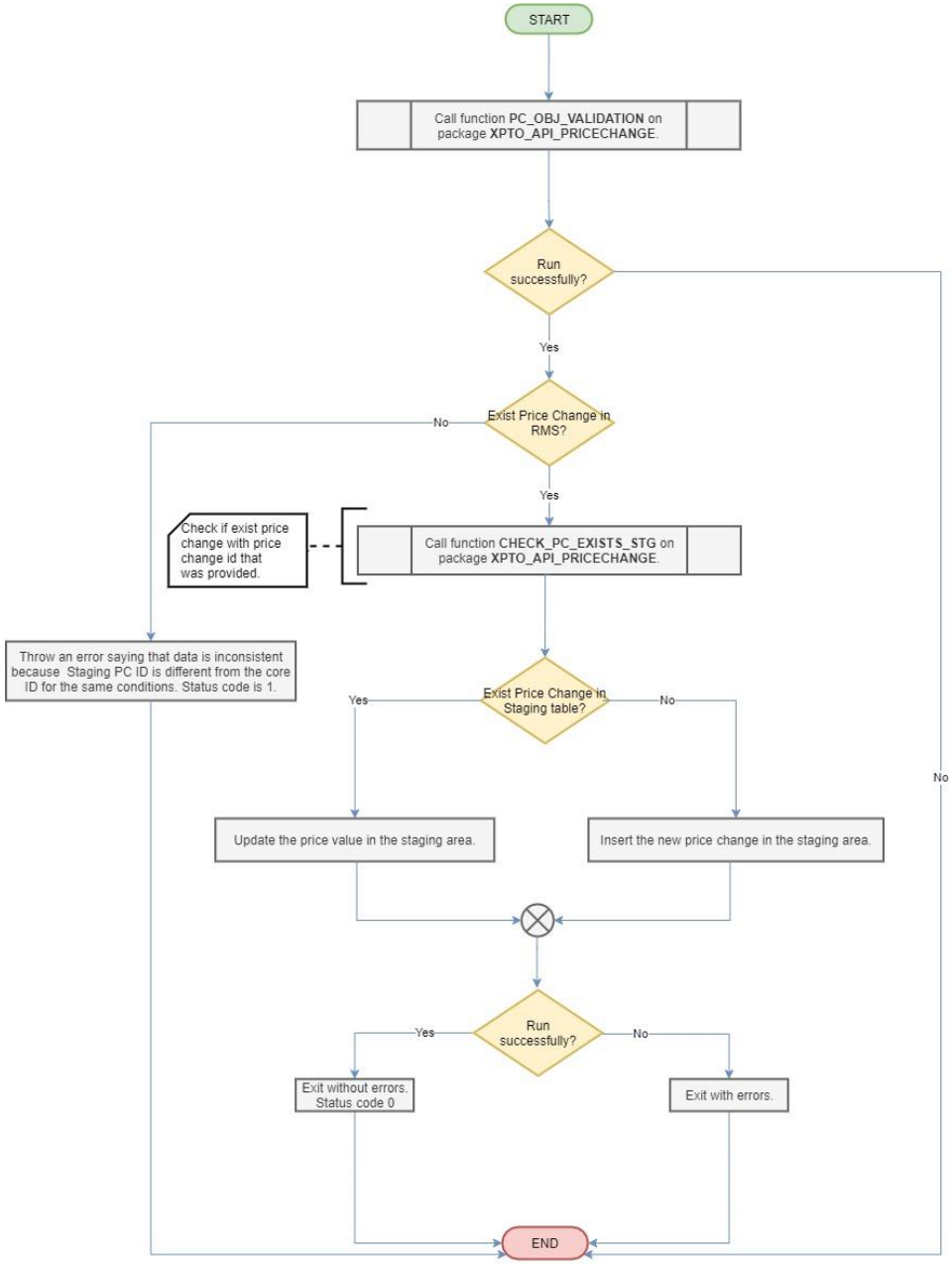


Figura 6- Diagrama da função *Update*

- ***Upsert*** (figura 7) – Esta operação suporta o requisito do cliente que é: o sistema deverá ser capaz de definir um estado válido para o registo da *price change* na área de *staging* quando o sistema não tiver visibilidade sobre o que existe no RPM. Para este cenário, a *interface* deverá executar a seguinte lógica:

- Validar nas tabelas do RPM se a *price change* existe para a combinação *Item/location* ou para a combinação *zone/ active date/ effective date/ reason code*;
- Se o registo for encontrado e tiver *active date* no futuro (*active date* > *Vdate*), então, para este registo em específico, irá ser inserido um registo na *staging* com estado 'U';
- Se o registo for encontrado e tiver *active date* no futuro (*active date* <= *Vdate*), então, para este registo em específico não será criado um registo na *staging* e retornará, para esse *record*, um código de erro 255 e uma mensagem de erro "Não se pode criar ou modificar uma *price change* ainda ativa".

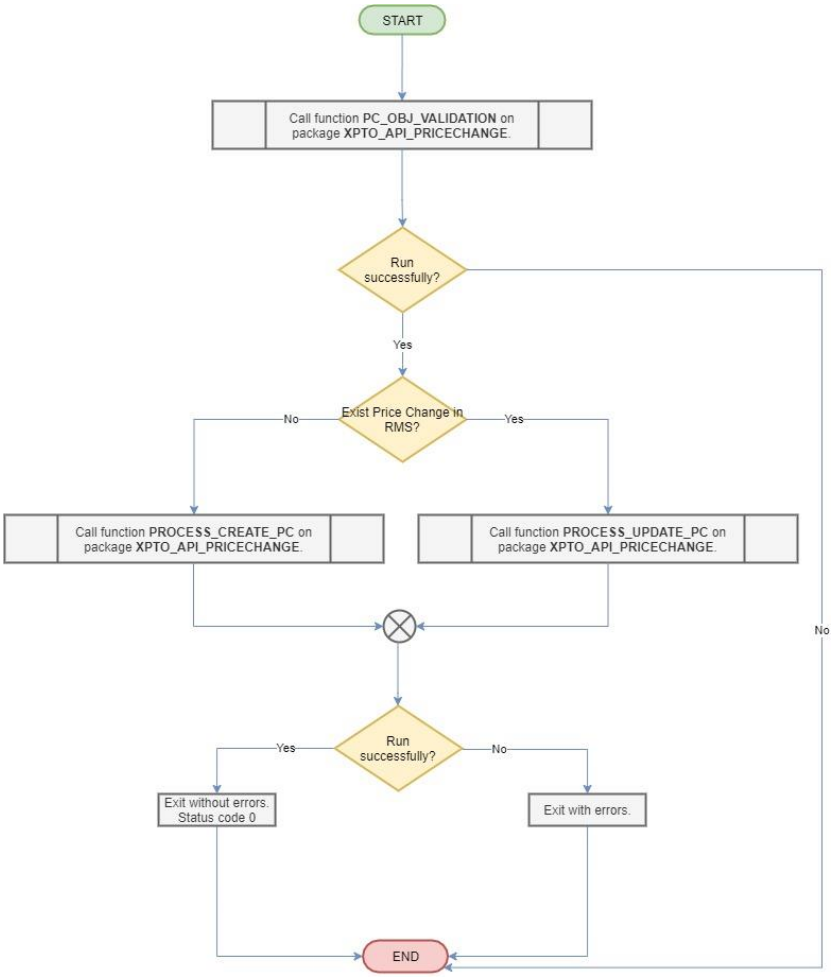


Figura 7- Diagrama da função *Upsert*

- ***Delete*** (figura 8) – Esta operação irá ser responsável por identificar todas as *price change* que são para apagar do sistema. Para isso, a chave para a pesquisa de tudo o que será para apagar é a procura de todas as combinações *Item/location* ou da combinação *zone/ effective date/ reason code/ price value*. Se existir um registo que cumpra esta validação, então irá ser inserido um novo registo na *staging* com estado "D". Se o registo não existir, então irá ser retornado 1 como código de erro e uma mensagem de erro "*Price change* não encontrada".

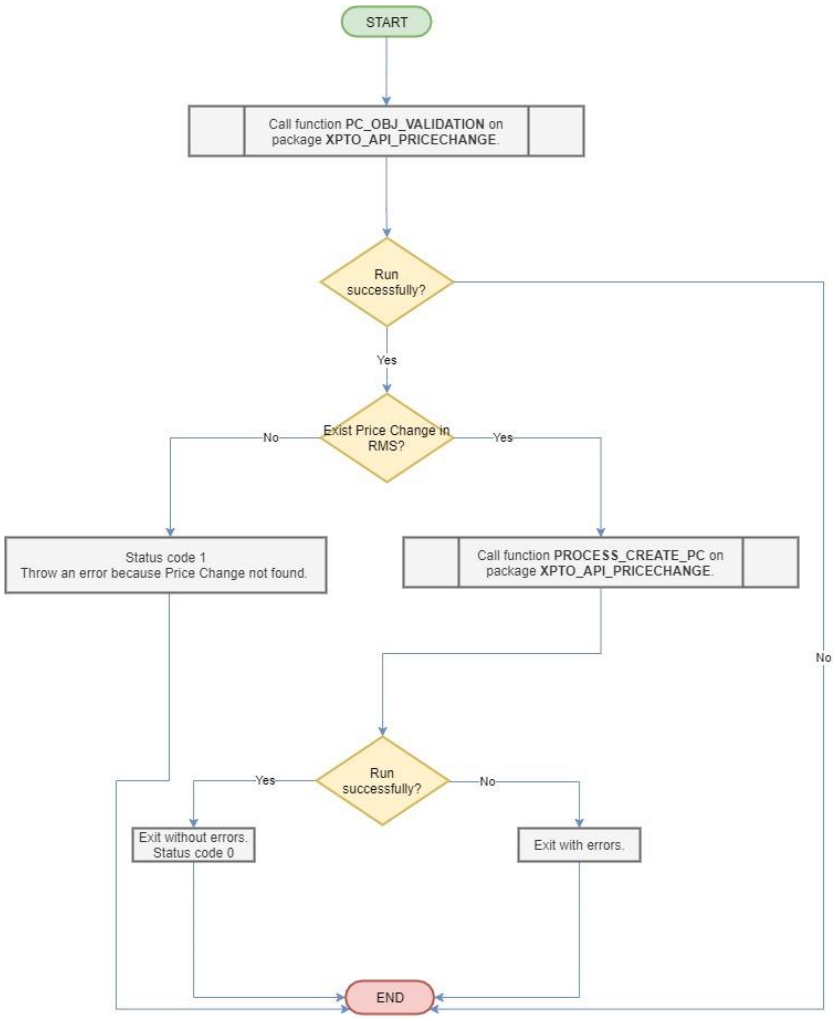


Figura 8- Diagrama da função *Delete*

- Retornar um estado e uma mensagem de erro quando aplicável. Se o *upload* do registo for feito com sucesso, então retornar um código de erro 0. Se o registo não puder ser inserido por dados inválidos (por exemplo, faltar um atributo

que é obrigatório), então é necessário capturar o erro em detalhe e retornar o código de erro 1 com mensagem de erro “Dados inválidos”. Se o registo não for processado derivado de uma falha técnica (por exemplo, falha na chave primária), então retornar para o registo um código de erro 255 e mensagem de erro com a descrição relevante do mesmo. Em qualquer um destes cenários acima descritos, sempre que possível, o processo deverá continuar processando registo a registo até ao final do total de registos.

Como já foi mencionado acima, um dos *outputs* desta API é o mesmo conjunto de dados recebido para *upload*, mas a estes dados irá ser adicionado detalhe ao nível do código de erro bem como da mensagem de erro aplicável. Se um erro critico acontecer que impossibilite o processamento da API, então deverá ser capturado o erro a alto nível retornando um código de erro e uma mensagem de erro adequadas.

De forma a implementar toda a lógica descrita acima, para a API *Price Changes* foi necessário criar um novo *package* para conter todo este processamento. Na secção Anexos, concretamente no Anexo B deste documento, encontrar-se-ão os detalhes ao nível do código necessário para a implementação desta API. Segue-se abaixo o diagrama, figura 9, que ilustra todo o processamento, em geral, da API *Price Changes*. Esse procedimento será criado no novo package e será auxiliado for funções como as descritas acima. Para além disso irá conter mais um conjunto de validações extra que irão ajudar a validar se os parâmetros estão bem preenchidos, e se os dados já existem ou não no RMS. Para isso foram criadas funções especificas, restritas deste procedimento que só serão usadas dentro das funções do procedimento, tais como Validações Iniciais, *Create, Update, Upsert e Delete*. Para além disso, foi necessário criar um objeto, com uma estrutura bem definida de dados para se poder receber os dados dos sistemas externos, bem como para no final da execução da API devolver a esses sistemas externos a mesma estrutura de dados com a informação adicional devidamente preenchida, o estado da execução e a mensagem de erro associada, quando aplicável.

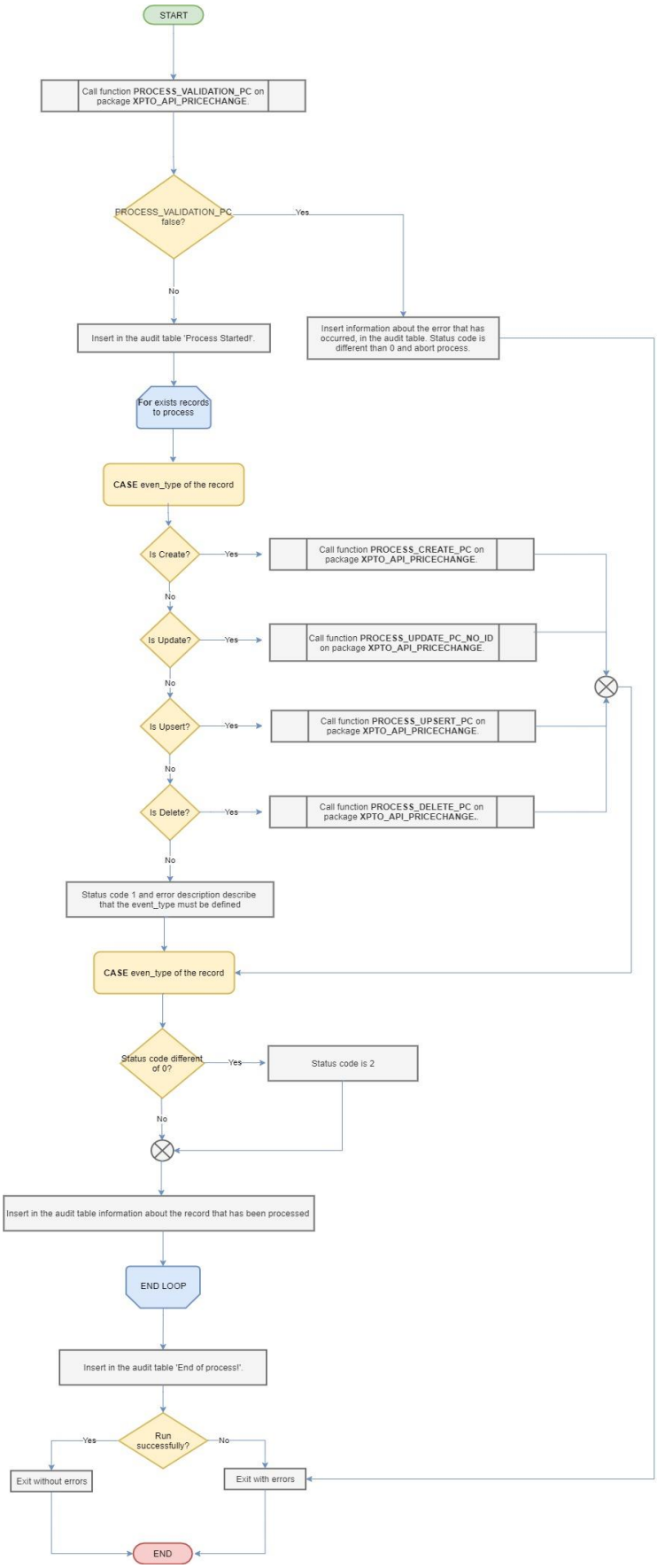


Figura 9- Diagrama de todo o procedimento da *API Price Changes*

5.2.3 – Agosto 2019: XXRMS405 API *Cost Change Inbound (PI2)*

A *interface* XXRMS405 consiste numa API de *inbound* que tem como função fazer a comunicação com os restantes sistemas externos acerca de *cost changes* que possam surgir, e irá comunicá-las ao RMS.

Esta nova API deverá receber um conjunto de *cost changes*, a serem introduzidas no RMS, e ainda deverá receber uma *flag* a indicar o modo a que deverá ser processada. Assim sendo, esta API poderá ser executada em dois modos:

- **Síncrono –** os registos serão inseridos na *staging* e depois serão processados para as tabelas de *cost changes* do RMS. Um estado será retornado indicando se a *cost change* está apta para ser introduzida no RMS com sucesso;

- **Assíncrono –** os registos serão inseridos na *staging* e depois serão colocados numa *queue* para processamento assíncrono.

Dependendo da operação a ser executada pelo sistema externo, um retorno *output* será pedido – o mesmo conjunto de registos, onde cada um será enriquecido com um estado do resultado, seja ele um erro ou não. Esta API irá retornar um estado e uma mensagem de erro globais, tabela 4. O objetivo é fazer chegar a quem invocou a API um estado geral sobre o processamento da API, sendo que em caso de erros críticos na execução, a mensagem de erro global irá preenchida com informação relevante.

Para o estado global, foram definidos alguns códigos de erro:

Código do Estado Global	Descrição	Significado
0	Sucesso	Toda a operação foi executada com sucesso.
1	Excedeu o número máximo permitido de registos	O número de registos excedeu o limite máximo estabelecido na configuração da API para os modos específicos, síncrono e assíncrono.

2	Processado com erros	A operação foi completada, mas alguns registos falharam e não foram inseridos na tabela de *staging* do RMS para modo assíncrono, ou alguns registos falharam e não foram inseridos na tabela de *staging* do RMS para modo síncrono.
255	Erro	Toda a operação falhou sendo que nenhum registo foi inserido na tabela de *staging*

Tabela 4 - Código de Estado de Erro Global: API Cost Change

Como se pode receber dos sistemas externos mais do que uma *cost change*, foi necessário criar códigos de estado ao nível do detalhe, tabela 5. Esta necessidade de criar estes códigos para o detalhe surge no sentido em que, como podemos receber mais do que uma *cost change* para processar, a API vai tratando uma a uma e surge a necessidade de em cada processar de um dos registos, atribuir um estado ao nível do detalhe para que, no final de todos os registos serem processados, tendo em conta os estados do detalhe, se poder atribuir um estado global, que estão detalhados na tabela 4. Assim sendo, como possível retorno de estados para o detalhe, pode ter-se:

Código do Estado Detalhe	Descrição	Significado
0	Sucesso	Toda a operação foi executada com sucesso.
1	Dados Inválidos	Dados inválidos foram requisitados para o registo processado, resultando num erro específico de falha nos dados. Isto significa que o registo não foi inserido na tabela de *staging* para *cost changes*, mas não impede que os outros registos sejam processados.
2	Erro de Indução	Validações especificas falharam durante o processamento em modo síncrono. O registo não foi inserido nas tabelas do RMS.

255	Erro	Um erro ocorreu durante o processar do registo para a tabela de *staging* para *cost changes*. Uma descrição significativa do erro será escrita neste campo.

Tabela 5 - Código de Estado de Erro do Detalhe: API Cost Change

Relativamente à logica de processamento, cada vez que a operação de *upload* é invocada a seguinte lógica terá de acontecer:

- **Validações de Sistema –** O primeiro passo consiste em validar se as configurações da API estão devidamente configuradas, para o número máximo de registos que podem ser processados de uma vez. Se nenhuma configuração obrigatória estiver configurada, o processo deverá parar e retornar um erro;

- **Validações Base –** Devem ser feitas validações aos parâmetros de *input* do objeto de forma a se ter a certeza de que estão bem preenchidos. Se esta validação falhar, o processo deverá parar e retornar um erro. Por outro lado, a API deverá continuar e irá validar se o número de registos presentes no conjunto de dados recebidos como *input* se excede ou não o valor configurado para a API, para o modo escolhido (síncrono ou assíncrono), e no caso de ser superior ao configurado, o processo deverá parar e retornar o código do erro 1 - "Excedeu o número máximo de registos permitidos";

- **Processamento dos Registos –** Uma vez que as validações passaram com sucesso, a API poderá começar a processar o que recebeu como input. Ela irá iterar por cada *cost change* recebida, e irá processar algumas validações de negócio e técnicas, para no final poder inserir na *staging* os registos;
    - Dependendo do tipo de evento, a API irá ter diferentes ações:
        - ***Upsert*** (figura 10) - A comunicação de um evento deste tipo assume que a fonte de dados não tem conhecimento de todas as *cost change* existentes no sistema. Poderá ser um *Create* ou um *Update* de uma *cost change*. Neste caso, caberá à API detetar qual será a operação: criar um registo na *staging* tendo

como *action New* para criação ou *MOD* para *update*. Dependendo da operação identificada para ser processada, a lógica descrita abaixo para as ações *Create* ou *Update* irá ser executada;

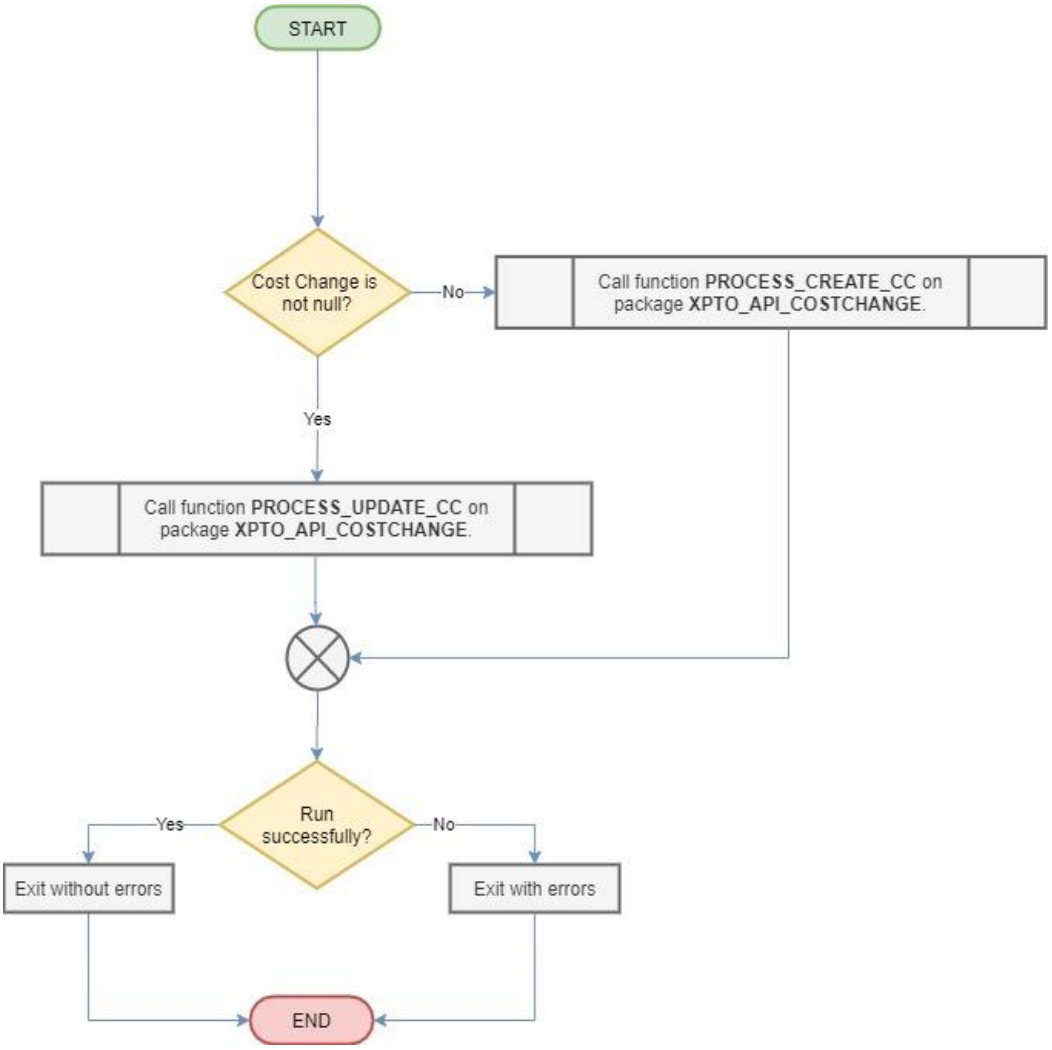


Figura 10- Função *Upsert API Cost Change*

- ***Cancel*** (figura 11) - A comunicação de um evento deste tipo assume que a fonte de dados tem conhecimento de que os dados da *cost change* a que se refere, existe no sistema. Neste caso, um registo deverá ser criado na *staging* tendo *action MOD*. O registo a ser criado irá ter estado *Cancel* e terá de ter um *cost change ID* existente (será procurado internamente no caso de

não ser fornecido no *input*), relacionado com uma *cost change* existente no RMS;

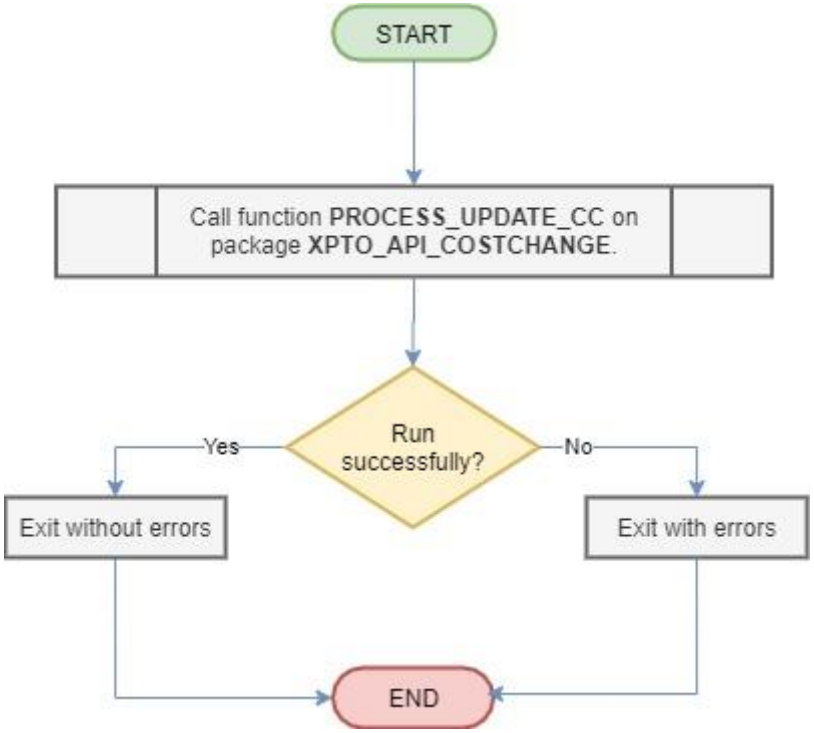


Figura 11- Função *Cancel API Cost Change*

Neste momento, não é espectável que a fonte tenha conhecimento de todas as *cost changes* existentes no RMS, como tal, apenas evento do tipo *Upsert* e *Cancel* irão ser recebidos. Ambos os modos irão realizar uma procura interna no RMS pelo *cost change ID* que irá coincidir com a chave *Item/supplier/country/active date/location type*. Deverá ser dada particular atenção ao caso em que existem várias *cost changes* para a mesma chave – isto pode acontecer porque o RMS permite que várias *cost changes* existam, mas apenas uma se encontre em estado *Approved*, as outras deverão estar em estado *Worksheet*, por exemplo. Neste cenário, a lógica aplicada é a seguinte:

- Se quando procurar no RMS pela *cost change* baseada na chave mencionada acima, um registo em estado *Approved* existir, deverá retornar o *cost change ID* para esse mesmo registo;

- Se não existir um registo em estado *Approve*d mas existirem várias cost changes em estado *Worksheet*, então irá retornar o Cost Change ID do registo mais recente;

Os tipos de eventos que se seguem serão implementados e expostos pela API, no entanto não será expectável que os sistemas externos invoquem esses tipos de eventos. Eles apenas irão ser usados internamente para suportar o evento *Upsert*:

- ***Create*** (figura 12) - A comunicação de um evento deste tipo assume que a fonte de dados tem conhecimento de todas as *cost change* existentes no sistema e identifica o registo da *cost change* como sendo algo completamente novo para o RMS. Neste caso, um registo deverá ser criado na *staging* tendo *action New*, e um *cost change ID* temporário será gerado para o registo enquanto estiver na *staging*. Após isso, deverão ser usadas as validações e os processos base para integrar o registo nas tabelas finais de *cost changes*, com um *cost change ID* final;

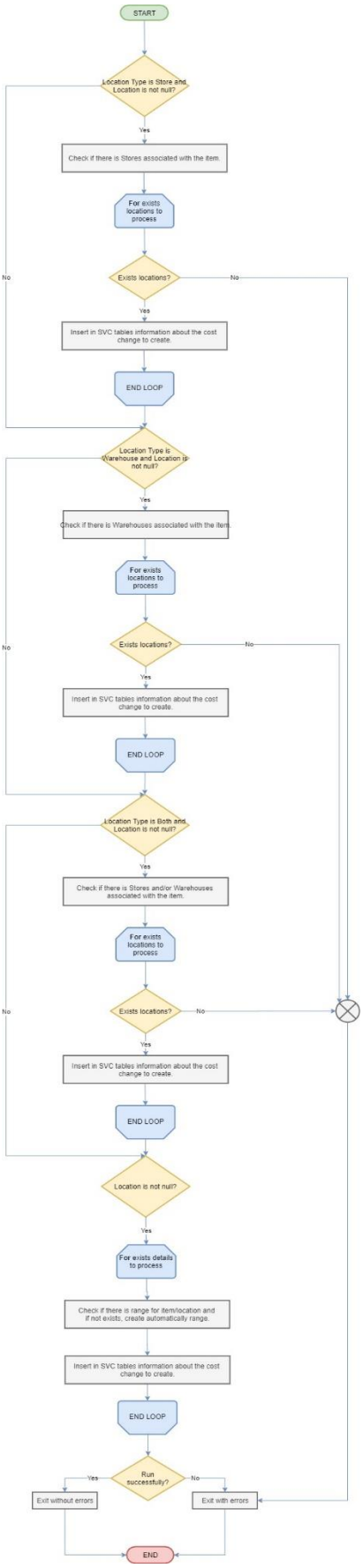


Figura 12- Função *Create API Cost Change*

- ***Update*** (figura 13) - A comunicação de um evento deste tipo assume que a fonte de dados tem conhecimento de todas as *cost change* existentes no sistema e identifica o registo da *cost change* como algo que já existe no sistema (quem invoca este tipo de evento deverá fornecer o *cost change ID*, relacionado com uma *cost change* que já existe no RMS). Neste caso, dependendo do estado do registo para fazer *update*, a seguinte lógica deverá ser aplicada:

    - Se o registo existente não estiver em estado *Approve*: o registo deverá ser inserido na *staging* tendo *action MOD* e os novos valores recebidos, o novo estado será definido de acordo com os valores recebidos;

    - Se o registo existente estiver em estado *Approve*: Um registo deverá ser inserido na staging tendo action "MOD" tendo todos os novos valores recebidos (*update cost* e *reason code*) mas estado igual a *Worksheet*; De seguida, outro registo deverá ser inserido, igual ao anterior, mas tendo estado *Approve*. Isto é obrigatório porque o processo não permite, para uma *cost change* já aprovada, modificar os valores e aprovar numa só linha. A *cost change* deve ser modificada primeiro para o estado *Worksheet* e só depois poderá ser aprovada.

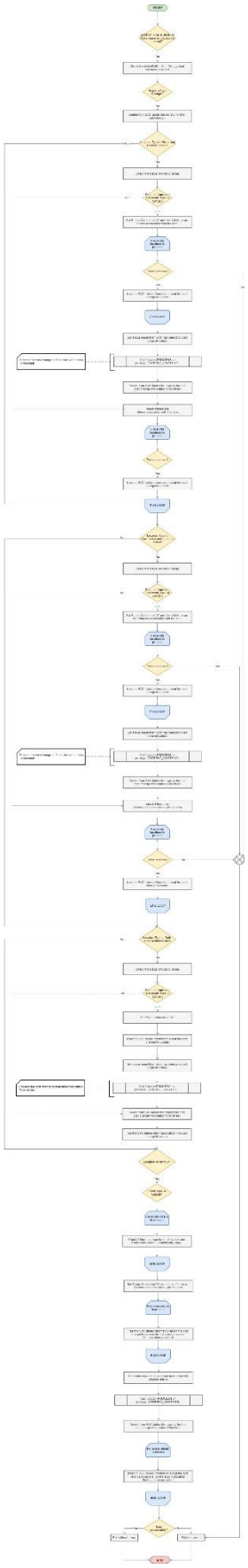


Figura 13- Função *Update API Cost Change*

Uma vez que o registo já está inserido na *staging*, ter-se-á de usar as validações e os processos base para integrar os registos nas tabelas finais de *cost changes*, atualizando a *cost change* a que faz referência com as informações do registo.

Uma abordagem especial deve ser tida em conta em relação ao campo *location type* e *location* que fazem parte do registo:

- Quando um único detalhe existe para o registo, a *location type* é preenchida e a *location* está vazia: se a *location type* é *warehouse*, então um detalhe do registo da *cost change* tem de ser criado ao nível da Item/location para todos o *warehouse* físicos que estão *ranged* para o item. Se a l*ocation type* é *store*, então um detalhe do registo da *cost change* tem de ser criado ao nível da *Item/location* para todas as *stores* que estão *ranged* para o item.

- Quando existem vários detalhes, a *location type* sendo *warehouse* ou *store*, a *location* terá de ser fornecida para cada uma, e um registo deverá ser criado na *staging* para cada um ao nível da *Item/location.* É expectável que a *location* fornecida no caso de ser um *warehouse* seja o *physical WH ID.*

- Quando um único detalhe existe para o registo, a *location type* é "B" para *Both*, e a l*ocation* está vazia: criar um detalhe para o registo da *cost change* ao nível do Item.

Dependendo do modo de processamento, diferentes ações irão ter lugar.

- **Processamento Síncrono –** Para cada registo inserido na *staging*, será invocado o processamento base para integrar cada registo no RMS de acordo com o tipo de evento e o estado fornecido. Como resultado desta operação, o objeto de *input* será enriquecido de informação, para a instância do *cost change ID* final, e com o devido código de erro e a respetiva mensagem quando aplicável. Se a inserção na *staging* falhar devido às validações, então um código de erro 1 deverá ser usado, enquanto que se falhar durante a inserção da *staging* nas tabelas finais de *cost changes*, um código de erro 2 deverá ser usado.

- **Processamento Assíncrono –** Cada registo inserido na *staging* será colocado numa *queue*, de modo a que o processamento assíncrono os escolha para processamento, quando possível. Neste caso, as operações da API acabam aqui, sem esperar que registos sejam processados e integrados no RMS. O objeto de *input* que será retornado é enriquecido de informação, com o *process ID* ligado ao que está na *staging*, e um correto código de erro, quando aplicável. Se a inserção na *staging* falhar devido às validações, então um código de erro 1 deverá ser usado.

Um dos outputs esperados da API é o mesmo conjunto de registos recebido como input, com a adição do estado e da descrição do erro quando aplicável (isto é espectável para cada registo no nível mais alto, o *header*, não incluindo o detalhe). Os registos deveram ser capazes de receber o *process ID* que lhes será atribuído quando forem inseridos com sucesso na *staging* para o modo síncrono/assíncrono, e receberão o *cost change ID* final quando forem inseridos nas tabelas finais de *cost changes* no RMS para o modo síncrono apenas.

No caso de um erro critico, que impossibilitará a API de executar (por exemplo um objeto invalido) então a alto nível um código de estado erro deverá ser retornado e uma descrição técnica do erro também. Este código de estado de erro pode ser retornado no caso de o limite de registos ser atingido, ou se um ou mais registos tiverem falhado validações e não foram integrados.

De forma a implementar toda a lógica descrita acima, para a API *Cost Changes* foi necessário criar um novo package para conter todo este processamento. Na secção Anexos, concretamente no Anexo B deste documento, encontrar-se-ão os detalhes ao nível do código necessário para a implementação desta API. Segue-se abaixo o diagrama, figura 14, que ilustra todo o processamento, em geral, da API *Cost Changes*. Esse procedimento será criado no novo package e será auxiliado for funções como as descritas acima. Para além disso irá conter mais um conjunto de validações extra que irão ajudar a validar se os parâmetros estão bem preenchidos. Para isso foram criadas funções especificas, restritas deste procedimento que só serão usadas dentro das funções do procedimento, tais como Validações, *Create, Update, Upsert* e *Cancel*, sendo que só é expectável que quem fizer uma chamada a esta API apenas utilize *Upsert* e *Cancel*. Para além disso, foi necessário criar um objeto, com uma estrutura bem definida

de dados para se poderem receber os dados dos sistemas externos, bem como para no final da execução da API devolver a esses sistemas externos a mesma estrutura de dados com a informação adicional devidamente preenchida, sendo ela o estado da execução e a mensagem de erro associada, quando aplicável.

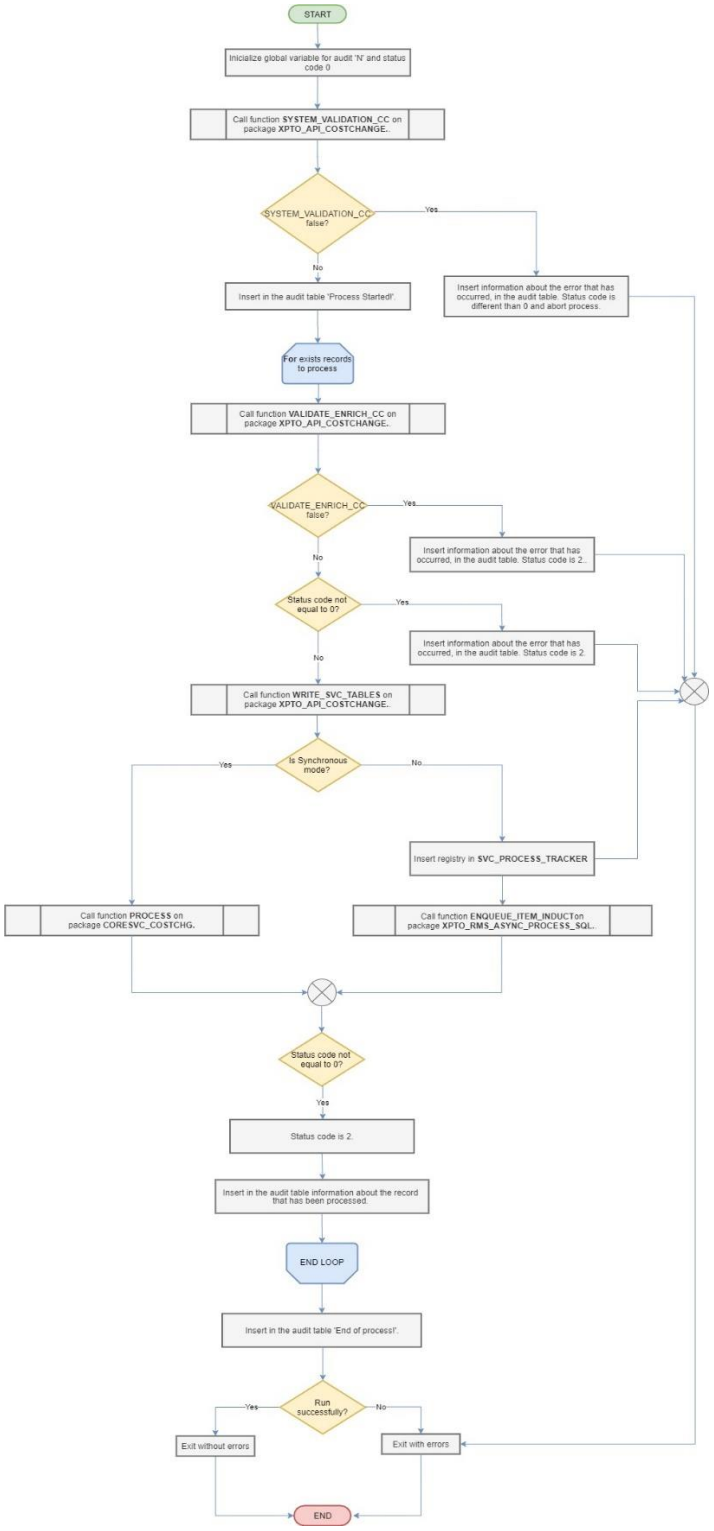


Figura 14- Diagrama do Procedimento da *API Cost Change*

5.2.4 – Janeiro 2020: XXRMS418 *Item Attribute By Date* (PI3)

A interface XXRMS418 consiste numa API de *inbound* que tem como função fazer a comunicação com os restantes sistemas externos isto porque o RMS necessita de receber, desses sistemas externos, valores de UDA[2] e CFA[3] para determinados Itens e isso será gerido pela data. Para isso, surgiu a necessidade de criar esta nova API que espera receber esses atributos para inseri-los posteriormente numa tabela de *staging*. Os registos que irão estar na *staging* serão processados e consumidos, na data apropriada, nas tabelas finais do RMS. Para o objetivo desta API apenas será tratada a parte de receber os atributos dos sistemas externos e inserir esses registos na *staging*. O processo onde os registos são consumidos, numa data apropriada, fará parte do objetivo de outro processamento.

Como forma de introduzir o que serão os diferentes cenários a ter em conta nesta API é importante que um Item é identificado por quatro atributos formulando assim uma chave: *Item/ attribute type/ attribute ID (UDA ou CFA ID)/ effective date.*

Criar um novo *Item Attribute by Date*:

- Deverá ser feito uma chamada da API providenciando um registo Item atribute tendo como tipo de evento *Upsert*. Baseado nesse tipo de evento a API irá procurar na *staging* por um registo que combine com a chave providenciada. A partir do momento em que nada é encontrado, então estamos perante um novo registo, e como tal, terá de ser criado um novo registo na *staging* para poder ser processada mais tarde.

[2] *User Defined Attributes* (UDA) – permite definir atributos e associar os mesmos a itens específicos, itens pertencentes a uma lista de itens ou até mesmo itens de um departamento, classe ou subclasse específica.
[3] *Custom Flex Attributes* (CFA) – é uma estrutura de metadados que pode ser usada para fornecer suporte a personalizações específicas do cliente.

Atualizar um *Item Attribute by Date*:

Se aquilo que se pretender fazer é um *update* a um item atribute, dois cenários podem ocorrer:

- O que é pretendido é mudar o valor de um item atribute já existente, para uma data especifica. Para conseguir isso, um registo *Upsert* terá de ser enviado para o mesmo *Item/ attribute type/ attribute ID/ effective date*, e com o novo valor.

- O que é pretendido é mudar a *effective date* de um *item attribute* que já existe na *staging*. Para conseguir isso, dois registos terão de ser enviados. O primeiro, um registo para *Delete*, com o *mesmo Item/ attribute type/ attribute ID/ effective date*, para eliminar o registo da *staging*. Depois, um registo *Upsert*, com o mesmo *Item/ attribute type/ attribute ID* com a nova *effective date* em que o item atribute deve iniciar.

Eliminar um *Item Attribute by Date:*

- Para se poder cancelar um Item atribute já existente definido para uma data especifica, que ainda não foi atingida, dever-se-á enviar um registo "Delete", como o *Item/ attribute type/ attribute ID/ effective date* pretendido, para apagar o registo da *staging*. É expectável que se o registo for encontrado na *staging*, será apagado da mesma, mas posteriormente será movido, mas para uma tabela de arquivo.

Esta nova API deverá receber como *input* um conjunto de registo de item *attributes* para inserir na tabela *staging* do RMS e é necessário retornar como *output* o mesmo conjunto de registos, onde cada um será enriquecido com um estado do resultado, seja ele um erro ou não. Esta API irá retornar um estado e uma mensagem de erro globais, tabela 6. A ideia é fazer chegar a quem invocou a API um estado geral sobre o processamento da API, sendo que em caso de erros críticos na execução, a mensagem de erro global irá preenchida com informação relevante.

Para o estado global, foram definidos alguns códigos de erro:

Código do Estado Global	Descrição	Significado
0	Sucesso	Toda a operação foi executada com sucesso.
1	Excedeu o número máximo permitido de registos	O número de registos excedeu o limite máximo estabelecido na configuração da API.
2	Processado com erros	A operação foi completada, mas alguns registos falharam e não foram inseridos na tabela de *staging* para *item attributes*.
255	Erro	Toda a operação falhou sendo que nenhum registo foi inserido na tabela de *staging* para *item attributes*.

Tabela 6 - Código de Estado de Erro Global: API Item Attribute by Date

Como se pode receber dos sistemas externos mais do que um *Item attribute*, foi necessário criar códigos de estado ao nível do detalhe, tabela 7. Esta necessidade de criar estes códigos para o detalhe surge no sentido em que, como podemos receber mais do que um *Item attribute* para processar, a API vai tratando um a um e surge a necessidade de a cada processar de cada um dos registos, atribuir um estado ao nível do detalhe para que, no final de todos os registos serem processados, tendo em conta os estados do detalhe, se poder atribuir um estado global, que estão detalhados na tabela acima, tabela 6. Assim sendo, como possível retorno de estados para o detalhe, pode ter-se:

Código do Estado Detalhe	Descrição	Significado
0	Sucesso	Toda a operação foi executada com sucesso.
1	Dados Inválidos	Dados inválidos foram requisitados para o registo processado, resultando num erro específico de falha nos dados. Isto significa que o registo não

		foi inserido na tabela de *staging* para *item attributes*, mas não impede que os outros registos sejam processados.
255	Erro	Um erro ocorreu durante o processar do registo para a tabela de *staging* para *item attributes*. Uma descrição significativa do erro será escrita neste campo.

Tabela 7 - Código de Estado de Erro do Detalhe: API Item Attribute by Date

Relativamente à logica de processamento, cada vez que a operação de *upload* é invocada a seguinte lógica terá de acontecer:

- Realizar validações iniciais:
  - Verificar se o número máximo de registos para *upload* está ou não configurado;
  - Verificar se o número de registos para processar não é excedido;
  - Se alguma das validações acima falhar, retornar o estado/mensagem de erro apropriada – nenhum registo irá ser processado;

- Para cada registo presente no conjunto de *Item attributes by date* recebido é necessário:
  - **Validações base –** Para cada registo, a API irá executar validações básicas aos atributos, para verificar o item a que está relacionado e se os *Item attributes* (UDA ou CFA) existem ambos na base de dados para o tipo especificado. Se não estiverem, irá parar o processamento do registo, e será enriquecido o objeto de *input* com um estado de "*Bad Data*", com código 1, com a respetiva mensagem de erro. Se estiverem, então deverá ser feita uma validação à *effective date* se esta está correta (superior à data atual), e aplica a mesma logica de erro de "*Bad Data*" se não estiver correta.
  - Uma vez que as validações foram executadas com sucesso para o registo, dependendo do tipo do evento, a API irá ter diferentes ações, sendo elas:

- ***Upsert*** (figura 15) – A comunicação de um evento deste tipo assume que a fonte de dados não tem conhecimento dos *item attributes by date* existentes na *staging*. Neste caso, caberá à API detetar qual será a operação: criar um registo na *staging* tendo como *action New* para criação ou *MOD* para *update*. Dependendo da operação identificada para ser processada, a lógica descrita abaixo para as operações *Create* ou *Update* irá ser executada;

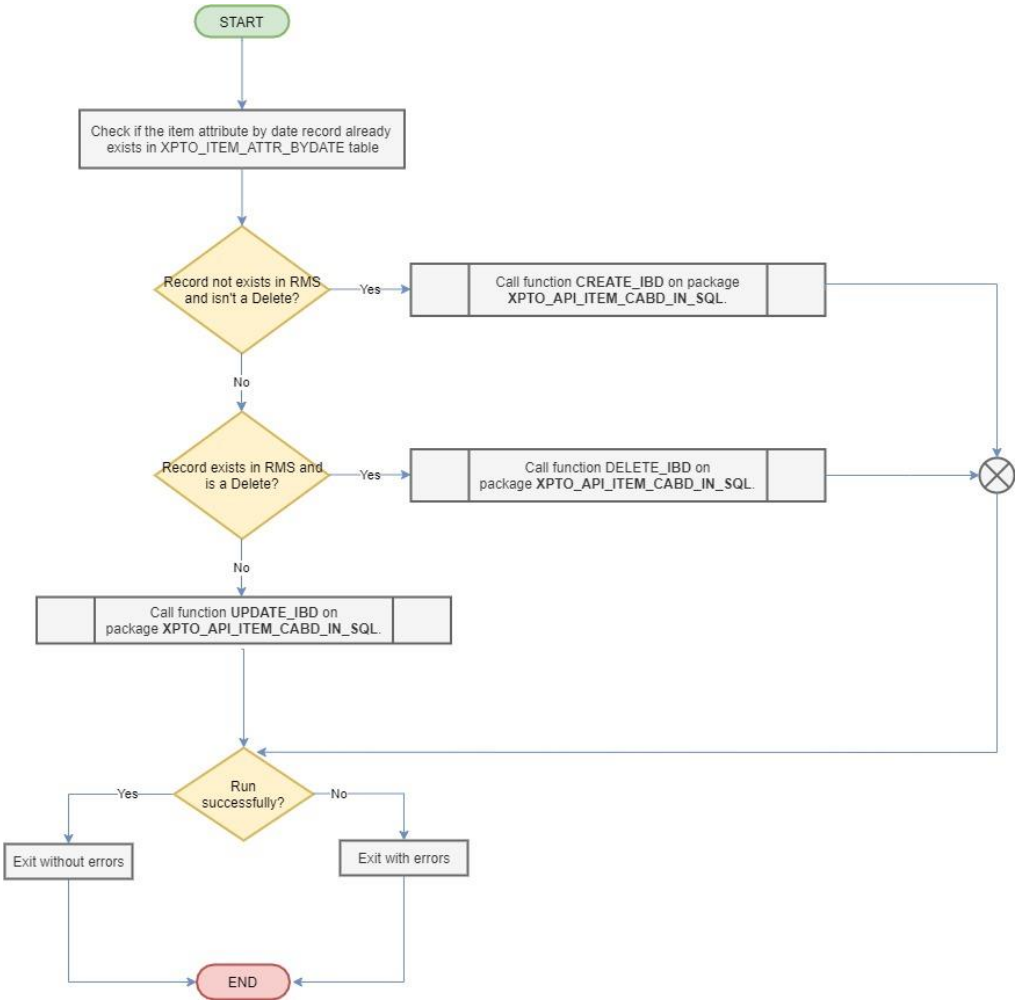


Figura 15- Diagrama da Função *Upsert API Item Attribute By Date*

- **Delete** (figura 16) – A comunicação de um evento deste tipo assume que a fonte de dados tem conhecimento dos *Item attributes by date* existentes na *staging*. Isto significa que existe um *item attribute by date* na *staging*, com *effective date* que ainda não está vencida (está no futuro). Neste caso, o registo do

item attribute a que se refere deverá ser copiado para uma tabela de histórico para ser possível fazer o *tracking* do mesmo. Os registos históricos serão purgados mais tarde no tempo, mas isso já não faz parte do objetivo desta API.

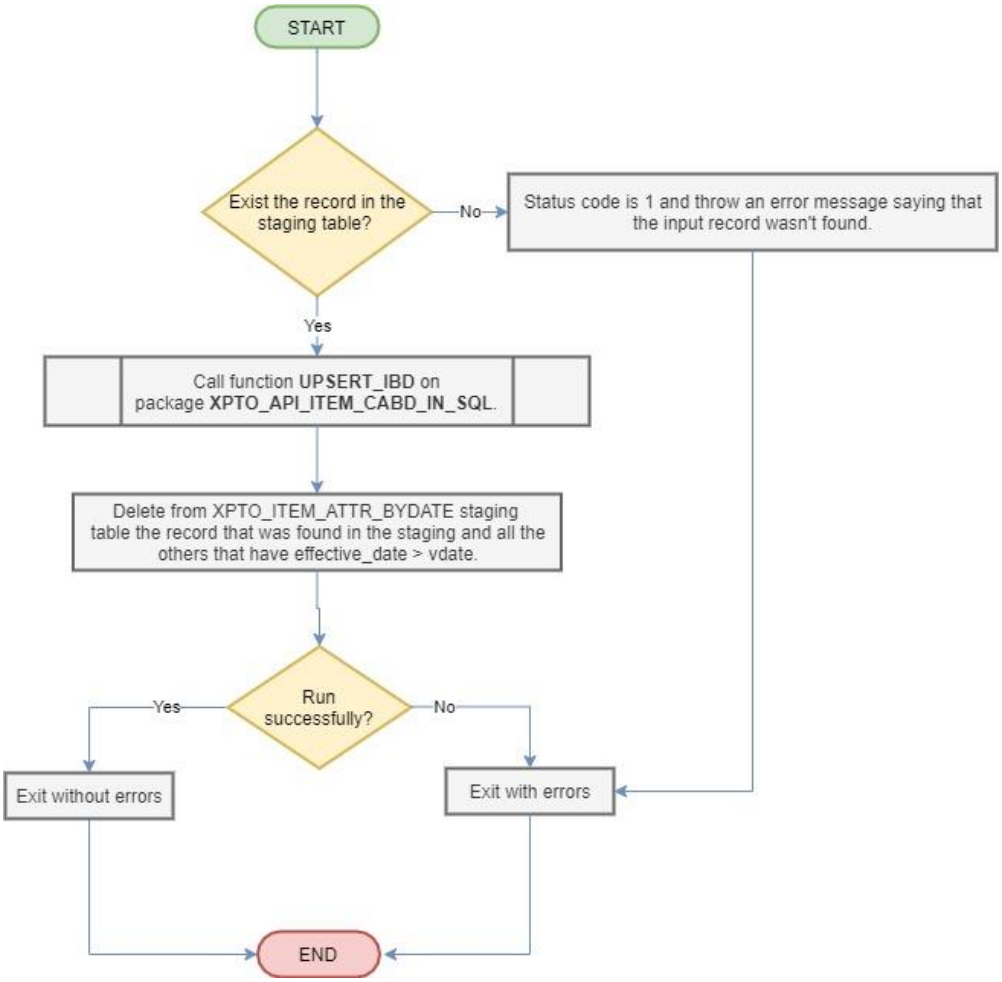


Figura 16- Diagrama da Função *Delete API Item Attribute By Date*

Os tipos de eventos que se seguem serão implementados e expostos pela API, no entanto não será expectável que os sistemas externos invoquem esses tipos de eventos. Eles apenas irão ser usados internamente para suportar o evento Upsert:

- ***Create*** (figura 17) - A comunicação de um evento deste tipo assume que a fonte de dados tem conhecimento de todos os *Item attributes by date* existentes e identifica o registo *Item attributes by date* como sendo algo completamente novo para a *staging* do RMS. Neste caso, um registo deverá ser criado na *staging* tendo *action New*;

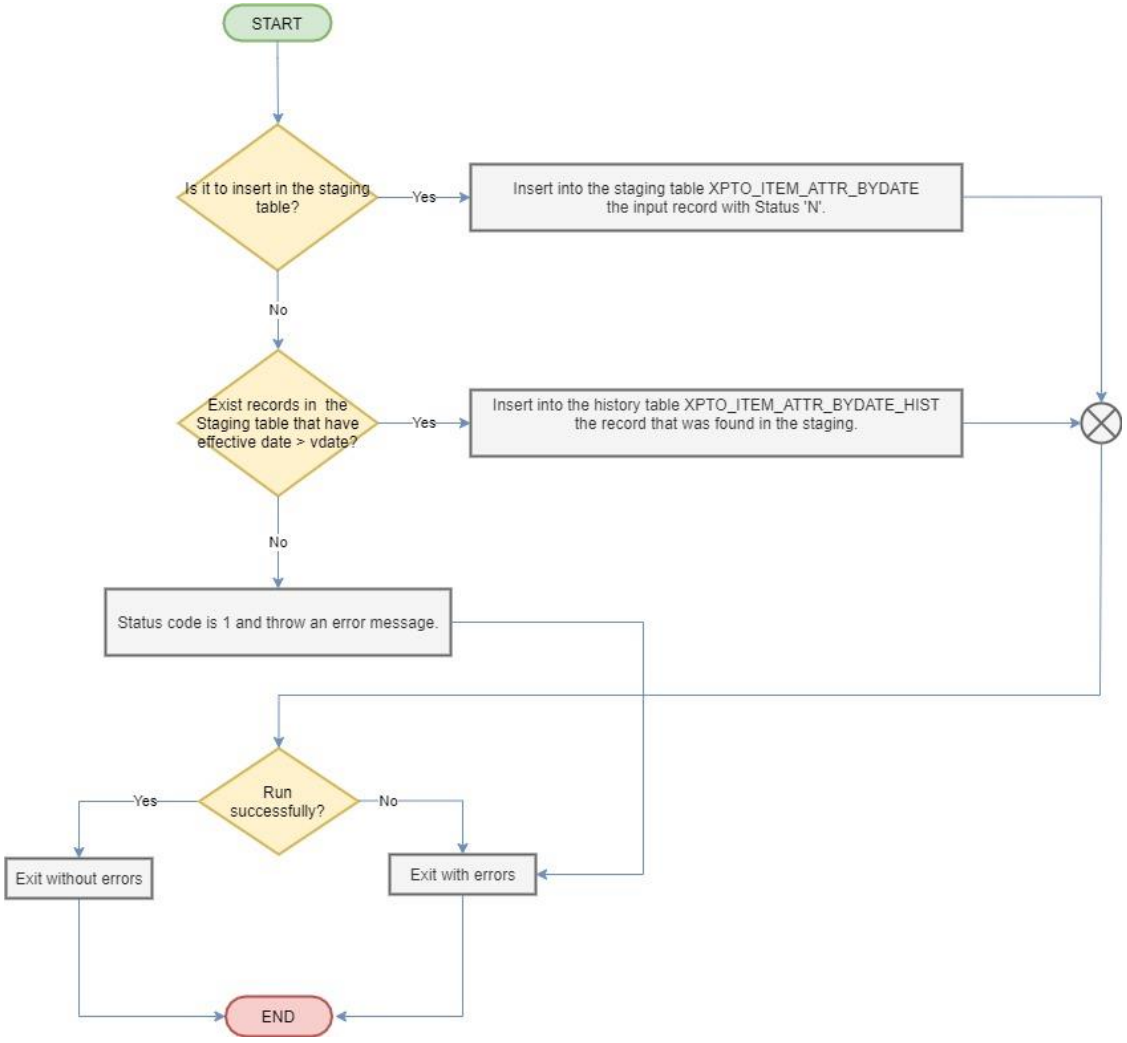


Figura 17- Diagrama da Função *Create API Item Attribute By Date*

- **Update** (figura 18) - A comunicação de um evento deste tipo assume que a fonte de dados tem conhecimento de todos os *item attributes by date* existentes e identifica o registo *item attributes by date* como sendo algo que já existe na *staging* do RMS (quem fizer chamada desta API terá de fornecer o *Item/ attribute type/ attribute ID/ effective date*). Neste caso, irá atualizar o valor do *item attribute* do registo. Se o registo existente estive com estado "E" ele será substituído pelo valor e voltará ao estado *New*.

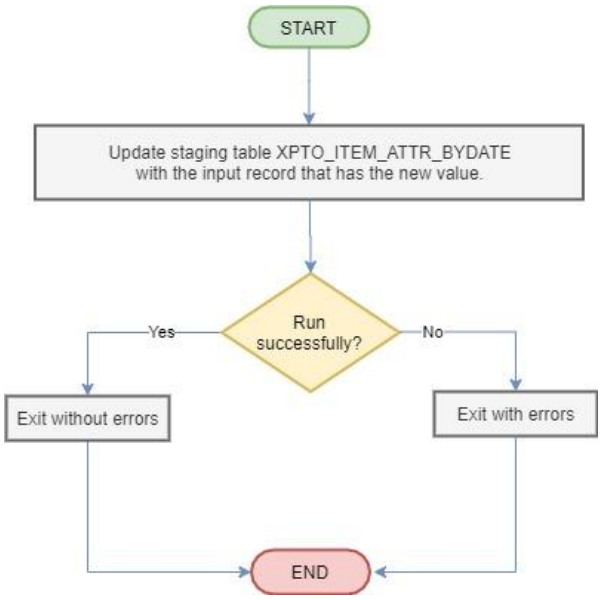


Figura 18- Diagrama da Função *Update API Item Attribute By Date*

Uma vez que o registo foi inserido na *staging*, a API irá enriquecer o objeto de input que contém o registo do *item attribute by date* e irá enriquecer o mesmo com o respetivo código de erro e a respetiva mensagem de erro, quando aplicável, e irá proceder com o processamento do próximo registo.

No caso de um erro critico, que impossibilitará a API de executar (por exemplo um objeto invalido) então a alto nível um código de estado erro deverá ser retornado e uma descrição técnica do erro também. Este código de estado de erro pode ser retornado no caso de o limite de registos ser atingido, ou se um ou mais registos tiverem falhado validações e não foram integrados.

De forma a implementar toda a lógica descrita acima, para a API *Item Attribute By Date* foi necessário criar um novo *package* para conter todo este processamento. Na secção Anexos, concretamente no Anexo B deste documento, encontrar-se-ão os detalhes ao nível do código necessário para a implementação desta API. Segue-se abaixo o diagrama, figura 19, que ilustra todo o processamento, em geral, da API *Item Attribute By Date*. Esse procedimento será criado no novo package e será auxiliado por funções como as descritas acima. Para além disso irá conter mais um conjunto de validações extra que irão ajudar a validar se os parâmetros estão bem preenchidos. Para isso foram criadas funções especificas, restritas deste procedimento que só serão usadas dentro das funções do procedimento, tais como Validações, *Create, Update, Upsert e Cancel*, sendo

que só é expectável que quem fizer uma chamada a esta API apenas utilize *Upsert* e *Cancel*. Para esta API não foi necessário criar um objeto visto que este já se encontrava criado.

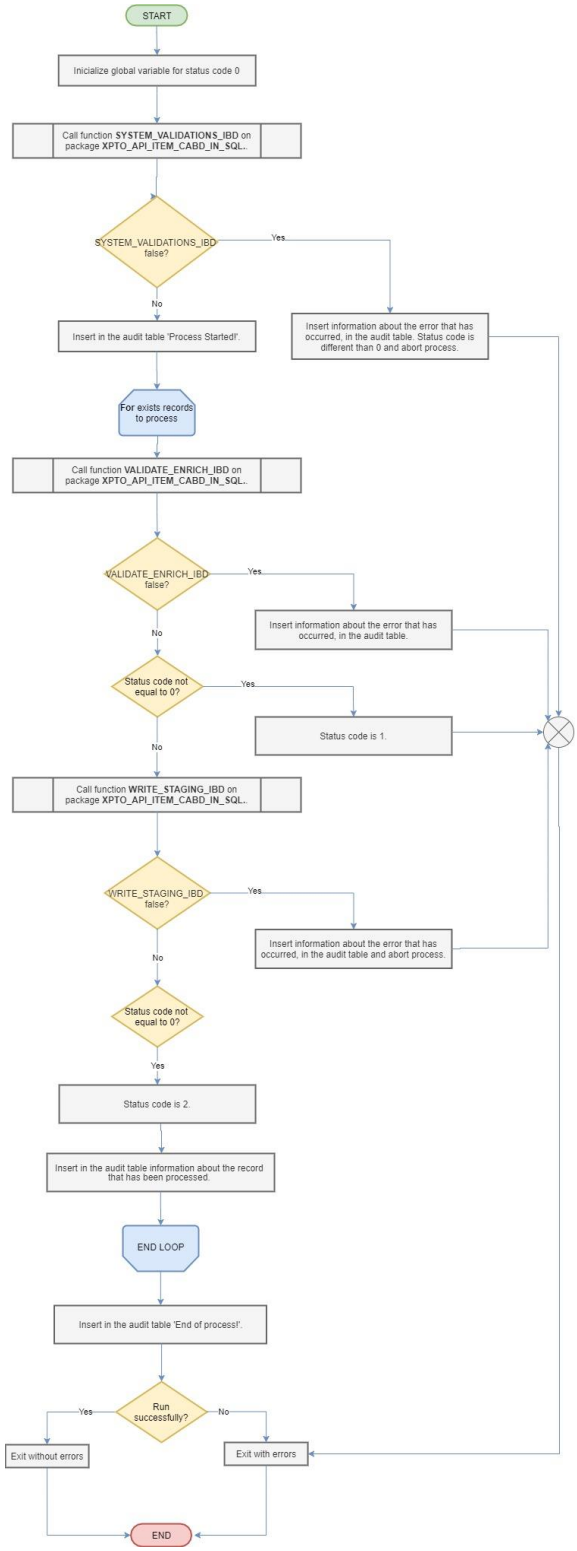


Figura 19- Diagrama do procedimento *UPLD_ITEM_BYDATE_TO_STG*

6 – Conclusão

6.1 – Reflexão do trabalho

Com este trabalho foi possível ter o primeiro contacto direto em projetos de desenvolvimento de *software*, para um determinado cliente, e também ter o primeiro contacto com as metodologias de gestão de projeto, como o Scrum, usando práticas do Agile. Derivado da utilização dessa metodologia, o projeto foi evoluindo em linha com as expectativas, fazendo chegar ao cliente todas as tarefas planeadas por sprint, com sucesso e com a menor taxa de bugs possível, de forma a que o cliente ficasse satisfeito com o produto entregue.

Foram adquiridas diversas competências em resultado das reuniões diárias, tais como a capacidade de fazer um ponto de situação juntamente com toda a equipa, explicar o que foi feito até ao momento, o que falta fazer, como se irá proceder para continuar, tirar dúvidas sobre componentes menos bem entendidas, entre outras.

Foram também adquiridos conhecimentos relacionados com a implementação de sistemas ERP, na medida em que, foi necessário customizar alguns módulos de modo a satisfazer as necessidades do cliente, o que vai de encontro a um fator critico de sucesso, *Change Managment,* mostrando que o RMS não tinha os processos compatíveis com a nova estrutura que o cliente XPTO queria implementar, e como tal foi necessário forçar a reengenharia de processos de modo a se encontrar soluções para satisfazer as novas necessidades do cliente.

O recurso a customizações no RMS envolveu trabalhar diretamente em base de dados, trabalhando com PL/SQL, e ainda com *batchs*, trabalhando com linguagem C. Nesse sentido, foram adquiridos conhecimentos técnicos de PL/SQL e de linguagem C derivado dessa mesma experiência na customização do RMS.

Por fim, e fazendo um balanço de todo o projeto, o cliente manifestou-se satisfeito nas reuniões de final de PI, a taxa de bugs foi sempre bastante reduzida e como tal pode dizer-se que foi um projeto bem-sucedido.

6.2 – Reflexão Pessoal sobre o trabalho

Ao longo de todo o estágio, e em específico ao longo de todo este trabalho, foram variadíssimas as coisas positivas. Estar num projeto desta dimensão, com um cliente estrangeiro, onde inúmeras vezes se receberam visitas do cliente para realizar alguns testes, para se fazerem reuniões onde se falava sobre como tinha corrido um determinado PI em específico, o podermos deixar opções de melhoria em algumas abordagens, o podermos falar sobre aquilo que gostaríamos que continuasse e estivesse a ser bem feito, etc. Tudo isto permitiu criar um bom ambiente de trabalho, não só entre equipa de desenvolvimento no ITC, como também entre os elementos ITC e o cliente.

Ao nível da gestão do projeto no ITC, a pessoa responsável por gerir este projeto em específico, em termos de trabalho sempre nos deixou à vontade com qualquer tipo de situação. Foi uma pessoa bastante ativa, fazia reuniões com alguma frequência com a equipa de desenvolvimento, para podermos falar dos nossos bloqueios com determinadas tarefas para que pudéssemos ter ajuda dos restantes colegas de modo a não se perder demasiado tempo com esses bloqueios e continuarmos os desenvolvimentos.

Ao nível da equipa de desenvolvimento, todos os colegas sempre foram bastante amigáveis. Nunca disseram não quando era necessário algum esclarecimento, mesmo se estivessem, como por vezes estavam, apertados em termos de tempo para completar as suas tarefas. Criou-se uma excelente relação entre a equipa o que fez crescer um ambiente bastante descontraído, mas acima de tudo responsável.

No geral, foi uma experiência bastante positiva. Aprendi imenso estando neste projeto desde capacidade técnica ao nível de PL/SQL, até à parte de aprender a lidar com o cliente, falar com o cliente, etc.

6.3 – Reflexão sobre dificuldades encontradas

Quanto às dificuldades encontradas, durante todo o trabalho estas foram várias associadas aos desenvolvimentos fruto da inexperiência a trabalhar com o RMS ou até mesmo por algumas indefinições por parte da equipa funcional. Entrando em detalhe em alguns casos específicos, as dificuldades encontradas foram as seguintes:

- Na tarefa de construção da primeira API para este projeto, API *Price Changes*, a maior dificuldade encontrada foi o facto de não existir uma estrutura devidamente pensada para abordar a construção de todas as API. Foi necessário por várias vezes intervir neste desenvolvimento após ter sido entregue. Houve vários bugs abertos pelo cliente reportando a necessidade de corrigir alguma coisa, ou porque estava em falta algum procedimento;
- Na tarefa de construção da API Cost Change, a maior dificuldade foi encontrar no RMS funções que já existiam e que se pudessem reutilizar no novo desenvolvimento. Foi uma tarefa complicada, que levou imensos dias. Foi necessário fazer um debug exaustivo no RMS em diversos packages para perceber quais funções se podiam usar e qual era o seu comportamento. Ninguém sabia ao certo qual seria o package onde estariam as funções e isso sem dúvida que foi uma dificuldade, mas que no final resultou numa capacidade gigante de aprender a fazer debug no RMS por processos que já existem de base e que podem ser reutilizados.

Referências

Agile Alliance. (2020). 12 Principles Behind the Agile Manifesto. Acedido em maio 10, 2020, em https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

Agile Alliance. (2020). The Agile Manifesto. Acedido em maio 10, 2020, em https://www.agilealliance.org/agile101/the-agile-manifesto/

Arvidsson, J., & Kojic, D. (2017). *Critical Success Factors in ERP Implementation - The Perspective of the Procurement System User.* Master Thesis in Business Administration. Jönköping International Business School, Suécia.

Oracle. (2019). *Oracle® Retail Merchandising Implementation Guide Release 16.0*.

Oracle. (2020). *What is ERP?*. Acedido em maio 10, 2020, em https://www.oracle.com/applications/erp/what-is-erp.html#link1

Scrumportugal. (n.d.). Agile Scrum Roles – Scrum Master, Product Owner and Development Team. Acedido em maio 10, 2020, em https://www.exin.com/article/agile-scrum-roles-scrum-master-product-owner-and-development-team?language_content_entity=en

Scrumportugal. (n.d.). Agile Sprint. Acedido em maio 10, 2020, em http://www.scrumportugal.pt/agile-sprint/

Scrumportugal. (n.d.). Kanban Board. Acedido em maio 10, 2020, em http://www.scrumportugal.pt/kanban-board/

Scrumportugal. (n.d.). Scrum - Guia para scrum em português. Acedido em maio 10, 2020, em http://www.scrumportugal.pt/scrum/

Anexos

Anexo A – Planeamento do Trabalho

Tarefa	Tipo de tarefa	Descrição
1	Levantamento de informação	Levantamento da informação resultante de pesquisas realizadas em artigos, publicações. Interação com os elementos da empresa acerca do projeto, do glossário usado na empresa, de como funciona todo o processo desde o momento em que se inicia um projeto até ao momento em que é entrega ao cliente, entre outros.
2	Escrita da pré-dissertação	Documentar o tema a abordar, a descrição do objetivo do estágio profissional.
3	Escrita da dissertação	Descrição de todo o processo decorrido durante o período do estágio profissional, revisão da tarefa 2.
4	Revisão da dissertação	Revisão da dissertação.

Tabela 8 - Planeamento Detalhado do Trabalho

Tarefas	Mar		Abr		Maio		Jun		Jul		Ago		Set		Out		Nov		Dez		Jan	
1																						
2																						
3																						
4																						

Tabela 9 - Calendarização do Trabalho

Anexo B – Apêndice de cada tarefa

RB05 *Item Delete Pending* (PI1)

Batch

Este *batch XPTO_ITEM_DELPEND.pc, representado na figura 20*, construído com base em linguagem C, irá fazer uma chamada de um *package* da base de dados RMS que irá conter toda a lógica de processamento e irá dar como *output* uma mensagem de erro, caso haja algum erro, e um código de erro, que em caso de ter valor 0, significa que tudo correu bem.

```
159
160  int process()
161  {
162     char *function = "process";
163     long   pl_plsql_proc_failed = 0;
164     char   ps_plsql_proc_message[256];
165
166      EXEC SQL EXECUTE
167
168      DECLARE
169
170        L_error_message     VARCHAR2(255) := NULL;
171        L_program_error   int := 0;
172      BEGIN
173
174          IF XPTO_ITEM_DELPEND_SQL.PROCESS_ITEM_DELPEND(L_error_message,
175                                                        L_program_error,
176                                                        I_restart_driver_name => :ps_restart_driver_name,
177                                                        I_restart_num_threads => :ps_restart_num_threads,
178                                                        I_restart_thread_val  => :ps_restart_thread_val) = FALSE
179
180           THEN
181              :pl_plsql_proc_failed := 1;
182              :ps_plsql_proc_message := L_error_message;
183              :pi_program_error := L_program_error;
184         END IF;
185      END;
186      END-EXEC;
187
188    if (pi_program_error)
189      {
190        sprintf(err_data, "No records have been processed. Table DAILY_PURGE is locked.");
191        WRITE_ERROR(SQLCODE,function,"",err_data);
192        return(NON_FATAL);
193      }
194
195    if (SQL_ERROR_FOUND)
196      {
197        sprintf(err_data, "STORED FUNCTION CALL FAILED: function=%s, sqlcode=%d",
198            "process", SQLCODE);
199        LOG_MESSAGE(err_data);
200        WRITE_ERROR(SQLCODE,function,"",err_data);
201        return(-1);
202      }
203    if (pl_plsql_proc_failed)
204      {
205        sprintf(err_data, "INTERNAL STORED FUNCTION FAILED: function=%s, error=%s",
206            "XPTO_ITEM_DELPEND_SQL.PROCESS_ITEM_DELPEND", ps_plsql_proc_message);
207        LOG_MESSAGE(err_data);
208        WRITE_ERROR(RET_PROC_ERR,function,"",err_data);
209        return(-1);
210      }
211
212      return(0);
213
214     return(0);
215
216  } /* end of process */
```

Figura 20- XPTO_ITEM_DELPEND.pc

Package de Base de Dados

O código que se seguirá dirá respeito à parte *spec* e *body* da construção de um package PL/SQL. Existem sempre duas partes, um *spec* e um *body*. Do *spec* apenas faz parte a declaração das funções, que podem ser chamadas por outros *packages* ou procedimentos, acompanhada com uma breve descrição do que esta faz, e ainda também fazem parte todas as declarações de variáveis globais, que podem ser usadas em todas as funções do *package* em que são instanciadas. Quanto ao *body*, é neste onde se coloca toda a lógica necessária para a construção da função.

XPTO_ITEM_DELPEND.pks

```
CREATE OR REPLACE PACKAGE XPTO_ITEM_DELPEND_SQL AUTHID CURRENT_USER AS

  /*****************************************************************************/
  /* CREATE DATE - June 2019                                                   */
  /* CREATE USER - Joao Vilaverde                                              */
  /*                                                                           */
  /* PROJECT     - XPTO                                                        */
  /* DESCRIPTION - Package for RB05                                            */
  /*****************************************************************************/

  ---------------------------------------------------------------------------
  -- Public Constants
  ---------------------------------------------------------------------------
  CONST$PackageName     CONSTANT VARCHAR2(30) := 'XPTO_ITEM_DELPEND_SQL';
  CONST$StatusSupprimee CONSTANT INT := 5;

  ---------------------------------------------------------------------------
  -- Public Functions
  ---------------------------------------------------------------------------

  ---------------------------------------------------------------------------
  --Function Name : PROCESS_ITEM_DELPEND
  --Purpose       : Insert records on the Daily Purge table after it evaluates
  --                every Item that is in the purging conditions according to
  --                XPTOs business rules.
  ---------------------------------------------------------------------------
  FUNCTION PROCESS_ITEM_DELPEND(O_error_message       IN OUT VARCHAR2,
                                O_program_error       IN OUT int,
                                I_restart_driver_name IN v_restart_dept.driver_name%type DEFAULT NULL,
                                I_restart_num_threads IN v_restart_dept.num_threads%TYPE DEFAULT NULL,
                                I_restart_thread_val  IN v_restart_dept.thread_val%TYPE DEFAULT NULL)
    RETURN BOOLEAN;
  ---------------------------------------------------------------------------
END XPTO_ITEM_DELPEND_SQL;
/
```

Figura 21- XPTO_ITEM_DELPEND.pks

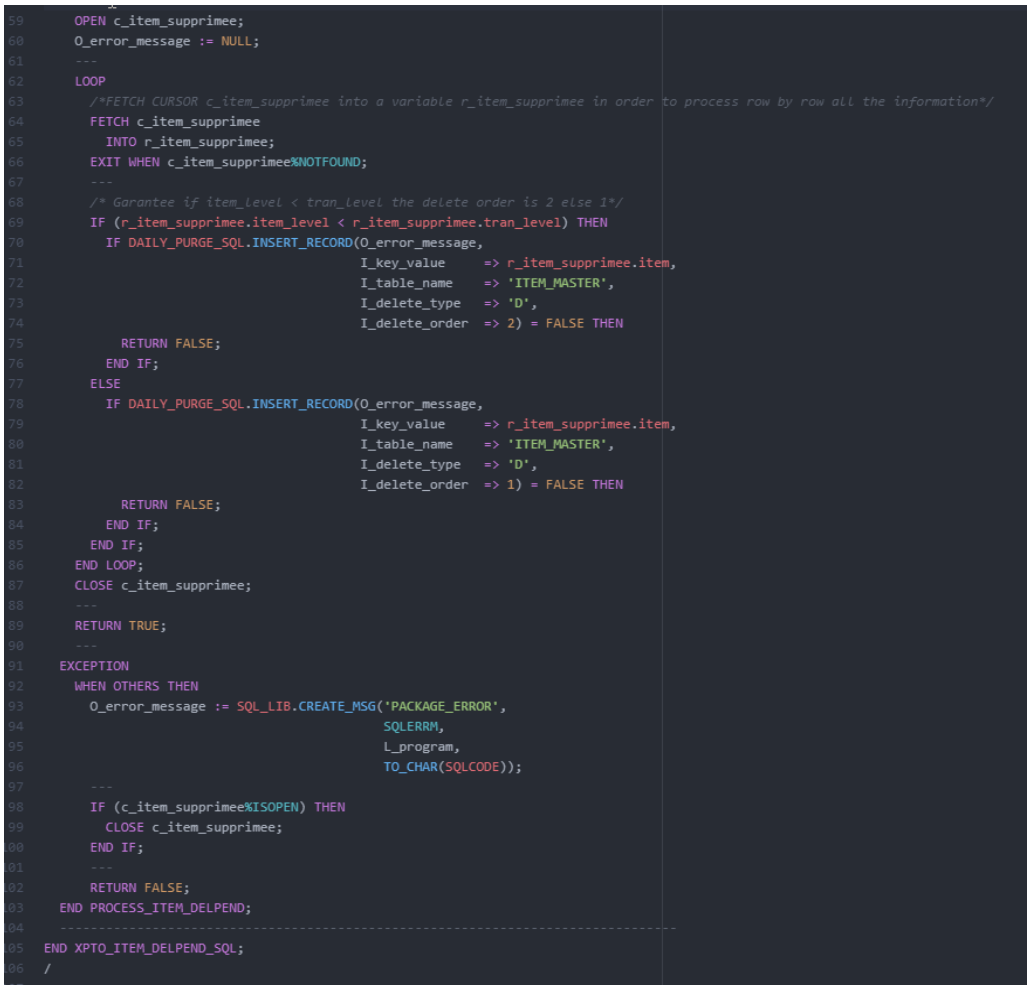
Como se pode verificar no *spec* do *package XPTO_ITEM_DELPEND.pks*, pode ver-se um cabeçalho, onde está presente a identificação do autor que criou o *package*, a data, o projeto e uma descrição do propósito da criação do *package*, neste caso será para suportar o RB05. Para além do cabeçalho, podemos ver que existe a declaração de duas variáveis globais, que poderão ser usadas para substituir o uso dos respetivos valores aos quais estas são atribuídas, sendo esta uma boa prática de programação porque sempre que for necessário mudar o valor destas variáveis, que podem estar a ser usadas

em diferentes partes do código, basta irmos ao *spec* e mudar esse mesmo valor sem ter de andar a fazê-lo ao longo de todo o código com risco de esquecer alguma parte. Por fim, existe também uma declaração das funções que terão visibilidade para serem usadas noutros *packages*, ou até mesmo procedimentos de base de dados. A cada função deve acompanhar uma respetiva descrição, bem como todas as variáveis de *INPUT* (IN) e *OUTPUT* (OUT), tal como podemos ver na Figura 21.

XPTO_ITEM_DELPEND.pkb

```
CREATE OR REPLACE PACKAGE BODY XPTO_ITEM_DELPEND_SQL AS

  ----------------------------------------------------------------------------
  -- Public functions implementation
  ----------------------------------------------------------------------------

  ----------------------------------------------------------------------------
  FUNCTION PROCESS_ITEM_DELPEND(O_error_message       IN OUT VARCHAR2,
                                O_program_error       IN OUT int,
                                I_restart_driver_name IN v_restart_dept.driver_name%type DEFAULT NULL,
                                I_restart_num_threads IN v_restart_dept.num_threads%TYPE DEFAULT NULL,
                                I_restart_thread_val  IN v_restart_dept.thread_val%TYPE DEFAULT NULL)
    RETURN BOOLEAN IS
    ---
    L_program               VARCHAR2(100) := CONST$PackageName || '.PROCESS_ITEM_DELPEND';
    L_value_status_supprimee INT := CONST$StatusSupprimee;
    ---
    /*CURSOR to lock table daily purge*/
    CURSOR c_daily_purge_lock IS
      SELECT 1 FROM daily_purge FOR UPDATE NOWAIT;
    ---
    /*CURSOR to get the item that have status Supprim?e*/
    CURSOR c_item_supprimee IS
      SELECT uil.item, itm.item_level, itm.tran_level
        FROM xpto_mom_dvm xmd,
             uda_item_lov   uil,
             item_master    itm,
             v_restart_dept rv
       WHERE to_char(uil.uda_id) = xmd.value_1
         AND itm.item = uil.item
         AND xmd.func_area = 'UDA'
         AND xmd.parameter = 'LIFECYCLE_STATUS'
         AND uil.uda_value = L_value_status_supprimee
         AND rv.driver_name = I_restart_driver_name
         AND rv.driver_value = itm.dept
         AND rv.num_threads = TO_NUMBER(I_restart_num_threads)
         AND rv.thread_val = TO_NUMBER(I_restart_thread_val)
       ORDER BY itm.dept;
    ---
    r_item_supprimee c_item_supprimee%ROWTYPE;
    ---
  BEGIN
    ---
    BEGIN
      ---
      /*Daily Purge table is locked, will show a error message, in error logs with that information*/
      OPEN c_daily_purge_lock;
      CLOSE c_daily_purge_lock;
    EXCEPTION
      WHEN OTHERS THEN
        O_error_message := SQL_LIB.CREATE_MSG('DAILY_PURGE TABLE',
                                              'No records have been processed. Table DAILY_PURGE is locked.',
                                              L_program,
                                              TO_CHAR(SQLCODE));
        O_program_error := 1;
        RETURN FALSE;
    END;
    ---
```

Figura 22- XPTO_ITEM_DELPEND.pkb (parte 1)



```
    OPEN c_item_supprimee;
    O_error_message := NULL;
    ---
    LOOP
      /*FETCH CURSOR c_item_supprimee into a variable r_item_supprimee in order to process row by row all the information*/
      FETCH c_item_supprimee
        INTO r_item_supprimee;
      EXIT WHEN c_item_supprimee%NOTFOUND;
      ---
      /* Garantee if item_level < tran_level the delete order is 2 else 1*/
      IF (r_item_supprimee.item_level < r_item_supprimee.tran_level) THEN
        IF DAILY_PURGE_SQL.INSERT_RECORD(O_error_message,
                                         I_key_value     => r_item_supprimee.item,
                                         I_table_name    => 'ITEM_MASTER',
                                         I_delete_type   => 'D',
                                         I_delete_order  => 2) = FALSE THEN
          RETURN FALSE;
        END IF;
      ELSE
        IF DAILY_PURGE_SQL.INSERT_RECORD(O_error_message,
                                         I_key_value     => r_item_supprimee.item,
                                         I_table_name    => 'ITEM_MASTER',
                                         I_delete_type   => 'D',
                                         I_delete_order  => 1) = FALSE THEN
          RETURN FALSE;
        END IF;
      END IF;
    END LOOP;
    CLOSE c_item_supprimee;
    ---
    RETURN TRUE;
    ---
  EXCEPTION
    WHEN OTHERS THEN
      O_error_message := SQL_LIB.CREATE_MSG('PACKAGE_ERROR',
                                            SQLERRM,
                                            L_program,
                                            TO_CHAR(SQLCODE));
      ---
      IF (c_item_supprimee%ISOPEN) THEN
        CLOSE c_item_supprimee;
      END IF;
      ---
      RETURN FALSE;
  END PROCESS_ITEM_DELPEND;
  ----------------------------------------------------------------------------
END XPTO_ITEM_DELPEND_SQL;
/
```

Figura 23- XPTO_ITEM_DELPEND.pkb (parte 2)

Teste Unitário

Cenário Positivo - Inserir os dados corretamente na tabela de Purga para apagar.

Para a execução do cenário de teste usaram-se os seguintes dados:

UDA_ITEM_LOV

	ITEM	UDA_ID	UDA_VALUE	CREATE_DATETIME	LAST_UPDATE_DATETIME
1	82043021	280	5	27/05/2019 14:56	27/05/2019 14:56
2	81979692	280	5	27/05/2019 14:57	27/05/2019 14:57
3	45339557	280	5	01/07/2019 11:31	01/07/2019 11:31
4	2007000000038	280	5	01/07/2019 11:31	01/07/2019 11:31
5	45339554	280	5	28/06/2019 17:46	28/06/2019 17:46
6	2007000000014	280	5	28/06/2019 17:46	28/06/2019 17:46

Tabela 10 - UDA_ITEM_LOV

ITEM MASTER

	ITEM	ITEM_NUMBER_TYPE	FORMAT_	PREFIX	ITEM_PARENT	ITEM_GRA	PACK_IND	ITEM_LEVEL	TRAN_LEVEL	I
1	82043021	MANL					N	1	1	N
2	81979692	MANL					N	1	1	N
3	45339557	MANL					N	1	2	N
4	45339554	MANL					N	1	2	N
5	2,007E+12	EAN13			45339557		N	2	2	N
6	2,007E+12	EAN13			45339554		N	2	2	N

Tabela 11 - ITEM MASTER

UDA_VALUES

	UDA_ID	UDA_VALUE	UDA_VALUE_DESC
1	280	1	Initialisée
2	280	2	Active services internes
3	280	3	Active commerce
4	280	4	En suppression
5	280	5	Supprimée

Tabela 12 - UDA_VALUES

UDA

	UDA_ID	UDA_DESC	MODULE	DISPLAY_TYPE	DATA_TYPE	DATA_LENGTH	SINGLE_VALUE_INDI
1	280	LMPL Life Cycle status	ITEM	LV	ALPHA		Y

Tabela 13 - UDA

Como se pode ver pelos dados usados, todos os itens têm *UDA value* 5, que corresponde o estado *Supprimée*. Todos os itens usados, têm relação *Item_level / Tran_level*. Existe a UDA configurada na base de dados para o Ciclo de vida de um Item. Assim sendo tudo está de acordo com os critérios necessários para se poder testar a inserção destes itens para delete.

Após correr o *batch* o resultado foi o seguinte:

```
35 select *
36   from daily_purge
37  where key_value in ('82043021',
38                      '81979692',
39                      '45339554',
40                      '45339557',
41                      '2007000000014',
42                      '2007000000038')
43  order by key_value, delete_order desc;
```

	KEY_VALUE	TABLE_NAME	DELETE_TYPE	DELETE_ORDER
1	2007000000014	ITEM_MASTER	D	1
2	2007000000038	ITEM_MASTER	D	1
3	45339554	ITEM_MASTER	D	2
4	45339557	ITEM_MASTER	D	2
5	81979692	ITEM_MASTER	D	1
6	82043021	ITEM_MASTER	D	1

Tabela 14 - Daily_Purge

Tal como era de esperar, todos os 6 *itens* foram marcados para serem apagados do RMS, visto que todos eles tinham *UDA_VALUE* 5, ou seja, possuíam estado *Supprimée* que os torna válidos para serem marcados na base de dados para delete, após a execução do *batch*. Podemos também verificar que a *DELETE_ORDER* se encontra correta, visto que os *itens* que possuíam *item_level* menor que *transaction level* teriam de ter *DELETE_ORDER* 2 e os restantes *DELETE_ORDER* 1, tal como podemos verificar na item master, os itens 3 e 4 possuem *item level* menor que o *transaction level* e estão devidamente categorizados na tabela de purga com *DELETE_ORDER* 2.

Posto isto, pode-se concluir que o teste foi positivo.

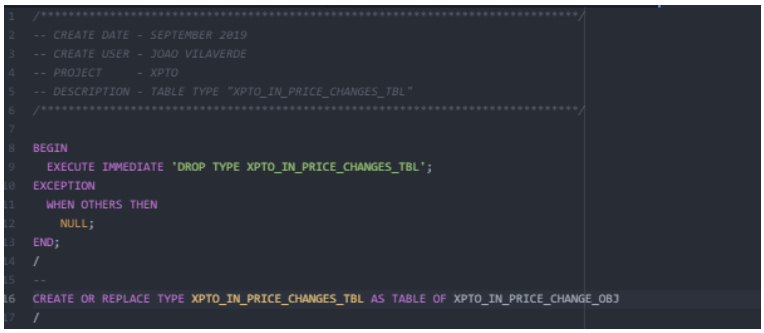
XXRMS203 API *Price Change Inbound* (PI2)

Objeto

Para a construção da API *Price Changes* foi necessário criar um objeto para poder receber, como também enviar, a estrutura de dados recebida por parte dos sistemas externos. Assim sendo para podermos construir um objeto também temos de construir um construtor, que é aquilo que nos permitirá utilizar na referência ao objeto durante o código. Dentro do mesmo objeto podemos ter vários tipos de construtores, que vão de encontro às informações que precisamos de usar da estrutura de dados. A construção

do objeto inicia-se com a estrutura total dos dados, todos os atributos usados irão estão listados logo no início do objeto, e de seguida segue-se então a declaração dos construtores, tal como poder-se-á ver abaixo.

Antes da criação propriamente dita do Objeto é necessário criar um tipo que irá referenciar o objeto. Após essa criação é que se cria o Objetivo do tipo dessa tabela

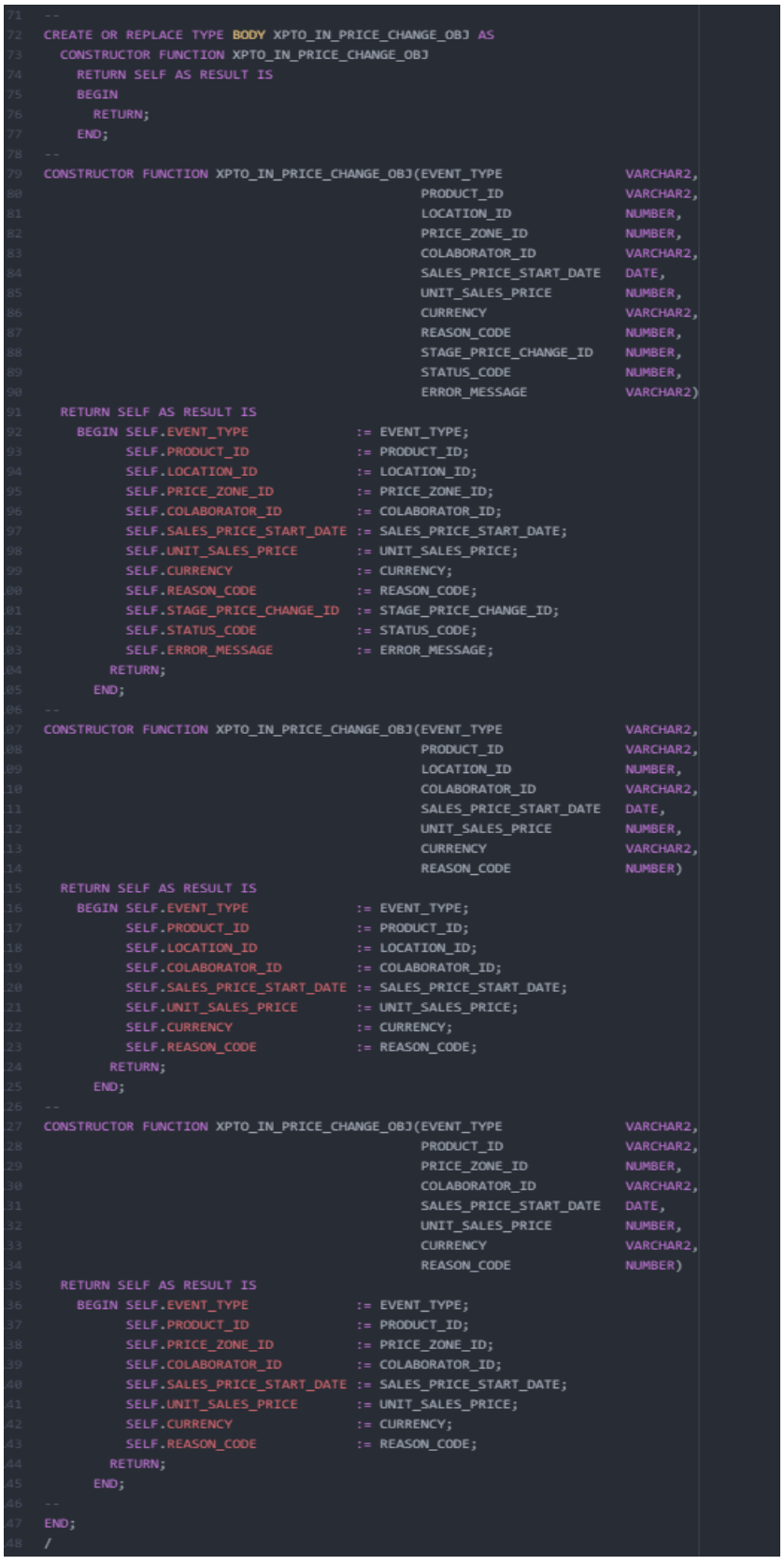


```
/*****************************************************************************/
-- CREATE DATE - SEPTEMBER 2019
-- CREATE USER - JOAO VILAVERDE
-- PROJECT     - XPTO
-- DESCRIPTION - TABLE TYPE "XPTO_IN_PRICE_CHANGES_TBL"
/*****************************************************************************/

BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE XPTO_IN_PRICE_CHANGES_TBL';
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
--
CREATE OR REPLACE TYPE XPTO_IN_PRICE_CHANGES_TBL AS TABLE OF XPTO_IN_PRICE_CHANGE_OBJ
/
```

Figura 24- XPTO_IN_PRICE_CHANGE_TBL

```
/*****************************************************************************/
-- CREATE DATE - SEPTEMBER 2019
-- CREATE USER - JOAO VILAVERDE
-- PROJECT     - XPTO
-- DESCRIPTION - OBJECT TYPE "XPTO_IN_PRICE_CHANGE_OBJ"
/*****************************************************************************/
BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE XPTO_IN_PRICE_CHANGE_OBJ FORCE';
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
--
--
CREATE OR REPLACE TYPE XPTO_IN_PRICE_CHANGE_OBJ AS OBJECT
(
  EVENT_TYPE              VARCHAR2(6),
  PRODUCT_ID              VARCHAR2(25),
  LOCATION_ID             NUMBER(10),
  PRICE_ZONE_ID           NUMBER(10),
  COLABORATOR_ID          VARCHAR2(250),
  SALES_PRICE_START_DATE  DATE,
  UNIT_SALES_PRICE        NUMBER(20,4),
  CURRENCY                VARCHAR2(3),
  REASON_CODE             NUMBER(6),
  STAGE_PRICE_CHANGE_ID   NUMBER(15),
  STATUS_CODE             NUMBER(3),
  ERROR_MESSAGE           VARCHAR2(1000),

--
  CONSTRUCTOR FUNCTION XPTO_IN_PRICE_CHANGE_OBJ RETURN SELF AS RESULT,
--
  CONSTRUCTOR FUNCTION XPTO_IN_PRICE_CHANGE_OBJ(EVENT_TYPE              VARCHAR2,
                                                PRODUCT_ID              VARCHAR2,
                                                LOCATION_ID             NUMBER,
                                                PRICE_ZONE_ID           NUMBER,
                                                COLABORATOR_ID          VARCHAR2,
                                                SALES_PRICE_START_DATE  DATE,
                                                UNIT_SALES_PRICE        NUMBER,
                                                CURRENCY                VARCHAR2,
                                                REASON_CODE             NUMBER,
                                                STAGE_PRICE_CHANGE_ID   NUMBER,
                                                STATUS_CODE             NUMBER,
                                                ERROR_MESSAGE           VARCHAR2)
    RETURN SELF AS RESULT,
--
  CONSTRUCTOR FUNCTION XPTO_IN_PRICE_CHANGE_OBJ(EVENT_TYPE              VARCHAR2,
                                                PRODUCT_ID              VARCHAR2,
                                                LOCATION_ID             NUMBER,
                                                COLABORATOR_ID          VARCHAR2,
                                                SALES_PRICE_START_DATE  DATE,
                                                UNIT_SALES_PRICE        NUMBER,
                                                CURRENCY                VARCHAR2,
                                                REASON_CODE             NUMBER)
    RETURN SELF AS RESULT,
--
  CONSTRUCTOR FUNCTION XPTO_IN_PRICE_CHANGE_OBJ(EVENT_TYPE              VARCHAR2,
                                                PRODUCT_ID              VARCHAR2,
                                                PRICE_ZONE_ID           NUMBER,
                                                COLABORATOR_ID          VARCHAR2,
                                                SALES_PRICE_START_DATE  DATE,
                                                UNIT_SALES_PRICE        NUMBER,
                                                CURRENCY                VARCHAR2,
                                                REASON_CODE             NUMBER)
    RETURN SELF AS RESULT
--
)
;
/
```

Figura 25- XPTO_IN_PRICE_CHANGE_OBJ (parte 1)



```
--
CREATE OR REPLACE TYPE BODY XPTO_IN_PRICE_CHANGE_OBJ AS
  CONSTRUCTOR FUNCTION XPTO_IN_PRICE_CHANGE_OBJ
    RETURN SELF AS RESULT IS
    BEGIN
      RETURN;
    END;
--
CONSTRUCTOR FUNCTION XPTO_IN_PRICE_CHANGE_OBJ(EVENT_TYPE               VARCHAR2,
                                              PRODUCT_ID               VARCHAR2,
                                              LOCATION_ID              NUMBER,
                                              PRICE_ZONE_ID            NUMBER,
                                              COLABORATOR_ID           VARCHAR2,
                                              SALES_PRICE_START_DATE   DATE,
                                              UNIT_SALES_PRICE         NUMBER,
                                              CURRENCY                 VARCHAR2,
                                              REASON_CODE              NUMBER,
                                              STAGE_PRICE_CHANGE_ID    NUMBER,
                                              STATUS_CODE              NUMBER,
                                              ERROR_MESSAGE            VARCHAR2)
  RETURN SELF AS RESULT IS
    BEGIN SELF.EVENT_TYPE             := EVENT_TYPE;
          SELF.PRODUCT_ID             := PRODUCT_ID;
          SELF.LOCATION_ID            := LOCATION_ID;
          SELF.PRICE_ZONE_ID          := PRICE_ZONE_ID;
          SELF.COLABORATOR_ID         := COLABORATOR_ID;
          SELF.SALES_PRICE_START_DATE := SALES_PRICE_START_DATE;
          SELF.UNIT_SALES_PRICE       := UNIT_SALES_PRICE;
          SELF.CURRENCY               := CURRENCY;
          SELF.REASON_CODE            := REASON_CODE;
          SELF.STAGE_PRICE_CHANGE_ID  := STAGE_PRICE_CHANGE_ID;
          SELF.STATUS_CODE            := STATUS_CODE;
          SELF.ERROR_MESSAGE          := ERROR_MESSAGE;
        RETURN;
      END;
--
CONSTRUCTOR FUNCTION XPTO_IN_PRICE_CHANGE_OBJ(EVENT_TYPE               VARCHAR2,
                                              PRODUCT_ID               VARCHAR2,
                                              LOCATION_ID              NUMBER,
                                              COLABORATOR_ID           VARCHAR2,
                                              SALES_PRICE_START_DATE   DATE,
                                              UNIT_SALES_PRICE         NUMBER,
                                              CURRENCY                 VARCHAR2,
                                              REASON_CODE              NUMBER)
  RETURN SELF AS RESULT IS
    BEGIN SELF.EVENT_TYPE             := EVENT_TYPE;
          SELF.PRODUCT_ID             := PRODUCT_ID;
          SELF.LOCATION_ID            := LOCATION_ID;
          SELF.COLABORATOR_ID         := COLABORATOR_ID;
          SELF.SALES_PRICE_START_DATE := SALES_PRICE_START_DATE;
          SELF.UNIT_SALES_PRICE       := UNIT_SALES_PRICE;
          SELF.CURRENCY               := CURRENCY;
          SELF.REASON_CODE            := REASON_CODE;
        RETURN;
      END;
--
CONSTRUCTOR FUNCTION XPTO_IN_PRICE_CHANGE_OBJ(EVENT_TYPE               VARCHAR2,
                                              PRODUCT_ID               VARCHAR2,
                                              PRICE_ZONE_ID            NUMBER,
                                              COLABORATOR_ID           VARCHAR2,
                                              SALES_PRICE_START_DATE   DATE,
                                              UNIT_SALES_PRICE         NUMBER,
                                              CURRENCY                 VARCHAR2,
                                              REASON_CODE              NUMBER)
  RETURN SELF AS RESULT IS
    BEGIN SELF.EVENT_TYPE             := EVENT_TYPE;
          SELF.PRODUCT_ID             := PRODUCT_ID;
          SELF.PRICE_ZONE_ID          := PRICE_ZONE_ID;
          SELF.COLABORATOR_ID         := COLABORATOR_ID;
          SELF.SALES_PRICE_START_DATE := SALES_PRICE_START_DATE;
          SELF.UNIT_SALES_PRICE       := UNIT_SALES_PRICE;
          SELF.CURRENCY               := CURRENCY;
          SELF.REASON_CODE            := REASON_CODE;
        RETURN;
      END;
--
END;
/
```

Figura 26- XPTO_IN_PRICE_CHANGE_OBJ (parte 2)

Package de Base de Dados

XPTO_API_PRICECHANGE.pks

```
CREATE OR REPLACE PACKAGE XPTO_API_PRICECHANGE AS

  ---------------------------------------------------------------------------
  /*************************************************************************/
  /* CREATE DATE - SEPTEMBER 2019                                          */
  /* CREATE USER - JOAO VILAVERDE                                          */
  /* PROJECT     - XPTO                                                    */
  /* DESCRIPTION - PACKAGE TO XXRPM203 - API Wrapper to support upload     */
  /*               operations of Price Changes into the RPM custom price   */
  /*               changes staging area.                                   */
  /*************************************************************************/
  ---------------------------------------------------------------------------
  --Procedure Name : UPLD_PC_TO_STG
  --Purpose       : Support upload operations of Price Changes into the
  --                RPM custom price changes staging area.
  --                xpto_in_price_change - Table type based on xpto_in_price_change_obj
  ---------------------------------------------------------------------------
  PROCEDURE UPLD_PC_TO_STG(IO_xpto_in_price_change IN OUT XPTO_IN_PRICE_CHANGES_TBL,
                           O_status_code             OUT NUMBER,
                           O_error_desc              OUT VARCHAR2);

END XPTO_API_PRICECHANGE;
```

Figura 27- XPTO_API_PRICECHANGE.pks

Ao nível do *spec* apenas se tem a referência ao procedimento que poderá ser invocado por outros sistemas externos. Todas as funções que deste procedimento fazem parte irão constar exclusivamente no *body* deste *package*.

XPTO_API_PRICECHANGE.pkb

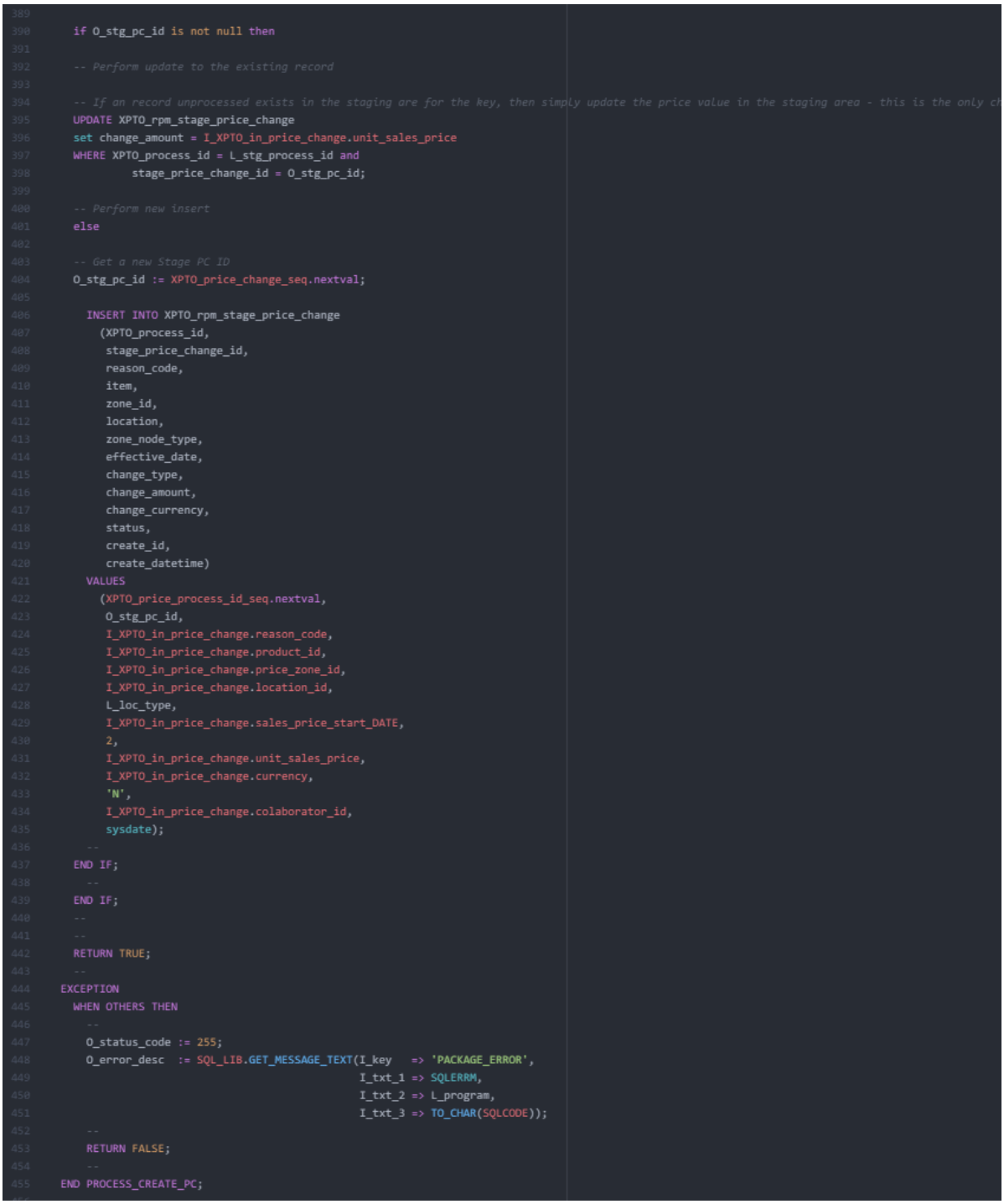
A figura abaixo mostra as funções que fazem parte do *body* do *packge XPTO_API_PRICECHANGE* e, assim sendo, uma vez que não estão presentes no *spec*, estas não podem ser chamadas de forma isolada por outro *package*. Apenas tudo o que está presente no *spec* pode ser chamado por outras entidades externas.

```
CREATE OR REPLACE PACKAGE BODY XPTO_API_PRICECHANGE AS

  ---------------------------------------------------------------------------------
  /*******************************************************************************/
  /* CREATE DATE - SEPTEMBER 2019                                                */
  /* CREATE USER - JOAO VILAVERDE                                                */
  /* PROJECT     - XPTO                                                          */
  /* DESCRIPTION - PACKAGE TO XXRPM203 - API Wrapper to support upload           */
  /*               operations of Price Changes into the RPM custom price         */
  /*               changes staging area.                                         */
  /*******************************************************************************/
  ---------------------------------------------------------------------------------

  /*******************************************************************************/
  /* CHANGES                                                                     */
  /* 2019-10-25 - JOAO VILAVERDE - MARS-1570,  MARS-1571 and MARS-2048           */
  /*******************************************************************************/

  ---------------------------------------------------------------------------------
  --   GLOBAL VARIABLES
  ---------------------------------------------------------------------------------
  LP_STORE     VARCHAR2(1) := 'S';
  LP_WAREHOUSE VARCHAR2(1) := 'W';
  LP_CREATE    VARCHAR2(6) := 'CREATE';
  LP_UPDATE    VARCHAR2(6) := 'UPDATE';
  LP_UPSERT    VARCHAR2(6) := 'UPSERT';
  LP_DELETE    VARCHAR2(6) := 'DELETE';
  LP_API_AUDIT VARCHAR2(1);
  --
  ---------------------------------------------------------------------------------
  --Function Name : PROCESS_CREATE_PC
  --Purpose       : Create price change, with status N, in the RPM staging area.
  --              I_XPTO_in_price_change -> TYpe to hold the records process
  ---------------------------------------------------------------------------------
  FUNCTION PROCESS_CREATE_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGE_OBJ,
                             O_stg_pc_id                OUT NUMBER,
                             O_status_code              OUT NUMBER,
                             O_error_desc               OUT VARCHAR2)
    RETURN BOOLEAN;

  ---------------------------------------------------------------------------------
  --Function Name : PROCESS_UPDATE_PC
  --Purpose       : Update price change, with status U, in the RPM staging area.
  --              I_XPTO_in_price_change -> TYpe to hold the records process
  ---------------------------------------------------------------------------------
  FUNCTION PROCESS_UPDATE_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGE_OBJ,
              I_price_change_id IN NUMBER,
              I_status IN VARCHAR2,
                             O_stg_pc_id                OUT NUMBER,
                             O_status_code              OUT NUMBER,
                             O_ERROR_DESC               OUT VARCHAR2)
    RETURN BOOLEAN;

  ---------------------------------------------------------------------------------
  --Function Name : PROCESS_UPSERT_PC
  --Purpose       : Create/Update price change, with status N or U, in the RPM staging area.
  --              I_XPTO_in_price_change -> TYpe to hold the records process
  ---------------------------------------------------------------------------------
  FUNCTION PROCESS_UPSERT_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGE_OBJ,
                             O_stg_pc_id                OUT NUMBER,
                             O_status_code              OUT NUMBER,
                             O_error_desc               OUT VARCHAR2)
    RETURN BOOLEAN;
  ---------------------------------------------------------------------------------
  --Function Name : PROCESS_DELETE_PC
  --Purpose       : Update price change, with status D, in the RPM staging area.
  ---              I_XPTO_in_price_change -> TYpe to hold the records process
  ---------------------------------------------------------------------------------
  FUNCTION PROCESS_DELETE_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGE_OBJ,
                             O_stg_pc_id                OUT NUMBER,
                             O_status_code              OUT NUMBER,
                             O_error_desc               OUT VARCHAR2)
    RETURN BOOLEAN;
  ---------------------------------------------------------------------------------
  --Function Name : PROCESS_VALIDATION_PC
  --Purpose       : Validations for RPM upload price change to stanging table.
  ---              I_XPTO_in_price_change -> TYpe to hold the records process
  ---------------------------------------------------------------------------------
  FUNCTION PROCESS_VALIDATION_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGES_TBL,
                                 O_status_code              OUT NUMBER,
                                 O_error_desc               OUT VARCHAR2)
    RETURN BOOLEAN;
  ---------------------------------------------------------------------------------
```

Figura 28- XPTO_API_PRICECHANGE.pkb

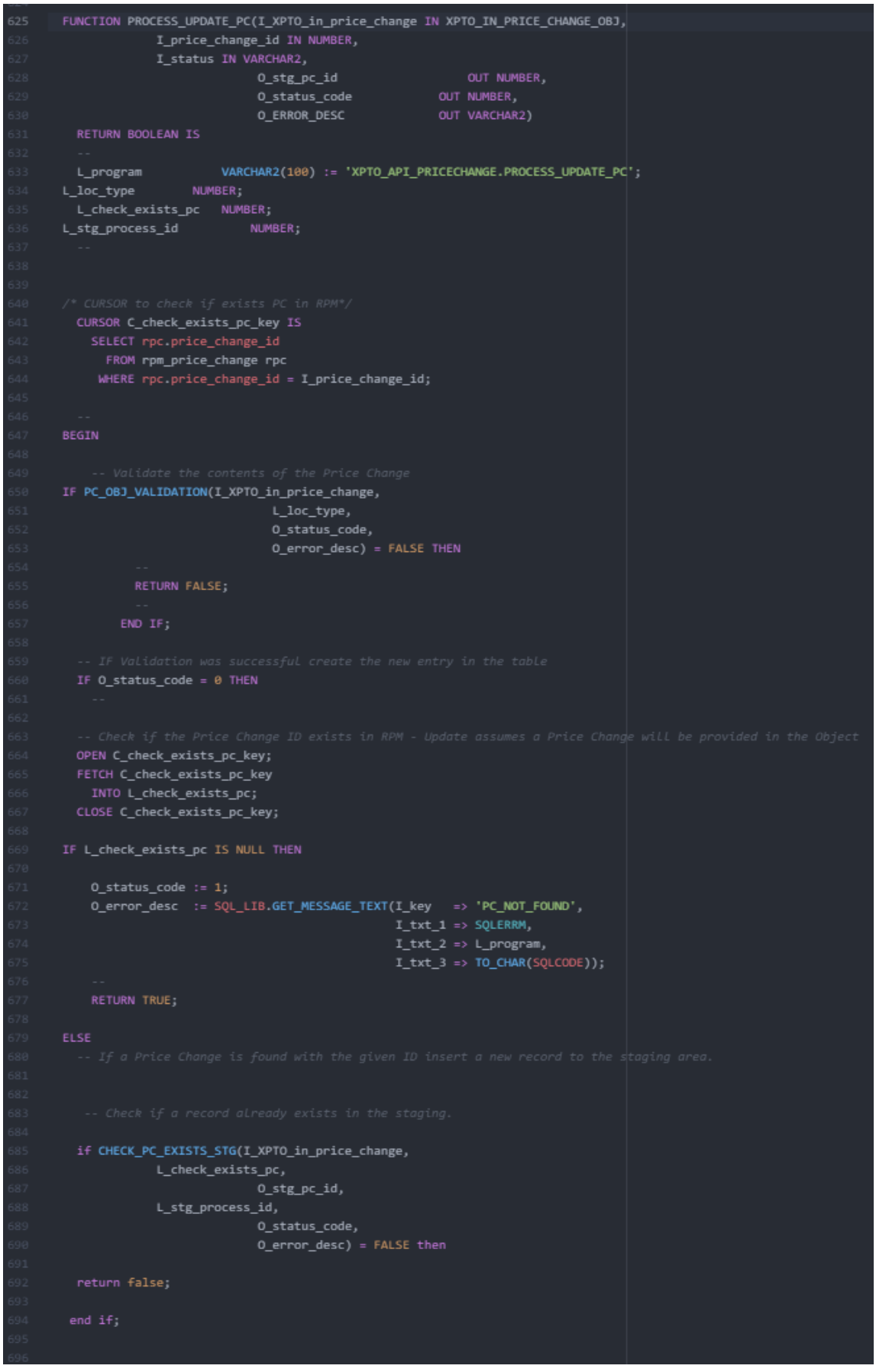
```
FUNCTION PROCESS_CREATE_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGE_OBJ,
                           O_stg_pc_id                 OUT NUMBER,
                           O_status_code           OUT NUMBER,
                           O_error_desc            OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program           VARCHAR2(100) := 'XPTO_API_PRICECHANGE.PROCESS_CREATE_PC';
L_loc_type        NUMBER;
L_stg_process_id          NUMBER;
  --

  --
BEGIN
  --
  -- Validate the contents of the Price Change
IF PC_OBJ_VALIDATION(I_XPTO_in_price_change,
                             L_loc_type,
                             O_status_code,
                             O_error_desc) = FALSE THEN
          --
          RETURN FALSE;
          --
        END IF;

  -- If Validation was successful create the new entry in the table
  IF O_status_code = 0 THEN
    --

  -- Check if a record already exists in the staging.

  if CHECK_PC_EXISTS_STG(I_XPTO_in_price_change,
            NULL,
                           O_stg_pc_id,
            L_stg_process_id,
                           O_status_code,
                           O_error_desc) = FALSE then

  return false;

 end if;
```

Figura 29- Função *Create API Price Changes* (parte 1)



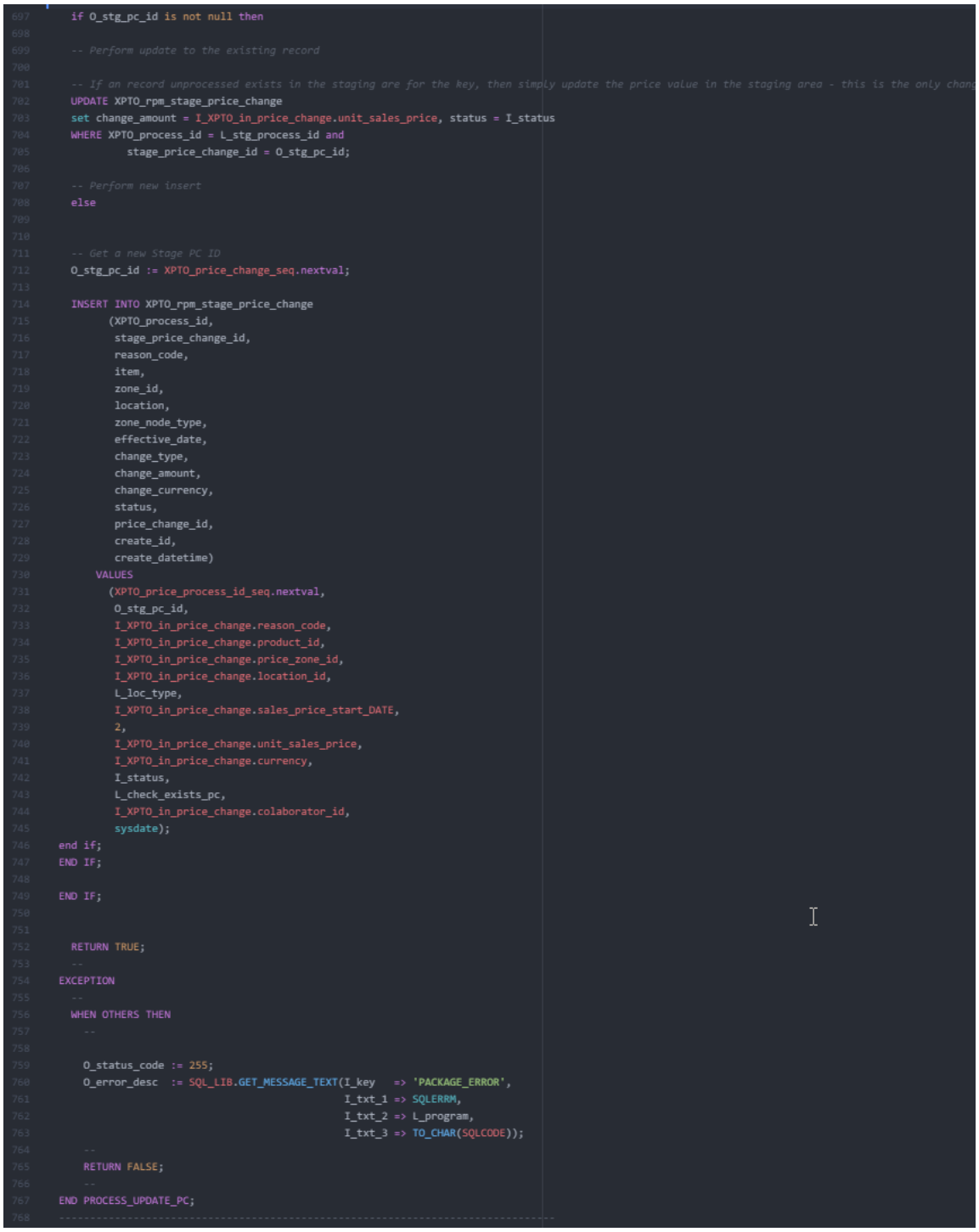
```
    if O_stg_pc_id is not null then

    -- Perform update to the existing record

    -- If an record unprocessed exists in the staging are for the key, then simply update the price value in the staging area - this is the only ch
    UPDATE XPTO_rpm_stage_price_change
    set change_amount = I_XPTO_in_price_change.unit_sales_price
    WHERE XPTO_process_id = L_stg_process_id and
             stage_price_change_id = O_stg_pc_id;

    -- Perform new insert
    else

    -- Get a new Stage PC ID
    O_stg_pc_id := XPTO_price_change_seq.nextval;

      INSERT INTO XPTO_rpm_stage_price_change
        (XPTO_process_id,
         stage_price_change_id,
         reason_code,
         item,
         zone_id,
         location,
         zone_node_type,
         effective_date,
         change_type,
         change_amount,
         change_currency,
         status,
         create_id,
         create_datetime)
      VALUES
        (XPTO_price_process_id_seq.nextval,
         O_stg_pc_id,
         I_XPTO_in_price_change.reason_code,
         I_XPTO_in_price_change.product_id,
         I_XPTO_in_price_change.price_zone_id,
         I_XPTO_in_price_change.location_id,
         L_loc_type,
         I_XPTO_in_price_change.sales_price_start_DATE,
         2,
         I_XPTO_in_price_change.unit_sales_price,
         I_XPTO_in_price_change.currency,
         'N',
         I_XPTO_in_price_change.colaborator_id,
         sysdate);
      --
    END IF;
      --
    END IF;
    --
    --
    RETURN TRUE;
    --
  EXCEPTION
    WHEN OTHERS THEN
      --
      O_status_code := 255;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                                I_txt_1 => SQLERRM,
                                                I_txt_2 => L_program,
                                                I_txt_3 => TO_CHAR(SQLCODE));
      --
      RETURN FALSE;
      --
  END PROCESS_CREATE_PC;
```

Figura 30- Função *Create API Price Changes* (parte 2)



```
FUNCTION PROCESS_UPDATE_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGE_OBJ,
             I_price_change_id IN NUMBER,
             I_status IN VARCHAR2,
                          O_stg_pc_id                 OUT NUMBER,
                          O_status_code            OUT NUMBER,
                          O_ERROR_DESC             OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program           VARCHAR2(100) := 'XPTO_API_PRICECHANGE.PROCESS_UPDATE_PC';
L_loc_type        NUMBER;
  L_check_exists_pc   NUMBER;
L_stg_process_id          NUMBER;
  --


/* CURSOR to check if exists PC in RPM*/
  CURSOR C_check_exists_pc_key IS
    SELECT rpc.price_change_id
      FROM rpm_price_change rpc
     WHERE rpc.price_change_id = I_price_change_id;

  --
BEGIN

    -- Validate the contents of the Price Change
IF PC_OBJ_VALIDATION(I_XPTO_in_price_change,
                             L_loc_type,
                             O_status_code,
                             O_error_desc) = FALSE THEN
          --
          RETURN FALSE;
          --
        END IF;

  -- IF Validation was successful create the new entry in the table
  IF O_status_code = 0 THEN
    --

  -- Check if the Price Change ID exists in RPM - Update assumes a Price Change will be provided in the Object
  OPEN C_check_exists_pc_key;
  FETCH C_check_exists_pc_key
    INTO L_check_exists_pc;
  CLOSE C_check_exists_pc_key;

IF L_check_exists_pc IS NULL THEN

    O_status_code := 1;
    O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PC_NOT_FOUND',
                                              I_txt_1 => SQLERRM,
                                              I_txt_2 => L_program,
                                              I_txt_3 => TO_CHAR(SQLCODE));
    --
    RETURN TRUE;

ELSE
  -- If a Price Change is found with the given ID insert a new record to the staging area.


   -- Check if a record already exists in the staging.

  if CHECK_PC_EXISTS_STG(I_XPTO_in_price_change,
             L_check_exists_pc,
                          O_stg_pc_id,
             L_stg_process_id,
                          O_status_code,
                          O_error_desc) = FALSE then

  return false;

 end if;
```

Figura 31- Função *Update API Price Changes* (parte 1)

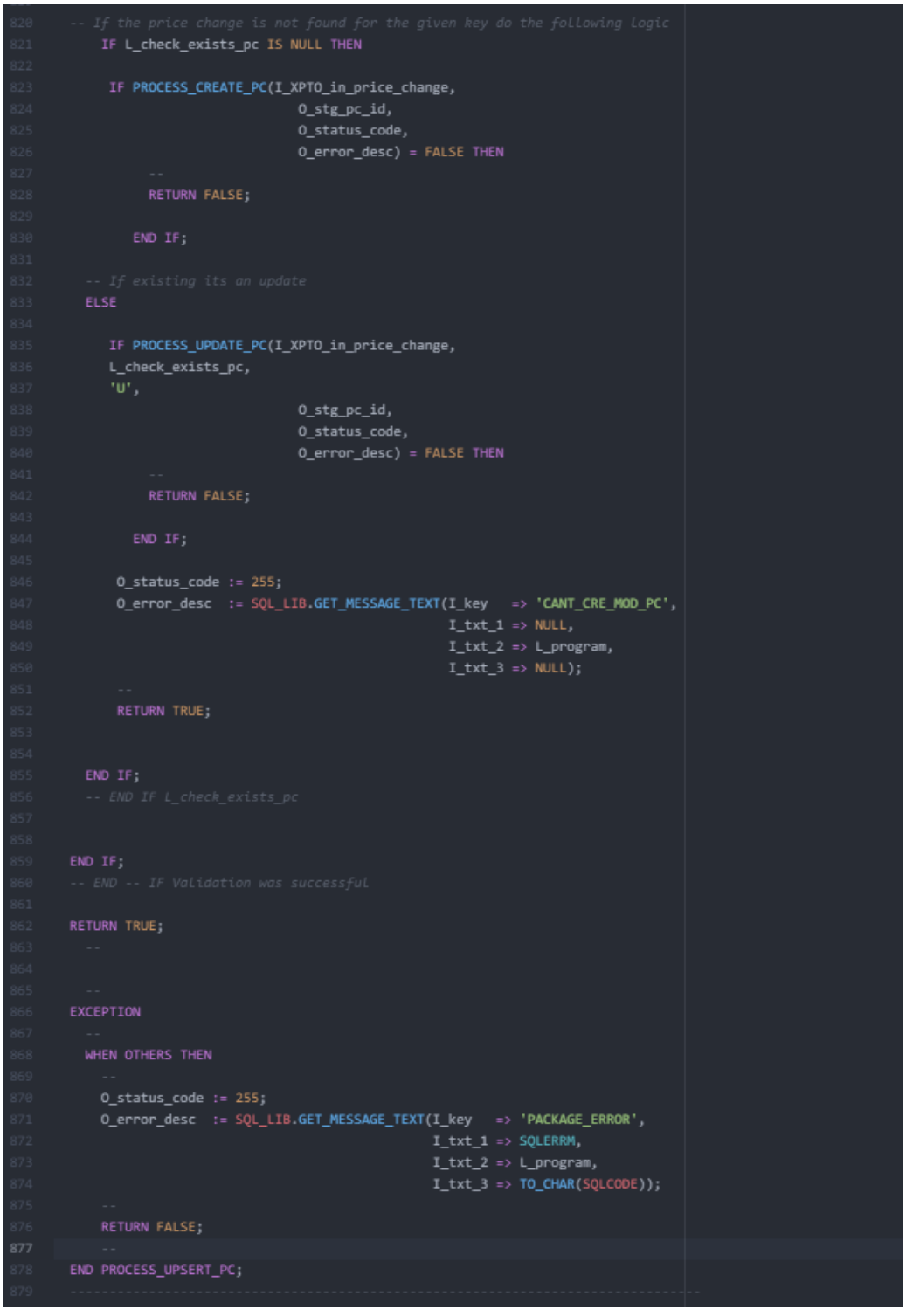


```
    if O_stg_pc_id is not null then

    -- Perform update to the existing record

    -- If an record unprocessed exists in the staging are for the key, then simply update the price value in the staging area - this is the only chang
    UPDATE XPTO_rpm_stage_price_change
    set change_amount = I_XPTO_in_price_change.unit_sales_price, status = I_status
    WHERE XPTO_process_id = L_stg_process_id and
             stage_price_change_id = O_stg_pc_id;

    -- Perform new insert
    else


    -- Get a new Stage PC ID
    O_stg_pc_id := XPTO_price_change_seq.nextval;

    INSERT INTO XPTO_rpm_stage_price_change
          (XPTO_process_id,
           stage_price_change_id,
           reason_code,
           item,
           zone_id,
           location,
           zone_node_type,
           effective_date,
           change_type,
           change_amount,
           change_currency,
           status,
           price_change_id,
           create_id,
           create_datetime)
        VALUES
          (XPTO_price_process_id_seq.nextval,
           O_stg_pc_id,
           I_XPTO_in_price_change.reason_code,
           I_XPTO_in_price_change.product_id,
           I_XPTO_in_price_change.price_zone_id,
           I_XPTO_in_price_change.location_id,
           L_loc_type,
           I_XPTO_in_price_change.sales_price_start_DATE,
           2,
           I_XPTO_in_price_change.unit_sales_price,
           I_XPTO_in_price_change.currency,
           I_status,
           L_check_exists_pc,
           I_XPTO_in_price_change.colaborator_id,
           sysdate);
  end if;
  END IF;

  END IF;


    RETURN TRUE;
    --
  EXCEPTION
    --
    WHEN OTHERS THEN
      --

      O_status_code := 255;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                                I_txt_1 => SQLERRM,
                                                I_txt_2 => L_program,
                                                I_txt_3 => TO_CHAR(SQLCODE));
      --
      RETURN FALSE;
      --
  END PROCESS_UPDATE_PC;
  ----------------------------------------------------------------------------------
```

Figura 32- Função *Update API Price Changes* (parte 2)

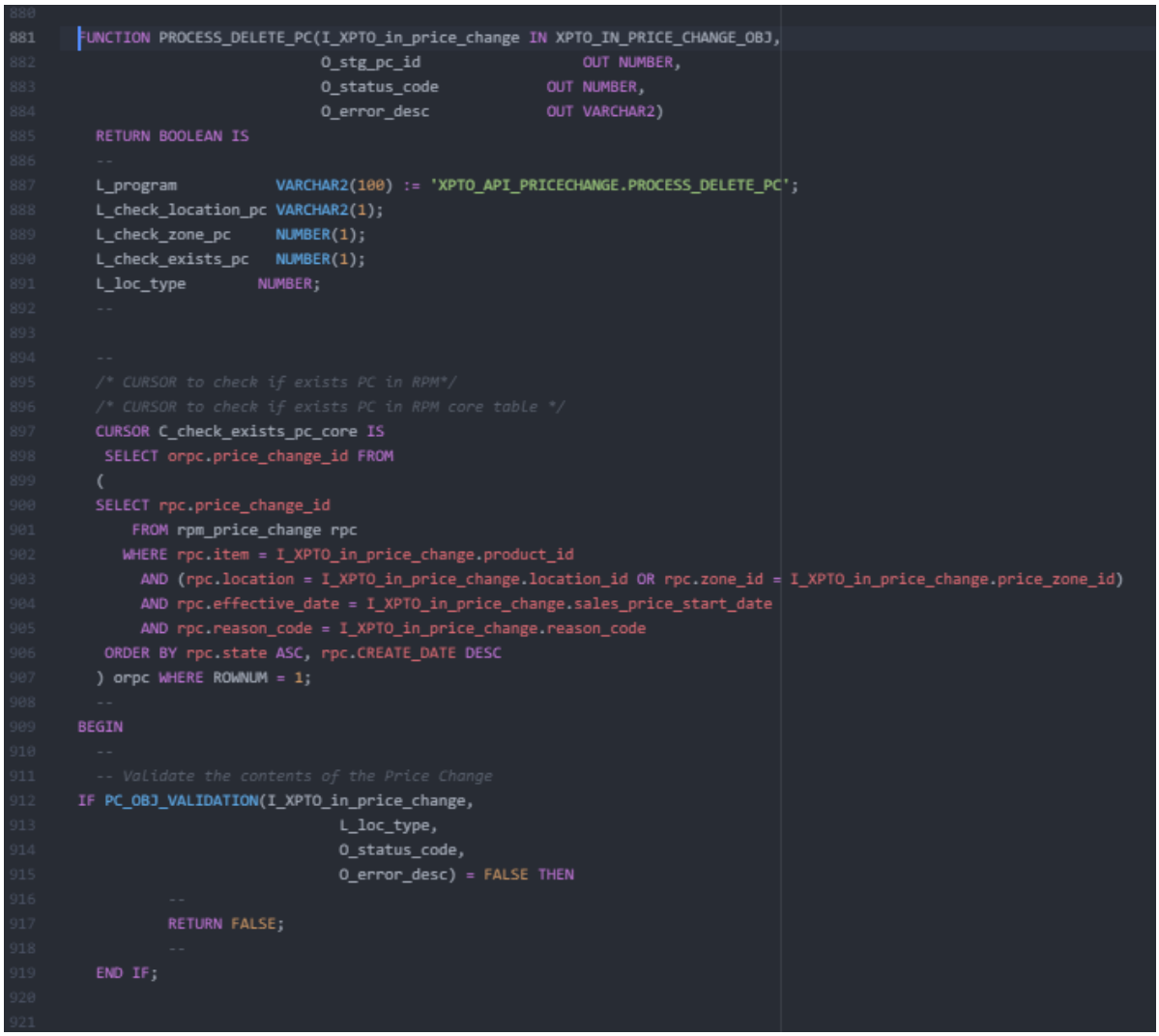
```
FUNCTION PROCESS_UPSERT_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGE_OBJ,
                           O_stg_pc_id                OUT NUMBER,
                           O_status_code              OUT NUMBER,
                           O_error_desc               OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program           VARCHAR2(100) := 'XPTO_API_PRICECHANGE.PROCESS_UPSERT_PC';
  L_check_exists_pc   NUMBER;
L_loc_type        NUMBER;
  --


  --
  /* CURSOR to check if exists PC in RPM core table */
  CURSOR C_check_exists_pc_core IS
   SELECT orpc.price_change_id FROM
  (
  SELECT rpc.price_change_id
      FROM rpm_price_change rpc
     WHERE rpc.item = I_XPTO_in_price_change.product_id
       AND (rpc.location = I_XPTO_in_price_change.location_id OR rpc.zone_id = I_XPTO_in_price_change.price_zone_id)
       AND rpc.effective_date = I_XPTO_in_price_change.sales_price_start_date
       AND rpc.reason_code = I_XPTO_in_price_change.reason_code
   ORDER BY rpc.state ASC, rpc.CREATE_DATE DESC
  ) orpc WHERE ROWNUM = 1;

  --
BEGIN
  --
  -- Validate the contents of the Price Change
IF PC_OBJ_VALIDATION(I_XPTO_in_price_change,
                           L_loc_type,
                           O_status_code,
                           O_error_desc) = FALSE THEN
          --
          RETURN FALSE;
          --
  END IF;


  -- If Validation was successful create the new entry in the table
  IF O_status_code = 0 THEN
    --

    -- Check if the Price Change ID exists in RPM - Update assumes a Price Change will be provided in the Object
    OPEN C_check_exists_pc_core;
  FETCH C_check_exists_pc_core
   INTO L_check_exists_pc;
  CLOSE C_check_exists_pc_core;

```

Figura 33- Função *Upsert API Price Changes* (parte 1)



```
-- If the price change is not found for the given key do the following logic
    IF L_check_exists_pc IS NULL THEN

     IF PROCESS_CREATE_PC(I_XPTO_in_price_change,
                          O_stg_pc_id,
                          O_status_code,
                          O_error_desc) = FALSE THEN
          --
          RETURN FALSE;

        END IF;

  -- If existing its an update
  ELSE

     IF PROCESS_UPDATE_PC(I_XPTO_in_price_change,
     L_check_exists_pc,
     'U',
                          O_stg_pc_id,
                          O_status_code,
                          O_error_desc) = FALSE THEN
          --
          RETURN FALSE;

        END IF;

      O_status_code := 255;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'CANT_CRE_MOD_PC',
                                                I_txt_1 => NULL,
                                                I_txt_2 => L_program,
                                                I_txt_3 => NULL);
      --
      RETURN TRUE;


  END IF;
  -- END IF L_check_exists_pc


END IF;
-- END -- IF Validation was successful

RETURN TRUE;
  --


  --
EXCEPTION
  --
  WHEN OTHERS THEN
    --
    O_status_code := 255;
    O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                              I_txt_1 => SQLERRM,
                                              I_txt_2 => L_program,
                                              I_txt_3 => TO_CHAR(SQLCODE));
    --
    RETURN FALSE;
    --
END PROCESS_UPSERT_PC;
--------------------------------------------------------------------------------
```

Figura 34- Função *Upsert API Price Changes* (parte 2)



```
FUNCTION PROCESS_DELETE_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGE_OBJ,
                           O_stg_pc_id                OUT NUMBER,
                           O_status_code           OUT NUMBER,
                           O_error_desc            OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program           VARCHAR2(100) := 'XPTO_API_PRICECHANGE.PROCESS_DELETE_PC';
  L_check_location_pc VARCHAR2(1);
  L_check_zone_pc     NUMBER(1);
  L_check_exists_pc   NUMBER(1);
  L_loc_type        NUMBER;
  --


  --
  /* CURSOR to check if exists PC in RPM*/
  /* CURSOR to check if exists PC in RPM core table */
  CURSOR C_check_exists_pc_core IS
   SELECT orpc.price_change_id FROM
  (
  SELECT rpc.price_change_id
      FROM rpm_price_change rpc
     WHERE rpc.item = I_XPTO_in_price_change.product_id
       AND (rpc.location = I_XPTO_in_price_change.location_id OR rpc.zone_id = I_XPTO_in_price_change.price_zone_id)
       AND rpc.effective_date = I_XPTO_in_price_change.sales_price_start_date
       AND rpc.reason_code = I_XPTO_in_price_change.reason_code
   ORDER BY rpc.state ASC, rpc.CREATE_DATE DESC
  ) orpc WHERE ROWNUM = 1;
  --
BEGIN
  --
  -- Validate the contents of the Price Change
IF PC_OBJ_VALIDATION(I_XPTO_in_price_change,
                           L_loc_type,
                           O_status_code,
                           O_error_desc) = FALSE THEN
          --
          RETURN FALSE;
          --
  END IF;
```

Figura 35- Função *Delete API Price Changes* (parte 1)

```
920
921
922       -- IF Validation was successful create the new entry in the table
923       IF O_status_code = 0 THEN
924
925
926       --
927       OPEN C_check_exists_pc_core;
928       FETCH C_check_exists_pc_core
929         INTO L_check_exists_pc;
930       CLOSE C_check_exists_pc_core;
931       --
932       IF L_check_exists_pc IS NOT NULL THEN
933         --
934            IF PROCESS_UPDATE_PC(I_XPTO_in_price_change,
935           L_check_exists_pc,
936           'U',
937                                O_stg_pc_id,
938                                O_status_code,
939                                O_error_desc) = FALSE THEN
940               --
941              RETURN FALSE;
942
943             END IF;
944
945
946
947         --
948       ELSE
949         --
950         O_status_code := 1;
951         O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PC_NOT_FOUND',
952                                                   I_txt_1 => SQLERRM,
953                                                   I_txt_2 => L_program,
954                                                   I_txt_3 => TO_CHAR(SQLCODE));
955         --
956         RETURN TRUE;
957         --
958       END IF;
959
960     END IF;
961
962       --
963       RETURN TRUE;
964       --
965     EXCEPTION
966       --
967       WHEN OTHERS THEN
968         --
969         O_status_code := 255;
970         O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
971                                                   I_txt_1 => SQLERRM,
972                                                   I_txt_2 => L_program,
973                                                   I_txt_3 => TO_CHAR(SQLCODE));
974         --
975         RETURN FALSE;
976         --
977     END PROCESS_DELETE_PC;
978     ----------------------------------------------------------------------------
```

Figura 36- Função *Delete API Price Changes* (parte 2)

```
FUNCTION PROCESS_VALIDATION_PC(I_XPTO_in_price_change IN XPTO_IN_PRICE_CHANGES_TBL,
                               O_status_code           OUT NUMBER,
                               O_error_desc            OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program             VARCHAR2(100) := 'XPTO_API_PRICECHANGE.PROCESS_VALIDATION_PC';
  L_max_no_rec_upld_rec XPTO_API_SETTINGS.MAX_SYNC_UPLD_REC%TYPE;
  L_api_audit_rec       VARCHAR2(1);
  --
  /* CURSOR to get the settings of the API */
  CURSOR C_api_settings IS
    SELECT max_sync_upld_rec
      FROM XPTO_api_settings
     WHERE api_name = 'XPTO_API_PRICECHANGE';
  --
BEGIN
  --
  OPEN C_api_settings;
  FETCH C_api_settings
    INTO L_max_no_rec_upld_rec;
  CLOSE C_api_settings;
  --


  --
  IF L_max_no_rec_upld_rec IS NULL THEN
    --
    O_status_code := 255;
    --
    O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'MAX_NO_REC_NULL',
                                             I_txt_1 => NULL,
                                             I_txt_2 => L_program,
                                             I_txt_3 => NULL);
    --
    --
    RETURN TRUE;
    --
  END IF;
  --
  IF L_max_no_rec_upld_rec < I_XPTO_in_price_change.count THEN
    --
    O_status_code := 1;
    --
    O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'MAX_NO_REC_LESS',
                                             I_txt_1 => NULL,
                                             I_txt_2 => L_program,
                                             I_txt_3 => NULL);
    --
    RETURN TRUE;
    --
  END IF;
  --
  RETURN TRUE;
  --
EXCEPTION
  WHEN OTHERS THEN
    --
    O_status_code := 255;
    --
    O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                             I_txt_1 => SQLERRM,
                                             I_txt_2 => L_program,
                                             I_txt_3 => TO_CHAR(SQLCODE));
    --
    RETURN FALSE;
    --
END PROCESS_VALIDATION_PC;
--------------------------------------------------------------------------------
```

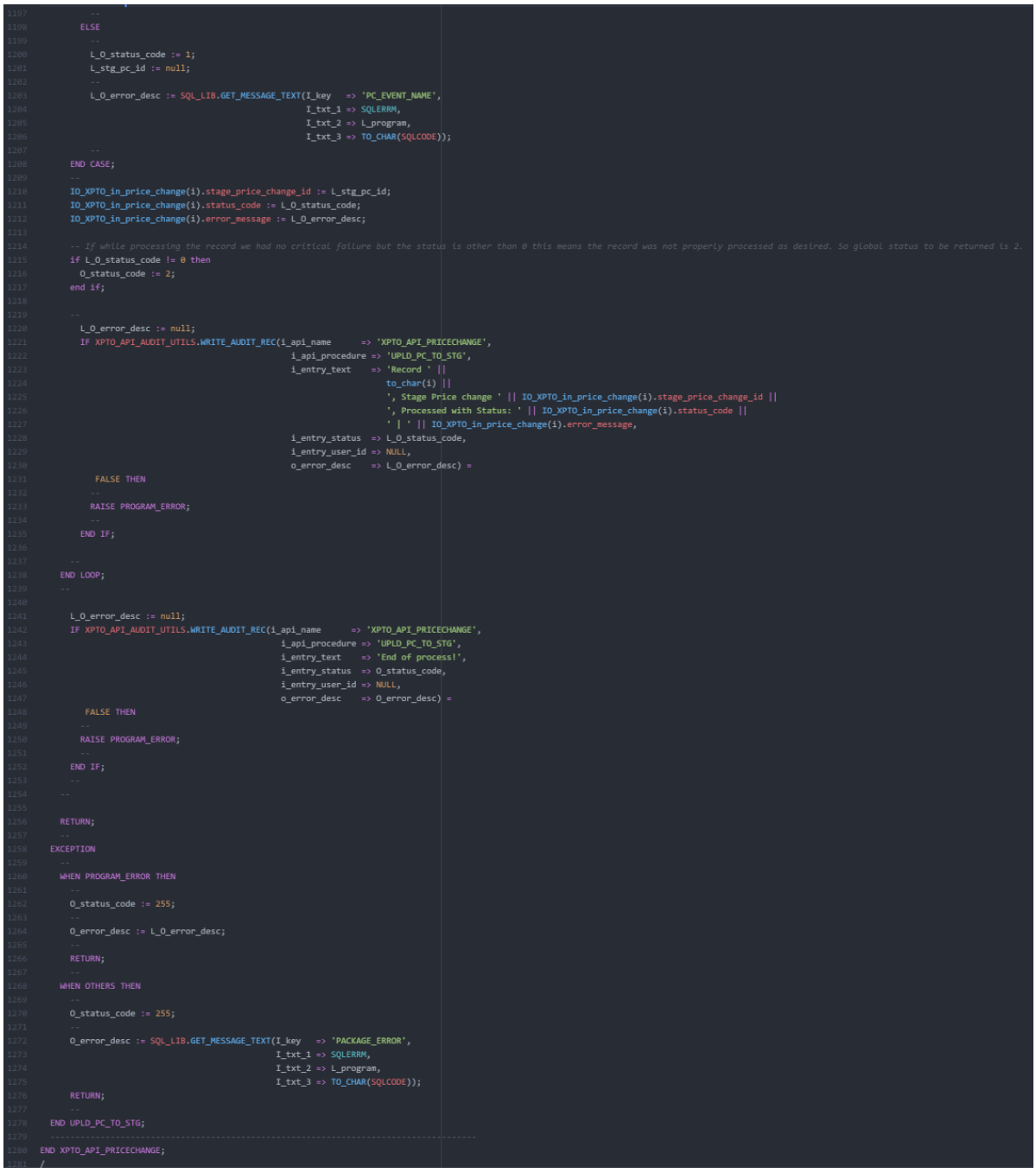
Figura 37- Função das Validações Iniciais *API Price Changes*

```
--Procedure Name : UPLD_PC_TO_STG
--Purpose        : Support upload operations of Price Changes into the
--                  RPM custom price changes staging area.
--                  XPTO_in_price_change - Table type based on XPTO_in_price_change_obj
--------------------------------------------------------------------------------
PROCEDURE UPLD_PC_TO_STG(IO_XPTO_in_price_change IN OUT XPTO_IN_PRICE_CHANGES_TBL,
                         O_status_code             OUT NUMBER,
                         O_error_desc              OUT VARCHAR2) IS
  --
  L_program       VARCHAR2(100) := 'XPTO_API_PRICECHANGE.UPLD_PC_TO_STG';
  L_O_status_code NUMBER;
  L_O_error_desc  VARCHAR2(1000);
  L_stg_pc_id       NUMBER;
  --
  -- EXCEPTION
  --
  PROGRAM_ERROR EXCEPTION;
  --
BEGIN
  --
  O_status_code  := 0; -- Initial Status to be returned is Success
  L_O_error_desc := NULL;
  --


  --
  IF PROCESS_VALIDATION_PC(IO_XPTO_in_price_change,
                           L_O_status_code,
                           L_O_error_desc) = FALSE THEN
    --

      IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => 'XPTO_API_PRICECHANGE',
                                              i_api_procedure => 'UPLD_PC_TO_STG',
                                              i_entry_text    => 'An error occured while executing the global validations: ' ||
                                                                 L_O_error_desc,
                                              i_entry_status  => O_status_code,
                                              i_entry_user_id => NULL,
                                              o_error_desc    => L_O_error_desc) =
         FALSE THEN
        --
        RAISE PROGRAM_ERROR;
        --
      END IF;
      --
    --
  END IF;
  --

  -- Exit in case the Validation function returns a status other than 0
  IF L_O_status_code != 0 then

    -- START AUDIT
      --
      IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => 'XPTO_API_PRICECHANGE',
                                              i_api_procedure => 'UPLD_PC_TO_STG',
                                              i_entry_text    => 'Initial process validations failure: ' ||
                                                                 L_O_error_desc,
                                              i_entry_status  => NULL,
                                              i_entry_user_id => NULL,
                                              o_error_desc    => L_O_error_desc) =
         FALSE THEN
        --
        RAISE PROGRAM_ERROR;
        --
      END IF;
      --
    -- END AUDIT

    O_status_code := L_O_status_code;
    O_error_desc  := L_O_error_desc;
    return;

  END IF;
```

Figura 38- Procedimento de UPLD_PC_TO_STG *API Price Changes* (parte 1)

```
1118    -- In case validations succeeded, process each Price Change record
1119      --
1120      L_O_error_desc := null;
1121      IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => 'XPTO_API_PRICECHANGE',
1122                                              i_api_procedure => 'UPLD_PC_TO_STG',
1123                                              i_entry_text    => 'Process Started!',
1124                                              i_entry_status  => L_O_status_code,
1125                                              i_entry_user_id => NULL,
1126                                              o_error_desc    => L_O_error_desc) =
1127         FALSE THEN
1128        --
1129        RAISE PROGRAM_ERROR;
1130        --
1131      END IF;
1132      --
1133    --
1134    FOR i in 1 .. IO_XPTO_in_price_change.count LOOP
1135      --
1136      CASE IO_XPTO_in_price_change(i).event_type
1137      --
1138      /*CREATE*/
1139      --
1140        WHEN LP_CREATE THEN
1141          --
1142          IF PROCESS_CREATE_PC(IO_XPTO_in_price_change(i),
1143                               L_stg_pc_id,
1144                               L_O_status_code,
1145                               L_O_error_desc) = FALSE THEN
1146            --
1147            RAISE PROGRAM_ERROR;
1148            --CONTINUE;
1149            --
1150          END IF;
1151          --
1152      /*UPDATE*/
1153      --
1154        WHEN LP_UPDATE THEN
1155          --
1156          IF PROCESS_UPDATE_PC_NO_ID(IO_XPTO_in_price_change(i),
1157                'U',
1158                               L_stg_pc_id,
1159                               L_O_status_code,
1160                               L_O_error_desc) = FALSE THEN
1161            --
1162            RAISE PROGRAM_ERROR;
1163            --CONTINUE;
1164
1165
1166
1167            --
1168          END IF;
1169          --
1170      /*UPSERT*/
1171      --
1172        WHEN LP_UPSERT THEN
1173          --
1174          IF PROCESS_UPSERT_PC(IO_XPTO_in_price_change(i),
1175                               L_stg_pc_id,
1176                               L_O_status_code,
1177                               L_O_error_desc) = FALSE THEN
1178            --
1179            RAISE PROGRAM_ERROR;
1180            --CONTINUE;
1181            --
1182          END IF;
1183          --
1184      /*DELETE*/
1185      --
1186        WHEN LP_DELETE THEN
1187          --
1188          IF PROCESS_DELETE_PC(IO_XPTO_in_price_change(i),
1189                               L_stg_pc_id,
1190                               L_O_status_code,
1191                               L_O_error_desc) = FALSE THEN
1192            --
1193            RAISE PROGRAM_ERROR;
1194            --CONTINUE;
1195            --
1196          END IF;
```

Figura 39- Procedimento de UPLD_PC_TO_STG *API Price Changes* (parte 2)



```
        --
      ELSE
        --
        L_O_status_code := 1;
        L_stg_pc_id := null;
        --
        L_O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PC_EVENT_NAME',
                                                   I_txt_1 => SQLERRM,
                                                   I_txt_2 => L_program,
                                                   I_txt_3 => TO_CHAR(SQLCODE));
        --
    END CASE;
    --
    IO_XPTO_in_price_change(i).stage_price_change_id := L_stg_pc_id;
    IO_XPTO_in_price_change(i).status_code := L_O_status_code;
    IO_XPTO_in_price_change(i).error_message := L_O_error_desc;

    -- If while processing the record we had no critical failure but the status is other than 0 this means the record was not properly processed as desired. So global status to be returned is 2.
    if L_O_status_code != 0 then
      O_status_code := 2;
    end if;

    --
      L_O_error_desc := null;
      IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => 'XPTO_API_PRICECHANGE',
                                              i_api_procedure => 'UPLD_PC_TO_STG',
                                              i_entry_text    => 'Record ' ||
                                                 to_char(i) ||
                                                 ', Stage Price change ' || IO_XPTO_in_price_change(i).stage_price_change_id ||
                                                 ', Processed with Status: ' || IO_XPTO_in_price_change(i).status_code ||
                                                 ' | ' || IO_XPTO_in_price_change(i).error_message,
                                              i_entry_status  => L_O_status_code,
                                              i_entry_user_id => NULL,
                                              o_error_desc    => L_O_error_desc) =
         FALSE THEN
        --
        RAISE PROGRAM_ERROR;
        --
      END IF;

    --
  END LOOP;
  --

    L_O_error_desc := null;
    IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => 'XPTO_API_PRICECHANGE',
                                            i_api_procedure => 'UPLD_PC_TO_STG',
                                            i_entry_text    => 'End of process!',
                                            i_entry_status  => O_status_code,
                                            i_entry_user_id => NULL,
                                            o_error_desc    => O_error_desc) =
       FALSE THEN
      --
      RAISE PROGRAM_ERROR;
      --
    END IF;
    --
  --

  RETURN;
  --
EXCEPTION
  --
  WHEN PROGRAM_ERROR THEN
    --
    O_status_code := 255;
    --
    O_error_desc := L_O_error_desc;
    --
    RETURN;
    --
  WHEN OTHERS THEN
    --
    O_status_code := 255;
    --
    O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                             I_txt_1 => SQLERRM,
                                             I_txt_2 => L_program,
                                             I_txt_3 => TO_CHAR(SQLCODE));
    RETURN;
    --
END UPLD_PC_TO_STG;
--------------------------------------------------------------------------------
END XPTO_API_PRICECHANGE;
/
```

Figura 40- Procedimento de UPLD_PC_TO_STG *API Price Changes* (parte 3)

Teste Unitário

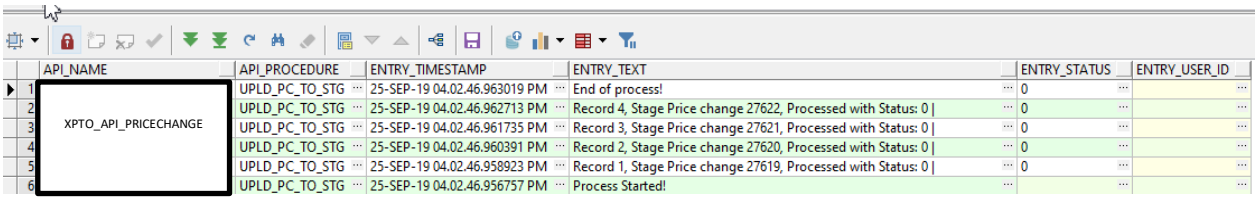
Cenário Positivo – Invocar o procedimento passando como input: Uma price change, ao nível da *location*, com evento *Create*; uma price change com evento *Update* sendo que a price change tem de existir no RMS; uma price change com evento *Upsert* sendo que a *price change* não existe no RMS; uma price change com evento *Delete* sendo que a p*rice change* tem de existir no RMS.

Para executar este teste foi necessário criar uma *script* de teste para poder simular a entrada de dados vinda dos sistemas externos. A *script* usada foi a seguinte:

```
declare

  lc_status_code number;
  lc_error_msg   varchar2(1000);

  lc_price_tbl "XPTO_IN_PRICE_CHANGES_TBL";
  lc_price_obj "XPTO_IN_PRICE_CHANGE_OBJ";

begin

  lc_price_tbl := "XPTO_IN_PRICE_CHANGES_TBL"();

  -- 1st Price change for testing:
  lc_price_obj                       := "XPTO_IN_PRICE_CHANGE_OBJ"();
  lc_price_obj.EVENT_TYPE            := 'CREATE';
  lc_price_obj.PRODUCT_ID            := '45449831';
  lc_price_obj.LOCATION_ID           := 400610;
  lc_price_obj.PRICE_ZONE_ID         := NULL;
  lc_price_obj.COLABORATOR_ID        := 'DOG';
  lc_price_obj.SALES_PRICE_START_DATE := to_date('01012020', 'DDMMYYYY');
  lc_price_obj.UNIT_SALES_PRICE      := 99;
  lc_price_obj.CURRENCY              := 'PLN';
  lc_price_obj.REASON_CODE           := '1';

  lc_price_tbl.extend();
  lc_price_tbl(lc_price_tbl.last) := lc_price_obj;

  -- 1st Price change for testing:
  lc_price_obj                       := "XPTO_IN_PRICE_CHANGE_OBJ"();
  lc_price_obj.EVENT_TYPE            := 'UPDATE';
  lc_price_obj.PRODUCT_ID            := '777011115';
  lc_price_obj.LOCATION_ID           := 700610;
  lc_price_obj.PRICE_ZONE_ID         := NULL;
  lc_price_obj.COLABORATOR_ID        := 'DOG';
  lc_price_obj.SALES_PRICE_START_DATE := to_date('22052019', 'DDMMYYYY');
  lc_price_obj.UNIT_SALES_PRICE      := 34;
  lc_price_obj.CURRENCY              := 'PLN';
  lc_price_obj.REASON_CODE           := '1';

  lc_price_tbl.extend();
  lc_price_tbl(lc_price_tbl.last) := lc_price_obj;

  -- 3rd Price change for testing:
  lc_price_obj                       := "XPTO_IN_PRICE_CHANGE_OBJ"();
  lc_price_obj.EVENT_TYPE            := 'UPSERT';
  lc_price_obj.PRODUCT_ID            := '777011115';
  lc_price_obj.LOCATION_ID           := 700610;
  lc_price_obj.PRICE_ZONE_ID         := NULL;
  lc_price_obj.COLABORATOR_ID        := 'DOG';
  lc_price_obj.SALES_PRICE_START_DATE := to_date('22052019', 'DDMMYYYY');
  lc_price_obj.UNIT_SALES_PRICE      := 34;
  lc_price_obj.CURRENCY              := 'PLN';
  lc_price_obj.REASON_CODE           := '1';

  lc_price_tbl.extend();
  lc_price_tbl(lc_price_tbl.last) := lc_price_obj;

  -- 4rd Price change for testing:
  lc_price_obj                       := "XPTO_IN_PRICE_CHANGE_OBJ"();
  lc_price_obj.EVENT_TYPE            := 'DELETE';
  lc_price_obj.PRODUCT_ID            := '49009650';
  lc_price_obj.LOCATION_ID           := 3600610;
  lc_price_obj.PRICE_ZONE_ID         := null;
  lc_price_obj.COLABORATOR_ID        := 'DOG';
  lc_price_obj.SALES_PRICE_START_DATE := to_date('02022019', 'DDMMYYYY');
  lc_price_obj.UNIT_SALES_PRICE      := 10;
  lc_price_obj.CURRENCY              := 'PLN';
  lc_price_obj.REASON_CODE           := '6';

  lc_price_tbl.extend();
  lc_price_tbl(lc_price_tbl.last) := lc_price_obj;

  XPTO_API_PRICECHANGE.UPLD_PC_TO_STG(lc_price_tbl,
                                      lc_status_code,
                                      lc_error_msg);

end;
```

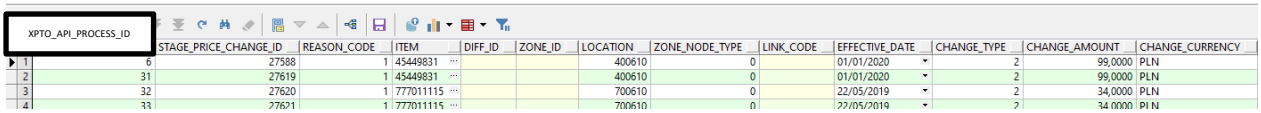
Figura 41- *Script* teste *Upload Price Change* (*Create, Update, Upsert, Delete*)

Após executar a script na base de dados, pode-se consultar o resumo da operação na tabela de auditoria que foi criada para armazenar os resultados durante as execuções, mostrando o *price change ID* que foi processada, um código do estado da execução e a respetiva mensagem de erro associada.



	API_NAME	API_PROCEDURE	ENTRY_TIMESTAMP	ENTRY_TEXT	ENTRY_STATUS	ENTRY_USER_ID
1	XPTO_API_PRICECHANGE	UPLD_PC_TO_STG	25-SEP-19 04.02.46.963019 PM	End of process!	0	
2		UPLD_PC_TO_STG	25-SEP-19 04.02.46.962713 PM	Record 4, Stage Price change 27622, Processed with Status: 0 |	0	
3		UPLD_PC_TO_STG	25-SEP-19 04.02.46.961735 PM	Record 3, Stage Price change 27621, Processed with Status: 0 |	0	
4		UPLD_PC_TO_STG	25-SEP-19 04.02.46.960391 PM	Record 2, Stage Price change 27620, Processed with Status: 0 |	0	
5		UPLD_PC_TO_STG	25-SEP-19 04.02.46.958923 PM	Record 1, Stage Price change 27619, Processed with Status: 0 |	0	
6		UPLD_PC_TO_STG	25-SEP-19 04.02.46.956757 PM	Process Started!		

Tabela 15 - XPTO_API_AUDIT



	XPTO_API_PROCESS_ID	STAGE_PRICE_CHANGE_ID	REASON_CODE	ITEM	DIFF_ID	ZONE_ID	LOCATION	ZONE_NODE_TYPE	LINK_CODE	EFFECTIVE_DATE	CHANGE_TYPE	CHANGE_AMOUNT	CHANGE_CURRENCY
1	6	27588	1	45449831			400610	0		01/01/2020	2	99,0000	PLN
2	31	27619	1	45449831			400610	0		01/01/2020	2	99,0000	PLN
3	32	27620	1	777011115			700610	0		22/05/2019	2	34,0000	PLN
4	33	27621	1	777011115			700610	0		22/05/2019	2	34,0000	PLN

Tabela 16 - XPTO_RPM_STAGE_PRICE_CHANGE

Como se pode ver na tabela *staging* do RPM, os records 1, 2 e 3, que correspondem a eventos *Create, Update e Upsert*, encontram-se na tabela e o record 4, que corresponde ao evento *Delete* não está na tabela pois foi eliminado.

Posto isto, pode-se concluir que o teste foi positivo.

XXRMS405 API *Cost Change Inbound* (PI2)

Objeto

Para esta API *Cost Change*, foi necessário criar dois objetos, um para suportar a estrutura de dados do *header* e outra para suportar a estrutura de dados do detalhe, isto porque para uma *cost change* temos sempre uma linha de *header* e poder-se-á ter uma ou mais linhas de detalhe associada a esse cabeçalho.

```
/*****************************************************************************/
-- CREATE DATE - OCTOBER 2019
-- CREATE USER - MARIO TEIXEIRA, JOAO VILAVERDE
-- PROJECT     - XPTO
-- DESCRIPTION - TABLE TYPE "XPTO_IN_COST_CHANGES_TBL"
/*****************************************************************************/

BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE XPTO_IN_COST_CHANGES_TBL';
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
--
CREATE OR REPLACE TYPE XPTO_IN_COST_CHANGES_TBL AS TABLE OF XPTO_IN_COST_CHANGE_OBJ
/

```

Figura 42- XPTO_IN_COST_CHANGES_TBL

```
/*****************************************************************************/
-- CREATE DATE - OCTOBER 2019
-- CREATE USER - MARIO TEIXEIRA, JOAO VILAVERDE
-- PROJECT     - XPTO
-- DESCRIPTION - OBJECT TYPE "XPTO_IN_COST_CHANGE_OBJ"
/*****************************************************************************/
BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE XPTO_IN_COST_CHANGES_TBL FORCE';
  EXECUTE IMMEDIATE 'DROP TYPE XPTO_IN_COST_CHANGE_OBJ FORCE';
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
--
--
CREATE OR REPLACE TYPE XPTO_IN_COST_CHANGE_OBJ AS OBJECT
(
  EVENT_TYPE               VARCHAR2(6),
  COST_CHANGE              NUMBER(8),
  COST_CHANGE_DESCRIPTION  VARCHAR2(120),
  REASON                   NUMBER(2),
  ACTIVE_DATE              DATE,
  STATUS                   VARCHAR(1),
  COST_CHANGE_ORIGIN       VARCHAR(3),
  STAGING_ID               NUMBER,
  COST_CHANGE_ID           NUMBER,
  STATUS_CODE              NUMBER,
  ERROR_MESSAGE            VARCHAR2(1000),
  IN_COST_CHANGES_DTL_TBL  XPTO_IN_COST_CHANGES_DTL_TBL,
--
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_OBJ RETURN SELF AS RESULT,
--
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_OBJ(
                               EVENT_TYPE               VARCHAR2,
                               COST_CHANGE              NUMBER,
                               COST_CHANGE_DESCRIPTION  VARCHAR2,
                               REASON                   NUMBER,
                               ACTIVE_DATE              DATE,
                               STATUS                   VARCHAR,
                               COST_CHANGE_ORIGIN       VARCHAR,
                               STAGING_ID               NUMBER,
                               COST_CHANGE_ID           NUMBER,
                               STATUS_CODE              NUMBER,
                               ERROR_MESSAGE            VARCHAR2,
                               IN_COST_CHANGES_DTL_TBL  XPTO_IN_COST_CHANGES_DTL_TBL)
    RETURN SELF AS RESULT
);
/
--
CREATE OR REPLACE TYPE BODY XPTO_IN_COST_CHANGE_OBJ AS
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_OBJ
    RETURN SELF AS RESULT IS
    BEGIN
      RETURN;
    END;
--
```

Figura 43- XPTO_IN_COST_CHANGE_OBJ (parte 1)

```
CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_OBJ(
                     EVENT_TYPE               VARCHAR2,
                     COST_CHANGE              NUMBER,
                     COST_CHANGE_DESCRIPTION  VARCHAR2,
                     REASON                   NUMBER,
                     ACTIVE_DATE              DATE,
                     STATUS                   VARCHAR,
                     COST_CHANGE_ORIGIN       VARCHAR,
                     STAGING_ID               NUMBER,
                     COST_CHANGE_ID           NUMBER,
                     STATUS_CODE              NUMBER,
                     ERROR_MESSAGE            VARCHAR2,
                     IN_COST_CHANGES_DTL_TBL  XPTO_IN_COST_CHANGES_DTL_TBL)

  RETURN SELF AS RESULT IS
    BEGIN
         SELF.EVENT_TYPE                 := EVENT_TYPE;
         SELF.COST_CHANGE                := COST_CHANGE;
         SELF.COST_CHANGE_DESCRIPTION    := COST_CHANGE_DESCRIPTION;
         SELF.REASON                     := REASON;
         SELF.ACTIVE_DATE                := ACTIVE_DATE;
         SELF.STATUS                     := STATUS;
         SELF.COST_CHANGE_ORIGIN         := COST_CHANGE_ORIGIN;
         SELF.STAGING_ID                 := STAGING_ID;
         SELF.COST_CHANGE_ID             := COST_CHANGE_ID;
         SELF.STATUS_CODE                := STATUS_CODE;
         SELF.ERROR_MESSAGE              := ERROR_MESSAGE;
         SELF.IN_COST_CHANGES_DTL_TBL    := IN_COST_CHANGES_DTL_TBL;
        RETURN;
      END;
--
END;
/
```

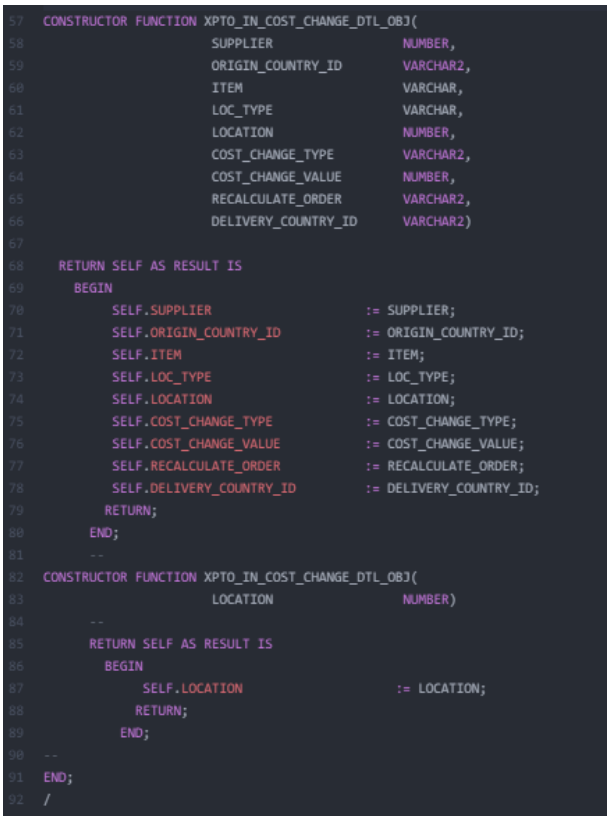
Figura 44- XPTO_IN_COST_CHANGE_OBJ (parte 2)

```
/*******************************************************************************/
-- CREATE DATE - OCTOBER 2019
-- CREATE USER - MARIO TEIXEIRA, JOAO VILAVERDE
-- PROJECT     - XPTO
-- DESCRIPTION - TABLE TYPE "XPTO_IN_COST_CHANGES_DTL_TBL"
/*******************************************************************************/

BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE XPTO_IN_COST_CHANGES_DTL_TBL';
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
--
CREATE OR REPLACE TYPE XPTO_IN_COST_CHANGES_DTL_TBL AS TABLE OF XPTO_IN_COST_CHANGE_DTL_OBJ
/
```

Figura 45- XPTO_IN_COST_CHANGES_DTL_TBL

```
/*******************************************************************************
-- CREATE DATE - OCTOBER 2019
-- CREATE USER - MARIO TEIXEIRA, JOAO VILAVERDE
-- PROJECT     - XPTO
-- DESCRIPTION - OBJECT TYPE "XPTO_IN_COST_CHANGE_DTL_OBJ"
*******************************************************************************/
BEGIN
  EXECUTE IMMEDIATE 'DROP TYPE XPTO_IN_COST_CHANGES_DTL_TBL FORCE';
  EXECUTE IMMEDIATE 'DROP TYPE XPTO_IN_COST_CHANGE_DTL_OBJ FORCE';
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
--
--
CREATE OR REPLACE TYPE XPTO_IN_COST_CHANGE_DTL_OBJ AS OBJECT
(
  SUPPLIER                NUMBER(10),
  ORIGIN_COUNTRY_ID       VARCHAR2(3),
  ITEM                    VARCHAR(25),
  LOC_TYPE                VARCHAR(1),
  LOCATION                NUMBER(10),
  COST_CHANGE_TYPE        VARCHAR2(2),
  COST_CHANGE_VALUE       NUMBER(20,4),
  RECALCULATE_ORDER       VARCHAR2(1),
  DELIVERY_COUNTRY_ID     VARCHAR2(3),
--
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_DTL_OBJ RETURN SELF AS RESULT,
--
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_DTL_OBJ(
                              SUPPLIER                NUMBER,
                              ORIGIN_COUNTRY_ID       VARCHAR2,
                              ITEM                    VARCHAR,
                              LOC_TYPE                VARCHAR,
                              LOCATION                NUMBER,
                              COST_CHANGE_TYPE        VARCHAR2,
                              COST_CHANGE_VALUE       NUMBER,
                              RECALCULATE_ORDER       VARCHAR2,
                              DELIVERY_COUNTRY_ID     VARCHAR2)
    RETURN SELF AS RESULT,
--
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_DTL_OBJ(
                              LOCATION                NUMBER)
    RETURN SELF AS RESULT
)
;
/
--
CREATE OR REPLACE TYPE BODY XPTO_IN_COST_CHANGE_DTL_OBJ AS
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_DTL_OBJ
    RETURN SELF AS RESULT IS
    BEGIN
      RETURN;
    END;
--
```

Figura 46- XPTO_IN_COST_CHANGES_DTL_OBJ (parte 1)



```
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_DTL_OBJ(
                      SUPPLIER                NUMBER,
                      ORIGIN_COUNTRY_ID       VARCHAR2,
                      ITEM                    VARCHAR,
                      LOC_TYPE                VARCHAR,
                      LOCATION                NUMBER,
                      COST_CHANGE_TYPE        VARCHAR2,
                      COST_CHANGE_VALUE       NUMBER,
                      RECALCULATE_ORDER       VARCHAR2,
                      DELIVERY_COUNTRY_ID     VARCHAR2)

    RETURN SELF AS RESULT IS
      BEGIN
          SELF.SUPPLIER                 := SUPPLIER;
          SELF.ORIGIN_COUNTRY_ID        := ORIGIN_COUNTRY_ID;
          SELF.ITEM                     := ITEM;
          SELF.LOC_TYPE                 := LOC_TYPE;
          SELF.LOCATION                 := LOCATION;
          SELF.COST_CHANGE_TYPE         := COST_CHANGE_TYPE;
          SELF.COST_CHANGE_VALUE        := COST_CHANGE_VALUE;
          SELF.RECALCULATE_ORDER        := RECALCULATE_ORDER;
          SELF.DELIVERY_COUNTRY_ID      := DELIVERY_COUNTRY_ID;
        RETURN;
      END;
      --
  CONSTRUCTOR FUNCTION XPTO_IN_COST_CHANGE_DTL_OBJ(
                      LOCATION                NUMBER)
      --
      RETURN SELF AS RESULT IS
        BEGIN
            SELF.LOCATION               := LOCATION;
          RETURN;
        END;
--
END;
/
```

Figura 47- XPTO_IN_COST_CHANGES_DTL_OBJ (parte 2)

Sequência

Para suportar esta API foi necessária a criação de uma sequência para se atribuir o *cost change ID*, provisório, a cada inserção de um registo novo na *staging*. Um *cost change ID* final será atribuído quando o registo for processado para as tabelas finais de *cost changes* no RMS.

```
/*****************************************************************************/
/* CREATE DATE - OCTOBER 2019                                                */
/* CREATE USER - JOAO VILAVERDE                                              */
/* PROJECT     - XPTO                                                        */
/* DESCRIPTION - DDL of "XPTO_COSTCHG_SVC_SEQ" sequence for XXRMS405         */
/*****************************************************************************/
BEGIN
  EXECUTE IMMEDIATE 'drop sequence XPTO_COSTCHG_SVC_SEQ';
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
--
CREATE SEQUENCE XPTO_COSTCHG_SVC_SEQ minvalue 1 maxvalue 999999999999999 start
  with 1 increment by 1 nocache;
```

Figura 46- XPTO_COSTCHG_SVC_SEQ

Package de Base de Dados

```
CREATE OR REPLACE PACKAGE XPTO_API_COSTCHANGE AS

  ---------------------------------------------------------------------------
  /*****************************************************************************/
  /* CREATE DATE - OCTOBER 2019                                                */
  /* CREATE USER - JOAO VILAVERDE, MARIO TEIXEIRA                              */
  /* PROJECT     - XPTO                                                        */
  /* DESCRIPTION - PACKAGE TO XXRMS405 - API Wrapper to support upload         */
  /*               operations of Cost Changes into the RMS                     */
  /*****************************************************************************/
  ---------------------------------------------------------------------------
  --Procedure Name : UPLD_COSTCHANGE
  --Purpose       : Support upload operations of Cost Changes into the RMS
  --                xpto_in_cost_change - Table type based on xpto_in_cost_change_obj
  ---------------------------------------------------------------------------
  PROCEDURE UPLD_COSTCHANGE(IO_xpto_in_cost_change IN OUT XPTO_IN_COST_CHANGES_TBL,
                            I_sync                 IN BOOLEAN,
                            O_status_code          OUT NUMBER,
                            O_error_desc           OUT VARCHAR2);

END XPTO_API_COSTCHANGE;
```

Figura 47- XPTO_API_COSTCHANGE.pks

```
1358  FUNCTION PROCESS_CREATE_CC(IO_XPTO_in_cost_change IN XPTO_IN_COST_CHANGE_OBJ,
1359                             O_seq_process_id         OUT NUMBER,
1360                             O_status_code            OUT NUMBER,
1361                             O_error_desc             OUT VARCHAR2)
1362    RETURN BOOLEAN IS
1363    --
1364    L_program              VARCHAR2(100)     := L_program_name || '.PROCESS_CREATE_CC';
1365    L_item_loc_exist       VARCHAR2(1);
1366    L_incr_row_seq         NUMBER            := 0;
1367    L_incr_to_range        NUMBER            := 0;
1368    L_incr_header          NUMBER            := 0;
1369    L_O_error_desc         VARCHAR2(1000);
1370    L_O_status_code        NUMBER;
1371    L_action               VARCHAR2(3)       :='NEW';
1372    L_seq_process_id       NUMBER            := CORESVC_ITEM_PSEQ.nextval;
1373    L_seq_costchange       NUMBER            := XPTO_COSTCHG_SVC_SEQ.nextval;
1374    L_specific_location    NUMBER(10);
1375    --
1376  BEGIN
1377    --
1378    L_O_error_desc := NULL;
1379     --
1380     IF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).loc_type = LP_STORE AND
1381        IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NULL
1382      THEN
1383       --
1384       OPEN C_loc_store_cc(IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).item);
1385       LOOP
1386         FETCH C_loc_store_cc
1387           INTO L_st_locations;
1388         EXIT when C_loc_store_cc%NOTFOUND;
1389         --
1390         -- check if there is stores associated to the item
1391         --
1392         IF L_st_locations IS NOT NULL
1393          THEN
1394            --
1395          L_incr_row_seq := L_incr_row_seq + 1;
1396          --
1397          IF NOT CREATE_CC_REC(IO_XPTO_in_cost_change,
1398                               L_incr_header,
1399                               L_incr_row_seq,
1400                               L_action,
1401                               L_st_locations,
1402                               L_seq_process_id,
1403                               L_seq_costchange,
1404                               L_O_status_code,
1405                               L_O_error_desc) THEN
1406             --
1407             O_status_code    := L_O_status_code;
1408             O_error_desc     := L_O_error_desc;
1409             RETURN FALSE;
1410             --
1411          --
1412          END IF;
1413         --
1414        ELSE
1415          --
1416          O_status_code := 1;
1417          O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'NOT_FOUND_LOC_S_CC',
1418                                                    I_txt_1 => NULL,
1419                                                    I_txt_2 => L_program,
1420                                                    I_txt_3 => NULL);
1421          --
1422          RETURN FALSE;
1423          --
1424         --
1425        END IF;
1426       --
1427       END LOOP;
1428       --
1429       CLOSE C_loc_store_cc;
```

Figura 48- Função Process_Create_CC *API Cost Change* (parte 1)

```
ELSIF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).loc_type = LP_WAREHOUSE AND
      IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NULL
   THEN
    --
    OPEN C_loc_wh_cc(IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).item);
    LOOP
      FETCH C_loc_wh_cc
        INTO L_wh_locations;
      EXIT when C_loc_wh_cc%NOTFOUND;
      --
      IF L_wh_locations IS NOT NULL
      THEN
        --
        L_incr_row_seq := L_incr_row_seq + 1;
        --
        IF NOT CREATE_CC_REC(IO_XPTO_in_cost_change,
                             L_incr_header,
                             L_incr_row_seq,
                             L_action,
                             L_wh_locations,
                             L_seq_process_id,
                             L_seq_costchange,
                             L_O_status_code,
                             L_O_error_desc) THEN
          --
          O_status_code    := L_O_status_code;
          O_error_desc     := L_O_error_desc;
          RETURN FALSE;
          --
        --
        END IF;
      --
    ELSE
      O_status_code := 1;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'NOT_FOUND_LOC_W_CC',
                                                I_txt_1 => NULL,
                                                I_txt_2 => L_program,
                                                I_txt_3 => NULL);
      --
      RETURN FALSE;
      --
    END IF;
    --
    END LOOP;
    --
    CLOSE C_loc_wh_cc;
    --
ELSIF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).loc_type = LP_BOTH AND
      IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NULL
  THEN
    --
    IF NOT CREATE_CC_ITEM_REC(IO_XPTO_in_cost_change,
                              L_action,
                              L_seq_process_id,
                              L_seq_costchange,
                              L_O_status_code,
                              L_O_error_desc) THEN
      --
      O_status_code    := L_O_status_code;
      O_error_desc     := L_O_error_desc;
      RETURN FALSE;
      --
    --
    END IF;
  --
  -- Case the location field is populated the cost change will be apllied to a specific location
  --
ELSIF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NOT NULL
  THEN
    --
  FOR i IN 1 .. IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL.count LOOP
   --
  OPEN C_item_loc_exist(IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).location,
                        IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).item);
  FETCH C_item_loc_exist
    INTO L_item_loc_exist;
  CLOSE C_item_loc_exist;
```

Figura 49- Função Process_Create_CC *API Cost Change* (parte 2)

```
        --
        --
        IF L_item_loc_exist IS NULL THEN
          --
          L_incr_to_range := L_incr_to_range + 1;
          --
          L_cost_rec.item              := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).item;
          L_cost_rec.supplier          := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).supplier;
          L_cost_rec.origin_country_id := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).origin_country_id;
          L_cost_rec.loc_type          := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).loc_type;
          L_cost_rec.location          := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).location;
          L_cost_rec.pack_type         := null;
          L_cost_rec.unit_cost         := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).COST_CHANGE_UNIT_COST;
          L_cost_rec.row_seq           := null;
          --
          L_item_locs(L_incr_to_range) := L_cost_rec;
          --
          --create item-location relation
          --
          IF CREATE_ITEMLOC(L_O_error_desc, L_seq_process_id, L_item_locs) = FALSE
           THEN
            --
            O_status_code := 1;
            O_error_desc  := L_O_error_desc;
            RETURN FALSE;
            --
          END IF;
          --
        END IF;
        --
        L_specific_location := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).location;
        --
        L_incr_row_seq := L_incr_row_seq + 1;
        --
        IF NOT CREATE_CC_LOC_REC(IO_XPTO_in_cost_change,
                                 L_incr_header,
                                 i,
                                 L_action,
                                 L_specific_location,
                                 L_seq_process_id,
                                 L_seq_costchange,
                                 L_O_status_code,
                                 L_O_error_desc) THEN
           --
           O_status_code    := L_O_status_code;
           O_error_desc     := L_O_error_desc;
           --
           RETURN FALSE;
           --
        --
        END IF;
        --
    END LOOP;
    --
    END IF;
    --
    O_status_code    := 0;
    O_seq_process_id := L_seq_process_id;
    O_error_desc     := L_O_error_desc;
    --
    RETURN TRUE;
    --
  EXCEPTION
    WHEN OTHERS THEN
      --
      O_status_code := 255;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                                I_txt_1 => SQLERRM,
                                                I_txt_2 => L_program,
                                                I_txt_3 => TO_CHAR(SQLCODE));
      --
      RETURN FALSE;
      --
    --
  END PROCESS_CREATE_CC;
```

Figura 50- Função Process_Create_CC *API Cost Change* (parte 3)

```
FUNCTION PROCESS_UPDATE_CC(IO_XPTO_in_cost_change IN OUT XPTO_IN_COST_CHANGE_OBJ,
                           O_seq_process_id         OUT NUMBER,
                           O_status_code            OUT NUMBER,
                           O_error_desc             OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program              VARCHAR2(100)     := L_program_name || '.PROCESS_UPDATE_CC';
  L_item_loc_exist       VARCHAR2(10);
  L_incr_to_range        NUMBER            := 0;
  L_incr_header          NUMBER            := 0;
  L_O_error_desc         VARCHAR2(1000);
  L_O_status_code        NUMBER;
  L_seq_process_id       NUMBER            := CORESVC_ITEM_PSEQ.nextval;
  L_incr_row_seq         NUMBER            := 0;
  L_check_cost_change    VARCHAR2(1);
  L_action               VARCHAR2(3)       := 'MOD';
  L_specific_location    NUMBER(10);
  L_cc_id                NUMBER(8)         := IO_XPTO_in_cost_change.cost_change;
  L_check_cc_status      VARCHAR2(1);
  L_original_status      VARCHAR2(1)       := IO_XPTO_in_cost_change.status;
  --
  -- cursor to check if the cost change is present in the staging table
  --
  cursor C_check_svc_costchange is
    select 'Y'
      from svc_cost_susp_sup_head
     where cost_change = IO_XPTO_in_cost_change.cost_change;
     --
  cursor C_check_cc_status is
   select 'Y'
     from cost_susp_sup_head
    where cost_change = IO_XPTO_in_cost_change.cost_change
      and status = 'A';
    --
BEGIN
  --
  L_O_error_desc := NULL;
  --
  --Delete from staging tables
  --Common code to (S, W, LOCATION)
  --
  IF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).loc_type = LP_STORE OR
     IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).loc_type = LP_WAREHOUSE OR
     IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NOT NULL
   THEN
    --
    OPEN C_check_svc_costchange;
    FETCH C_check_svc_costchange
      INTO L_check_cost_change;
   CLOSE C_check_svc_costchange;
   --Test if the cost change exists in staging table
   IF L_check_cost_change IS NOT NULL
    THEN
      --
      delete from svc_cost_susp_sup_head
      where cost_change = IO_XPTO_in_cost_change.cost_change;
      --
      delete from svc_cost_susp_sup_detail_loc
      where cost_change = IO_XPTO_in_cost_change.cost_change;
      --
   END IF;
   --
  END IF;
  --
  --  /*For loc_type -> S */
  --
  IF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).loc_type = LP_STORE AND
     IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NULL
   THEN
     --
     --
     -- ItÂ´s not possible to update a cost change with status 'A' so,
     -- itÂ´s necessary first update to 'W' and then to 'A'
     OPEN C_check_cc_status;
       FETCH C_check_cc_status
         INTO L_check_cc_status;
     CLOSE C_check_cc_status;
```

Figura 51- Função Process_Update_CC API Cost Change (parte 1)

```
1661        --
1662        IF L_check_cc_status IS NOT NULL AND IO_XPTO_in_cost_change.event_type <> LP_CANCEL -- status 'A'
1663         THEN
1664           --
1665           IO_XPTO_in_cost_change.status := 'W';
1666           -- get stores to process
1667            OPEN C_loc_store_cc(IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).item);
1668            LOOP
1669              FETCH C_loc_store_cc
1670                INTO L_st_locations;
1671                EXIT when C_loc_store_cc%NOTFOUND;
1672                --
1673                -- check if there are stores associated to the item
1674                --
1675                IF L_st_locations IS NOT NULL
1676                THEN
1677                  --
1678                  L_incr_row_seq := L_incr_row_seq + 1;
1679                  --
1680                    IF NOT CREATE_CC_REC(IO_XPTO_in_cost_change,
1681                                         L_incr_header,
1682                                         L_incr_row_seq,
1683                                         L_action,
1684                                         L_st_locations,
1685                                         L_seq_process_id,
1686                                         L_cc_id,
1687                                         L_O_status_code,
1688                                         L_O_error_desc) THEN
1689                                         --
1690                     O_status_code := L_O_status_code;
1691                     O_error_desc  := L_O_error_desc;
1692                     RETURN FALSE;
1693                     --
1694                    END IF;
1695                 --
1696                ELSE
1697                  O_status_code := 1;
1698                  O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'NOT_FOUND_LOC_S_CC',
1699                                                            I_txt_1 => NULL,
1700                                                            I_txt_2 => L_program,
1701                                                            I_txt_3 => NULL);
1702                  --
1703                 RETURN FALSE;
1704                 --
1705               END IF;
1706              --
1707            END LOOP;
1708             --
1709            CLOSE C_loc_store_cc;
1710           --restore values
1711           IO_XPTO_in_cost_change.status := L_original_status;
1712           L_incr_row_seq                := 0;
1713           L_incr_header                 := 0;
1714           --
1715           -- Process the cost change to final table with status = 'W'
1716           --
1717           IF CORESVC_COSTCHG.PROCESS(L_O_error_desc,
1718                                      L_seq_process_id) =  FALSE THEN
1719                                     --
1720            O_status_code := 2;
1721            O_error_desc  := L_O_error_desc;
1722            --
1723            RETURN FALSE;
1724            --
1725           END IF;
1726           -- Delete records from staging tables to insert new records
1727            delete from svc_cost_susp_sup_head
1728             where cost_change = IO_XPTO_in_cost_change.cost_change;
1729             --
1730            delete from svc_cost_susp_sup_detail_loc
1731             where cost_change = IO_XPTO_in_cost_change.cost_change;
1732             --
1733          --
1734        END IF;
```

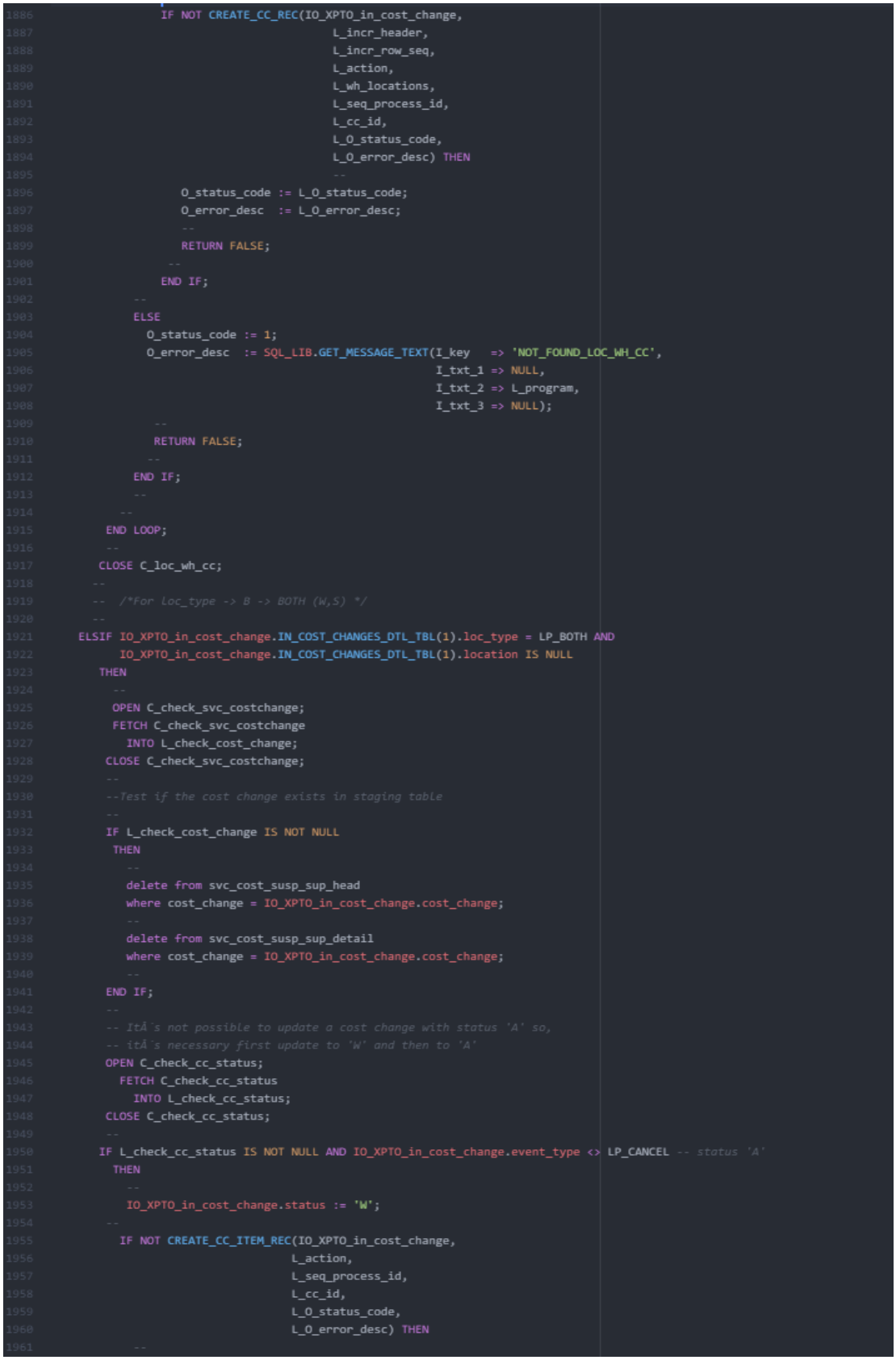
Figura 52- Função Process_Update_CC *API Cost Change* (parte 2)

```
1735        --
1736        -- Update record to the original status
1737        -- get stores to process
1738        OPEN C_loc_store_cc(IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).item);
1739        LOOP
1740          FETCH C_loc_store_cc
1741            INTO L_st_locations;
1742            EXIT when C_loc_store_cc%NOTFOUND;
1743            --
1744            -- check if there are stores associated to the item
1745            --
1746            IF L_st_locations IS NOT NULL
1747            THEN
1748              --
1749              L_incr_row_seq := L_incr_row_seq + 1;
1750              --
1751                IF NOT CREATE_CC_REC(IO_XPTO_in_cost_change,
1752                                     L_incr_header,
1753                                     L_incr_row_seq,
1754                                     L_action,
1755                                     L_st_locations,
1756                                     L_seq_process_id,
1757                                     L_cc_id,
1758                                     L_O_status_code,
1759                                     L_O_error_desc) THEN
1760                                     --
1761                   O_status_code := L_O_status_code;
1762                   O_error_desc  := L_O_error_desc;
1763                   --
1764                   RETURN FALSE;
1765                 --
1766                END IF;
1767            --
1768            ELSE
1769              O_status_code := 1;
1770              O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'NOT_FOUND_LOC_S_CC',
1771                                                        I_txt_1 => NULL,
1772                                                        I_txt_2 => L_program,
1773                                                        I_txt_3 => NULL);
1774               --
1775               RETURN FALSE;
1776              --
1777            END IF;
1778            --
1779          --
1780        END LOOP;
1781        --
1782       CLOSE C_loc_store_cc;
1783       --
1784       --  /*For loc_type -> W */
1785       --
1786      ELSIF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).loc_type = LP_WAREHOUSE AND
1787            IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NULL
1788        THEN
1789         --
1790         -- ItÂ´s not possible to update a cost change with status 'A' so,
1791         -- itÂ´s necessary first update to 'W' and then to 'A'
1792         OPEN C_check_cc_status;
1793           FETCH C_check_cc_status
1794             INTO L_check_cc_status;
1795         CLOSE C_check_cc_status;
1796         --
1797        IF L_check_cc_status IS NOT NULL AND IO_XPTO_in_cost_change.event_type <> LP_CANCEL -- status 'A'
1798          THEN
1799            --
1800            IO_XPTO_in_cost_change.status := 'W';
1801            -- get wh to process
1802             OPEN C_loc_wh_cc(IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).item);
1803             LOOP
1804               FETCH C_loc_wh_cc
1805                 INTO L_wh_locations;
1806                 EXIT when C_loc_wh_cc%NOTFOUND;
1807                 --
1808                 -- check if there are wh associated to the item
1809                 --
1810                 IF L_wh_locations IS NOT NULL
1811                 THEN
```

Figura 53- Função Process_Update_CC API Cost Change (parte 3)

```
                --
                L_incr_row_seq := L_incr_row_seq + 1;
                --
                  IF NOT CREATE_CC_REC(IO_XPTO_in_cost_change,
                                       L_incr_header,
                                       L_incr_row_seq,
                                       L_action,
                                       L_wh_locations,
                                       L_seq_process_id,
                                       L_cc_id,
                                       L_O_status_code,
                                       L_O_error_desc) THEN
                                       --
                    O_status_code := L_O_status_code;
                    O_error_desc  := L_O_error_desc;
                    RETURN FALSE;
                    --
                  END IF;
                --
              ELSE
                O_status_code := 1;
                O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'NOT_FOUND_LOC_WH_CC',
                                                          I_txt_1 => NULL,
                                                          I_txt_2 => L_program,
                                                          I_txt_3 => NULL);
                --
               RETURN FALSE;
               --
             END IF;
            --
          END LOOP;
           --
          CLOSE C_loc_wh_cc;
         --restore values
        IO_XPTO_in_cost_change.status := L_original_status;
        L_incr_row_seq                := 0;
        L_incr_header                 := 0;
         --
         -- Process the cost change to final table with status = 'W'
         --
         IF CORESVC_COSTCHG.PROCESS(L_O_error_desc,
                                    L_seq_process_id) =  FALSE THEN
                                   --
          O_status_code := 2;
          O_error_desc  := L_O_error_desc;
          --
          RETURN FALSE;
          --
         END IF;
         -- Delete records from staging tables to insert new records
          delete from svc_cost_susp_sup_head
           where cost_change = IO_XPTO_in_cost_change.cost_change;
           --
          delete from svc_cost_susp_sup_detail_loc
           where cost_change = IO_XPTO_in_cost_change.cost_change;
           --
       --
      END IF;
      --
      -- Update record to the original status
      -- get wh to process
      OPEN C_loc_wh_cc(IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).item);
      LOOP
        FETCH C_loc_wh_cc
          INTO L_wh_locations;
          EXIT when C_loc_wh_cc%NOTFOUND;
          --
          -- check if there are wh associated to the item
          --
          IF L_wh_locations IS NOT NULL
          THEN
            --
            L_incr_row_seq := L_incr_row_seq + 1;
            --
```

Figura 54- Função Process_Update_CC *API Cost Change* (parte 4)



```
                IF NOT CREATE_CC_REC(IO_XPTO_in_cost_change,
                                     L_incr_header,
                                     L_incr_row_seq,
                                     L_action,
                                     L_wh_locations,
                                     L_seq_process_id,
                                     L_cc_id,
                                     L_O_status_code,
                                     L_O_error_desc) THEN
                                     --
                   O_status_code := L_O_status_code;
                   O_error_desc  := L_O_error_desc;
                   --
                   RETURN FALSE;
                  --
                END IF;
              --
              ELSE
                O_status_code := 1;
                O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'NOT_FOUND_LOC_WH_CC',
                                                          I_txt_1 => NULL,
                                                          I_txt_2 => L_program,
                                                          I_txt_3 => NULL);
                 --
                 RETURN FALSE;
                --
              END IF;
              --
            --
          END LOOP;
          --
        CLOSE C_loc_wh_cc;
       --
       --  /*For loc_type -> B -> BOTH (W,S) */
       --
     ELSIF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).loc_type = LP_BOTH AND
           IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NULL
       THEN
         --
         OPEN C_check_svc_costchange;
         FETCH C_check_svc_costchange
           INTO L_check_cost_change;
        CLOSE C_check_svc_costchange;
        --
        --Test if the cost change exists in staging table
        --
        IF L_check_cost_change IS NOT NULL
         THEN
           --
           delete from svc_cost_susp_sup_head
           where cost_change = IO_XPTO_in_cost_change.cost_change;
           --
           delete from svc_cost_susp_sup_detail
           where cost_change = IO_XPTO_in_cost_change.cost_change;
           --
        END IF;
        --
        -- ItÂ´s not possible to update a cost change with status 'A' so,
        -- itÂ´s necessary first update to 'W' and then to 'A'
        OPEN C_check_cc_status;
          FETCH C_check_cc_status
            INTO L_check_cc_status;
        CLOSE C_check_cc_status;
        --
       IF L_check_cc_status IS NOT NULL AND IO_XPTO_in_cost_change.event_type <> LP_CANCEL -- status 'A'
         THEN
           --
           IO_XPTO_in_cost_change.status := 'W';
        --
          IF NOT CREATE_CC_ITEM_REC(IO_XPTO_in_cost_change,
                                    L_action,
                                    L_seq_process_id,
                                    L_cc_id,
                                    L_O_status_code,
                                    L_O_error_desc) THEN
             --
```

Figura 55- Função Process_Update_CC *API Cost Change* (parte 5)

```
            O_status_code := L_O_status_code;
            O_error_desc  := L_O_error_desc;
            RETURN FALSE;
            --
          --
          END IF;
          --
        IO_XPTO_in_cost_change.status := L_original_status;
        --
        --
        -- Process the cost change to final table with status = 'W'
        --
        IF CORESVC_COSTCHG.PROCESS(L_O_error_desc,
                                   L_seq_process_id) =  FALSE THEN
                                  --
          O_status_code := 2;
          O_error_desc  := L_O_error_desc;
          --
          RETURN FALSE;
          --
        END IF;
        --
        delete from svc_cost_susp_sup_head
        where cost_change = IO_XPTO_in_cost_change.cost_change;
        --
        delete from svc_cost_susp_sup_detail
        where cost_change = IO_XPTO_in_cost_change.cost_change;
        --
      END IF;
      --
      IF NOT CREATE_CC_ITEM_REC(IO_XPTO_in_cost_change,
                                L_action,
                                L_seq_process_id,
                                L_cc_id,
                                L_O_status_code,
                                L_O_error_desc) THEN
        --
        O_status_code := L_O_status_code;
        O_error_desc  := L_O_error_desc;
        RETURN FALSE;
        --
      --
      END IF;
      --
      --  /*For a specific location*/
      --
    ELSIF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).location IS NOT NULL
    THEN
      --
      IF IO_XPTO_in_cost_change.EVENT_TYPE <> LP_CANCEL
      THEN
        --
        FOR i IN 1 .. IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL.count LOOP
        --
        OPEN C_item_loc_exist(IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).location,
                              IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).item);
        FETCH C_item_loc_exist
          INTO L_item_loc_exist;
        CLOSE C_item_loc_exist;
        --
        IF L_item_loc_exist IS NULL THEN
          --
          L_incr_to_range := L_incr_to_range + 1;
          --
          L_cost_rec.item              := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).item;
          L_cost_rec.supplier          := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).supplier;
          L_cost_rec.origin_country_id := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).origin_country_id;
          L_cost_rec.loc_type          := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).loc_type;
          L_cost_rec.location          := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).location;
          L_cost_rec.pack_type         := null;
          L_cost_rec.unit_cost         := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).COST_CHANGE_UNIT_COST;
          L_cost_rec.row_seq           := i;
          --
          L_item_locs(L_incr_to_range) := L_cost_rec;
          --
          --create item-location relationship
          --
```

Figura 56- Função Process_Update_CC *API Cost Change* (parte 6)

```
          IF CREATE_ITEMLOC(L_O_error_desc, L_seq_process_id, L_item_locs) =
             FALSE THEN
            --
            O_status_code := 1;
            O_error_desc  := L_O_error_desc;
            RETURN FALSE;
            --
          END IF;
          --
        END IF;
        --
      END LOOP;
      --
      -- Its not possible to update a cost change with status 'A' so,
      -- its necessary first update to 'W' and then to 'A'
      OPEN C_check_cc_status;
        FETCH C_check_cc_status
         INTO L_check_cc_status;
      CLOSE C_check_cc_status;
      --
      IF L_check_cc_status IS NOT NULL  -- status 'A'
       THEN
        --
        IO_XPTO_in_cost_change.status := 'W';
        --
        FOR z in 1 .. IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL.COUNT LOOP
        --
        L_specific_location := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(z).location;
        --
         IF NOT CREATE_CC_LOC_REC(IO_XPTO_in_cost_change,
                                  L_incr_header,
                                  z,
                                  L_action,
                                  L_specific_location,
                                  L_seq_process_id,
                                  L_cc_id,
                                  L_O_status_code,
                                  L_O_error_desc) THEN
            --
            O_status_code := L_O_status_code;
            O_error_desc  := L_O_error_desc;
            RETURN FALSE;
         --
         END IF;
         --
        END LOOP;
       --
          IO_XPTO_in_cost_change.status := L_original_status;
          L_incr_header                 := 0;
          --
          -- Process the cost change to final table with status = 'W'
          --
          IF CORESVC_COSTCHG.PROCESS(L_O_error_desc,
                                     L_seq_process_id) =  FALSE THEN
                                    --
            O_status_code := 2;
            O_error_desc  := L_O_error_desc;
            --
            RETURN FALSE;
            --
          END IF;

         -- Delete records from staging tables to insert new records
         delete from svc_cost_susp_sup_head
          where cost_change = IO_XPTO_in_cost_change.cost_change;
          --
         delete from svc_cost_susp_sup_detail_loc
          where cost_change = IO_XPTO_in_cost_change.cost_change;
          --
      END IF;
      --
    END IF;
    --
      FOR k in 1 .. IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL.COUNT LOOP
      --
      L_specific_location := IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(k).location;
```

Figura 57- Função Process_Update_CC *API Cost Change* (parte 7)

```
2115          --
2116          IF NOT CREATE_CC_LOC_REC(IO_XPTO_in_cost_change,
2117                                   L_incr_header,
2118                                   k,
2119                                   L_action,
2120                                   L_specific_location,
2121                                   L_seq_process_id,
2122                                   L_cc_id,
2123                                   L_O_status_code,
2124                                   L_O_error_desc) THEN
2125             --
2126             O_status_code := L_O_status_code;
2127             O_error_desc  := L_O_error_desc;
2128             RETURN FALSE;
2129          --
2130          END IF;
2131          --
2132       END LOOP;
2133       --
2134       ELSE
2135          O_status_code := 1;
2136          O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'FAIL_TO_UPDATE_CC',
2137                                                    I_txt_1 => NULL,
2138                                                    I_txt_2 => L_program,
2139                                                    I_txt_3 => NULL);
2140          --
2141          RETURN FALSE;
2142          --
2143       END IF;
2144       --
2145       O_status_code    := 0;
2146       O_seq_process_id := L_seq_process_id;
2147       O_error_desc     := L_O_error_desc;
2148       --
2149       RETURN TRUE;
2150       --
2151    EXCEPTION
2152       WHEN OTHERS THEN
2153          --
2154          O_status_code := 255;
2155          O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
2156                                                    I_txt_1 => SQLERRM,
2157                                                    I_txt_2 => L_program,
2158                                                    I_txt_3 => TO_CHAR(SQLCODE));
2159          --
2160          RETURN FALSE;
2161          --
2162    END PROCESS_UPDATE_CC;
```

Figura 58- Função Process_Update_CC *API Cost Change* (parte 8)

```
FUNCTION PROCESS_UPSERT_CC(I_XPTO_in_cost_change  IN  OUT XPTO_IN_COST_CHANGE_OBJ,
                           L_O_process_id           OUT NUMBER,
                           O_status_code            OUT NUMBER,
                           O_error_desc             OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program              VARCHAR2(100) := L_program_name || '.PROCESS_UPSERT_CC';
  L_O_error_desc         VARCHAR2(1000);
  L_O_status_code        NUMBER;
  L_O_process_id_upsert  NUMBER;
  --
BEGIN
  --
  L_O_error_desc := NULL;
  --
    IF I_XPTO_in_cost_change.cost_change IS NOT NULL THEN
      --
      IF XPTO_API_COSTCHANGE.PROCESS_UPDATE_CC(I_XPTO_in_cost_change,
                                               L_O_process_id_upsert,
                                               L_O_status_code,
                                               L_O_error_desc) = FALSE THEN
        --
        O_status_code  := L_O_status_code;
        L_O_process_id := L_O_process_id_upsert;
        O_error_desc   := L_O_error_desc;
        --
        RETURN FALSE;
        --
      END IF;
      --
    ELSIF I_XPTO_in_cost_change.cost_change IS NULL THEN
      --
      IF XPTO_API_COSTCHANGE.PROCESS_CREATE_CC(I_XPTO_in_cost_change,
                                               L_O_process_id_upsert,
                                               L_O_status_code,
                                               L_O_error_desc) = FALSE THEN
        --
        O_status_code  := L_O_status_code;
        L_O_process_id := L_O_process_id_upsert;
        O_error_desc   := L_O_error_desc;
        --
        RETURN FALSE;
        --
      END IF;
      --
    END IF;
  --
  O_status_code    := 0;
  L_O_process_id   := L_O_process_id_upsert;
  O_error_desc     := L_O_error_desc;
  RETURN TRUE;
  --
EXCEPTION
  --
  WHEN OTHERS THEN
    --
    O_status_code := 255;
    O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                              I_txt_1 => SQLERRM,
                                              I_txt_2 => L_program,
                                              I_txt_3 => TO_CHAR(SQLCODE));
    --
    RETURN FALSE;
    --
END PROCESS_UPSERT_CC;
```

Figura 59- Função Process_Upsert_CC *API Cost Change*

```
FUNCTION PROCESS_CANCEL_CC(I_XPTO_in_cost_change  IN OUT  XPTO_IN_COST_CHANGE_OBJ,
                           L_O_process_id           OUT NUMBER,
                           O_status_code            OUT NUMBER,
                           O_error_desc             OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program VARCHAR2(100) := L_program_name || '.PROCESS_CANCEL_PC';
  L_O_process_id_cancel  NUMBER;
  L_O_error_desc         VARCHAR2(1000);
  L_O_status_code        NUMBER;
  --
BEGIN
  --
    IF PROCESS_UPDATE_CC(I_XPTO_in_cost_change,
                         L_O_process_id_cancel,
                         L_O_status_code,
                         L_O_error_desc) = FALSE THEN
                         --
      O_status_code  := L_O_status_code;
      L_O_process_id := L_O_process_id_cancel;
      O_error_desc   := L_O_error_desc;
      RETURN FALSE;
      --
    END IF;
  --
  O_status_code  := 0;
  L_O_process_id := L_O_process_id_cancel;
  O_error_desc   := L_O_error_desc;
  --
  RETURN TRUE;
  --
 EXCEPTION
   WHEN OTHERS THEN
     --
     O_status_code := 255;
     O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                               I_txt_1 => SQLERRM,
                                               I_txt_2 => L_program,
                                               I_txt_3 => TO_CHAR(SQLCODE));
     --
     RETURN FALSE;
     --
END PROCESS_CANCEL_CC;
```

Figura 60- Função Process_Cancel_CC *API Cost Change*

```
FUNCTION SYSTEM_VALIDATION_CC(I_XPTO_in_cost_change IN OUT XPTO_IN_COST_CHANGES_TBL,
                              L_I_sync               IN BOOLEAN,
                              O_status_code          OUT NUMBER,
                              O_error_desc           OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program                   VARCHAR2(100) := L_program_name || '.SYSTEM_VALIDATION_CC';
  L_max_sync_no_rec_upld_rec  XPTO_API_SETTINGS.MAX_SYNC_UPLD_REC%TYPE;
  L_max_async_no_rec_upld_rec XPTO_API_SETTINGS.MAX_ASYNC_UPLD_REC%TYPE;
  L_check_cc_origin           VARCHAR(3);
  L_check_cc_description      VARCHAR(120);
  --
  /* CURSOR to get the settings of the API */
  CURSOR C_api_settings IS
    SELECT max_sync_upld_rec, max_async_upld_rec
      FROM XPTO_api_settings
     WHERE api_name = 'XPTO_API_COSTCHANGE';
    --
    -- Cursors to check default values in XPTO_MOM_DVM
    --
    CURSOR C_check_cc_origin IS
      select value_1
        from XPTO_mom_dvm
       where func_area = 'CC_API_DEFAULTS'
         and parameter = 'COST_CHANGE_ORIGIN';
    --
    CURSOR C_check_cc_description IS
      select value_1
        from XPTO_mom_dvm
       where func_area = 'CC_API_DEFAULTS'
         and parameter = 'COST_CHANGE_DESCRIPTION';
  --
BEGIN
  --
  OPEN C_api_settings;
  FETCH C_api_settings
    INTO L_max_sync_no_rec_upld_rec, L_max_async_no_rec_upld_rec;
  CLOSE C_api_settings;
  --
  IF L_I_sync = TRUE THEN
  --
    IF L_max_sync_no_rec_upld_rec IS NULL THEN
      --
      O_status_code := 1;
      --
      O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'MAX_SYNC_NO_REC_NULL',
                                               I_txt_1 => NULL,
                                               I_txt_2 => L_program,
                                               I_txt_3 => NULL);
      --
      RETURN TRUE;
      --
    END IF;
    --
    IF L_max_sync_no_rec_upld_rec < I_XPTO_in_cost_change.count THEN
      --
      O_status_code := 1;
      --
      O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'MAX_SYNC_NO_REC_LESS_CC',
                                               I_txt_1 => NULL,
                                               I_txt_2 => L_program,
                                               I_txt_3 => NULL);
      --
      RETURN TRUE;
      --
    END IF;
  --
ELSIF L_I_sync = FALSE THEN
  --
  IF L_max_async_no_rec_upld_rec IS NULL THEN
    --
    O_status_code := 1;
    --
```

Figura 61- Função System_Validation_CC *API Cost Change* (Validações de sistema parte 1)

Figura 62- Função System_Validation_CC *API Cost Change* (Validações de sistema parte 2)

```
        --
        IF L_check_cc_description IS NOT NULL
          THEN
          --
            I_XPTO_in_cost_change(i).cost_change_description := L_check_cc_description;
          --
        ELSE
          --
          O_status_code := 1;
          --
          O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'CC_DESC_FAIL_DEFAULT',
                                                   I_txt_1 => NULL,
                                                   I_txt_2 => L_program,
                                                   I_txt_3 => NULL);
          --
          RETURN TRUE;
          --
        END IF;
      --
      END IF;
    --
  END LOOP;
  --
  RETURN TRUE;
  --
  EXCEPTION
    WHEN OTHERS THEN
      --
      O_status_code := 255;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                                I_txt_1 => SQLERRM,
                                                I_txt_2 => L_program,
                                                I_txt_3 => TO_CHAR(SQLCODE));
      --
      RETURN FALSE;
      --
  END SYSTEM_VALIDATION_CC;
```

Figura 63- Função System_Validation_CC *API Cost Change* (Validações de sistema parte 3)

```
FUNCTION VALIDATE_ENRICH_CC(IO_XPTO_in_cost_change IN OUT XPTO_IN_COST_CHANGE_OBJ,
                            O_status_code           OUT NUMBER,
                            O_error_desc            OUT VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_program              VARCHAR2(100) := L_program_name ||'.VALIDATE_ENRICH_CC';
  L_valid_origin_country VARCHAR2(1);
  L_check_origin_country VARCHAR2(3);
  L_count_loc_null       NUMBER        := 0;
  L_count_loc_not_null   NUMBER        := 0;
  --
  -- Cursors to check origin country id value
  --
  CURSOR C_check_origin_country IS
    select  isc.origin_country_id
      from item_supp_country isc
     where IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).item     = isc.item
       and IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).supplier = isc.SUPPLIER
       and isc.primary_country_ind                                    = 'Y';
  --
  CURSOR C_validate_origin_country IS
    select  'X'
      from item_supp_country isc
     where IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).item              = isc.item
       and IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).supplier          = isc.SUPPLIER
       and IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).origin_country_id = isc.origin_country_id
       and isc.primary_country_ind = 'Y';
--
BEGIN
  --
  -- Validate mandatory field on header
  --
  IF IO_XPTO_in_cost_change.event_type IS NULL OR
    IO_XPTO_in_cost_change.reason IS NULL OR
    IO_XPTO_in_cost_change.active_date IS NULL OR
    IO_XPTO_in_cost_change.status IS NULL OR
    IO_XPTO_in_cost_change.event_type = LP_CANCEL AND
    IO_XPTO_in_cost_change.status <> 'C' OR
    IO_XPTO_in_cost_change.colaborator_id IS NULL
  THEN
    --
    O_status_code := 1;
    O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'CC_BAD_DATA',
                                              I_txt_1 => NULL,
                                              I_txt_2 => L_program,
                                              I_txt_3 => NULL);
    --
    RETURN TRUE;
    --
  END IF;
  --
  -- Validate Specific Location
  --
  FOR i IN 1 .. IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL.count
    LOOP
      --
      IF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).location IS NOT NULL
        THEN
          --
          L_count_loc_not_null := L_count_loc_not_null +1;
      ELSE
        --
          L_count_loc_null := L_count_loc_null +1;
        --
      END IF;
       --
  END LOOP;
  --
  IF L_count_loc_not_null > 0 AND L_count_loc_null > 0
    THEN
      --
       O_status_code := 1;
       O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'CC_DET_LOC_NULL',
                                                 I_txt_1 => NULL,
                                                 I_txt_2 => L_program,
                                                 I_txt_3 => NULL);
                                                 --
      RETURN FALSE;
```

Figura 64- Função Validate_Enrich_CC *API Cost Change* (Validações Base parte 1)

```
      --
    END IF;
    --
    -- Validate detail mandatory fields
    --
  FOR i IN 1 .. IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL.count LOOP
      --
    IF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).supplier IS NULL OR
       IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).item IS NULL OR
       IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).loc_type IS NULL OR
       IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(i).COST_CHANGE_UNIT_COST IS NULL THEN
       --
       O_status_code := 1;
       O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'CC_BAD_DATA',
                                                 I_txt_1 => NULL,
                                                 I_txt_2 => L_program,
                                                 I_txt_3 => NULL);
                                                 --
       RETURN TRUE;
    --
    END IF;
  --
  END LOOP;
  --
  -- Validate origin country id
  --
  IF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).origin_country_id IS NOT NULL
   THEN
    --
   OPEN C_validate_origin_country;
   FETCH C_validate_origin_country
     INTO L_valid_origin_country;
   CLOSE C_validate_origin_country;
   --
   IF L_valid_origin_country IS NULL
     THEN
   O_status_code := 1;
   O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'COUNTRY_ID_NOT_MATCH',
                                             I_txt_1 => NULL,
                                             I_txt_2 => L_program,
                                             I_txt_3 => NULL);
     --
    RETURN TRUE;
    --
   END IF;
   --
  ELSIF IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).origin_country_id IS NULL
    THEN
     --
    OPEN C_check_origin_country;
     FETCH C_check_origin_country
      INTO L_check_origin_country;
    CLOSE C_check_origin_country;
     --
     IF L_check_origin_country IS NOT NULL
       THEN
         IO_XPTO_in_cost_change.IN_COST_CHANGES_DTL_TBL(1).origin_country_id := L_check_origin_country;
     ELSE
         O_status_code := 1;
         O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'COUNTRY_ID_NOT_FOUND',
                                                   I_txt_1 => NULL,
                                                   I_txt_2 => L_program,
                                                   I_txt_3 => NULL);
       --
       RETURN TRUE;
       --
     END IF;
    --
  END IF;
  --
RETURN TRUE;
  --
EXCEPTION
  WHEN OTHERS THEN
    --
    O_status_code := 255;
    O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                              I_txt_1 => SQLERRM,
                                              I_txt_2 => L_program,
                                              I_txt_3 => TO_CHAR(SQLCODE));
    --
    RETURN FALSE;
    --
END VALIDATE_ENRICH_CC;
```

Figura 65- Função Validate_Enrich_CC *API Cost Change* (Validações Base parte 2)

```
--Procedure Name : UPLD_COSTCHANGE
--Purpose        : Support upload operations of Cost Changes into the RMS
--               XPTO_in_cost_change - Table type based on XPTO_in_cost_change_obj
-------------------------------------------------------------------------------
  PROCEDURE UPLD_COSTCHANGE(IO_XPTO_in_cost_change IN OUT XPTO_IN_COST_CHANGES_TBL,
                            I_sync                 IN BOOLEAN,
                            O_status_code          OUT NUMBER,
                            O_error_desc           OUT VARCHAR2) AS
  --
  L_program          VARCHAR2(100) := L_program_name || '.UPLD_COSTCHANGE';
  L_O_status_code    NUMBER;
  L_O_error_desc     VARCHAR2(250);
  L_O_seq_process_id NUMBER;
  L_O_cost_change_id NUMBER;
  L_sync             BOOLEAN;
  L_O_error_msg      VARCHAR2(250);
  L_rms_async_id     NUMBER        := rms_async_id_seq.nextval;
  --
  -- EXCEPTION
  --
  PROGRAM_ERROR EXCEPTION;
  --
  /* CURSOR to get the final cost change that will be create when base process runs sucessfully */
  CURSOR C_get_cc_id(i_process_id NUMBER) IS
    SELECT cost_change
      FROM svc_cost_susp_sup_head s
     WHERE s.process_id = i_process_id;
  --
  /* CURSOR to get the error message from base process after trying create/update/cancel a cost change in core tables */
  CURSOR C_get_err_msg(i_process_id NUMBER)  IS
    SELECT cce.error_msg
      FROM coresvc_costchg_err cce
     WHERE cce.process_id = i_process_id;
  --
BEGIN
  --
  O_status_code  := 0; -- Initial Status to be returned is Success
  L_O_error_desc := NULL;
  L_sync := I_sync;
  --
  IF XPTO_API_COSTCHANGE.SYSTEM_VALIDATION_CC(IO_XPTO_in_cost_change,
                                              L_sync,
                                              L_O_status_code,
                                              L_O_error_desc) =
     FALSE THEN
    --
    IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                            i_api_procedure => 'UPLD_COSTCHANGE',
                                            i_entry_text    => 'An error occured while executing the global validations: ' ||
                                                               L_O_error_desc,
                                            i_entry_status  => O_status_code,
                                            i_entry_user_id => NULL,
                                            o_error_desc    => L_O_error_desc) =
       FALSE THEN
      --
      RAISE PROGRAM_ERROR;
      --
    END IF;
    --
  END IF;
  --
  -- Exit in case the Validation function returns a status other than 0
  --
  IF L_O_status_code != 0 then
    -- START AUDIT
     --
     IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                             i_api_procedure => 'UPLD_COSTCHANGE',
                                             i_entry_text    => 'Initial process validations failure: ' ||
                                                                L_O_error_desc,
                                             i_entry_status  => L_O_status_code,
                                             i_entry_user_id => NULL,
                                             o_error_desc    => L_O_error_desc) =
        FALSE THEN
       --
       RAISE PROGRAM_ERROR;
       --
     END IF;
     --
    -- END AUDIT
    O_status_code := L_O_status_code;
    O_error_desc  := L_O_error_desc;
    RETURN;
  --
  END IF;
  --
```

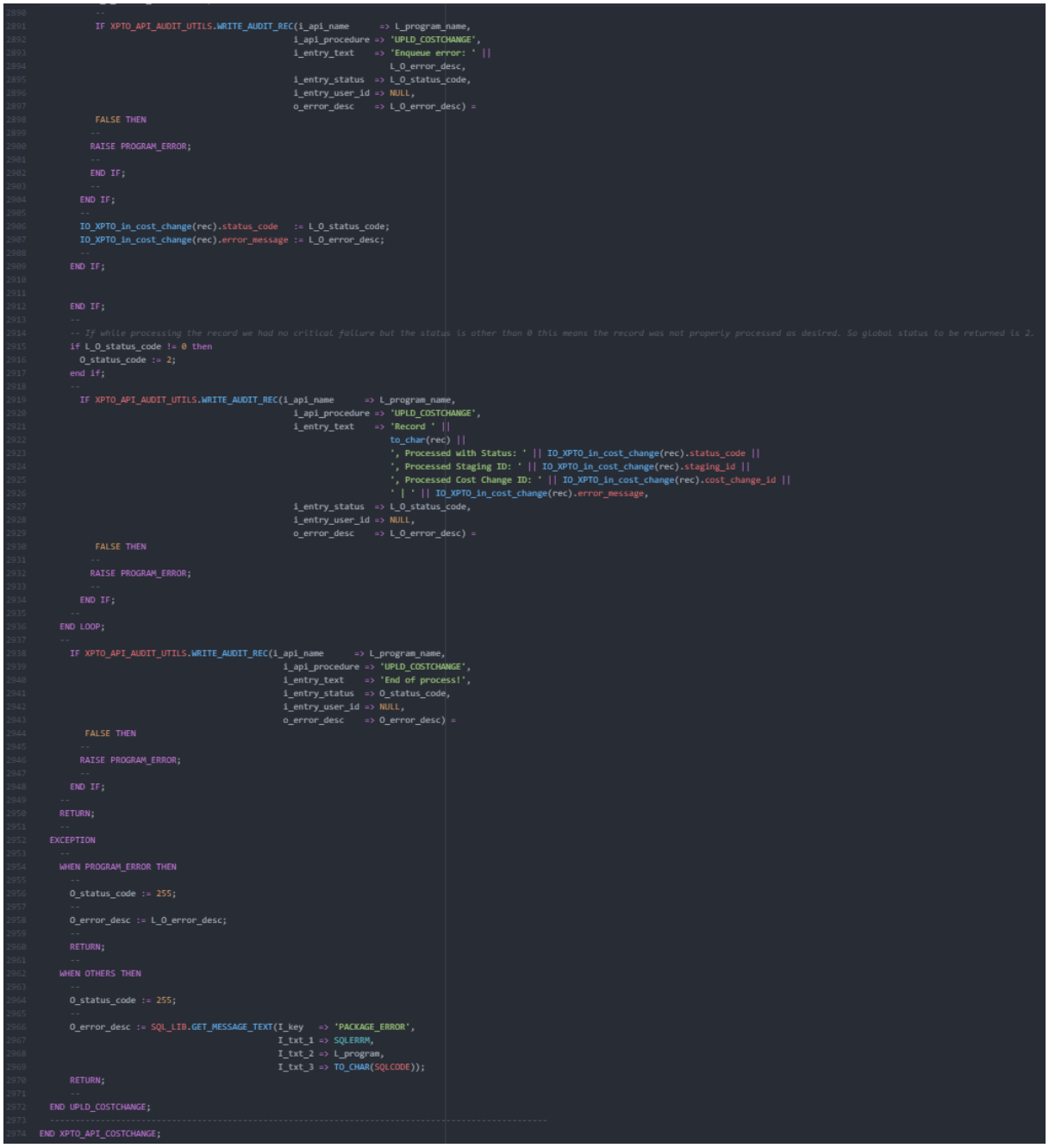
Figura 66- Procedimento UPLD_COSTCHANGE *API Cost Change* (parte 1)

```
    IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                            i_api_procedure => 'UPLD_COSTCHANGE',
                                            i_entry_text    => 'Process Started!',
                                            i_entry_status  => L_O_status_code,
                                            i_entry_user_id => NULL,
                                            o_error_desc    => L_O_error_desc) =
       FALSE THEN
      --
      RAISE PROGRAM_ERROR;
      --
    END IF;
    --
    -- For each cost change
    --
    FOR rec IN 1 .. IO_XPTO_in_cost_change.count LOOP
      --
      --Validate mandatory fields
      --
      IF VALIDATE_ENRICH_CC(IO_XPTO_in_cost_change(rec),
                            L_O_status_code,
                            L_O_error_desc) = FALSE THEN
        --
        IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                                i_api_procedure => 'UPLD_COSTCHANGE',
                                                i_entry_text    => 'Validations failure location: ' ||
                                                                   L_O_error_desc,
                                                i_entry_status  => L_O_status_code,
                                                i_entry_user_id => NULL,
                                                o_error_desc    => L_O_error_desc) =
           FALSE THEN
          --
          RAISE PROGRAM_ERROR;
          --
        END IF;
        --
      -- END AUDIT
      O_status_code := L_O_status_code;
      O_error_desc  := L_O_error_desc;
      RETURN;
        --
      END IF;
      --
      -- Exit in case the Validation function returns a status other than 0
      --
      IF L_O_status_code != 0 then
        -- START AUDIT
          --
        IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                                i_api_procedure => 'UPLD_COSTCHANGE',
                                                i_entry_text    => 'Validate Enrich failure: ' ||
                                                                   L_O_error_desc,
                                                i_entry_status  => L_O_status_code,
                                                i_entry_user_id => NULL,
                                                o_error_desc    => L_O_error_desc) =
             FALSE THEN
            --
            RAISE PROGRAM_ERROR;
            --
          END IF;
          --
        -- END AUDIT
        O_status_code := 2;
        IO_XPTO_in_cost_change(rec).status_code   := L_O_status_code;
        IO_XPTO_in_cost_change(rec).error_message := L_O_error_desc;
        --O_error_desc  := L_O_error_desc;
        --RETURN;
      --
      ELSE
      --
      --IF L_sync = TRUE THEN
      --
      IF XPTO_API_COSTCHANGE.WRITE_SVC_TABLES(IO_XPTO_in_cost_change(rec),
                                              L_O_seq_process_id,
                                              L_O_status_code,
                                              L_O_error_desc) = FALSE THEN
        --
        IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                                i_api_procedure => 'UPLD_COSTCHANGE',
                                                i_entry_text    => 'Write SVC abort: ' ||
                                                                   L_O_error_desc,
                                                i_entry_status  => L_O_status_code,
                                                i_entry_user_id => NULL,
                                                o_error_desc    => L_O_error_desc) =
           FALSE THEN
          --
          RAISE PROGRAM_ERROR;
          --
        END IF;
```

Figura 67- Procedimento UPLD_COSTCHANGE *API Cost Change* (parte 2)

```
2801          --
2802        -- END AUDIT
2803        O_status_code := L_O_status_code;
2804        O_error_desc  := L_O_error_desc;
2805        --
2806        RETURN;
2807          --
2808        END IF;
2809        --
2810        IO_XPTO_in_cost_change(rec).staging_id := L_O_seq_process_id;
2811        --
2812        -- Sync Mode
2813        --
2814        IF L_sync = TRUE THEN
2815          --
2816          L_O_error_msg := null;
2817          --
2818          IF CORESVC_COSTCHG.PROCESS(O_error_message => L_O_error_msg,
2819                                     I_process_id    => L_O_seq_process_id) =
2820            --
2821            FALSE THEN
2822            --
2823            L_O_status_code := 2;
2824            --
2825            OPEN C_get_err_msg(i_process_id => L_O_seq_process_id);
2826            FETCH C_get_err_msg
2827              INTO L_O_error_msg;
2828            CLOSE C_get_err_msg;
2829            --
2830            L_O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => L_O_error_msg,
2831                                                       I_txt_1 => NULL,
2832                                                       I_txt_2 => L_program,
2833                                                       I_txt_3 => NULL);
2834            --
2835            IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
2836                                                    i_api_procedure => 'UPLD_COSTCHANGE',
2837                                                    i_entry_text    => 'Validations failure process base: ' ||
2838                                                                       L_O_error_desc,
2839                                                    i_entry_status  => L_O_status_code,
2840                                                    i_entry_user_id => NULL,
2841                                                    o_error_desc    => L_O_error_desc) =
2842             FALSE THEN
2843            --
2844            RAISE PROGRAM_ERROR;
2845            --
2846            END IF;
2847            --
2848          END IF;
2849          --
2850          OPEN C_get_cc_id(i_process_id => L_O_seq_process_id);
2851          FETCH C_get_cc_id
2852            INTO L_O_cost_change_id;
2853          CLOSE C_get_cc_id;
2854          --
2855          IO_XPTO_in_cost_change(rec).cost_change_id := L_O_cost_change_id;
2856          IO_XPTO_in_cost_change(rec).status_code    := L_O_status_code;
2857          IO_XPTO_in_cost_change(rec).error_message  := L_O_error_desc;
2858          --
2859          -- Async Mode
2860          --
2861        ELSE
2862          --
2863          -- insert queue here!!
2864          --
2865          insert into svc_process_tracker(process_id,
2866                                          process_desc,
2867                                          template_key,
2868                                          action_type,
2869                                          process_source,
2870                                          process_destination,
2871                                          status,
2872                                          rms_async_id,
2873                                          action_date,
2874                                          user_id)
2875                                  values(IO_XPTO_in_cost_change(rec).staging_id,
2876                                          'Cost Change Desc',
2877                                          'COST_CHANGE',
2878                                          'U',
2879                                          'EXT', -- ou S9T
2880                                          'RMS',
2881                                          'N',
2882                                          L_rms_async_id,
2883                                          SYSDATE,
2884                                          user);
2885          --
2886          IF XPTO_RMS_ASYNC_PROCESS_SQL.ENQUEUE_ITEM_INDUCT(L_O_error_desc,
2887                                                            L_rms_async_id) = FALSE THEN
2888            --
2889            L_O_status_code := 255;
```

Figura 68- Procedimento UPLD_COSTCHANGE *API Cost Change* (parte 3)



```
            --
            IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                                    i_api_procedure => 'UPLD_COSTCHANGE',
                                                    i_entry_text    => 'Enqueue error: ' ||
                                                                       L_O_error_desc,
                                                    i_entry_status  => L_O_status_code,
                                                    i_entry_user_id => NULL,
                                                    o_error_desc    => L_O_error_desc) =
             FALSE THEN
            --
            RAISE PROGRAM_ERROR;
            --
            END IF;
            --
          END IF;
          --
          IO_XPTO_in_cost_change(rec).status_code   := L_O_status_code;
          IO_XPTO_in_cost_change(rec).error_message := L_O_error_desc;
          --
        END IF;


        END IF;
        --
        -- If while processing the record we had no critical failure but the status is other than 0 this means the record was not properly processed as desired. So global status to be returned is 2.
        if L_O_status_code != 0 then
          O_status_code := 2;
        end if;
        --
          IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                                  i_api_procedure => 'UPLD_COSTCHANGE',
                                                  i_entry_text    => 'Record ' ||
                                                                     to_char(rec) ||
                                                                     ', Processed with Status: ' || IO_XPTO_in_cost_change(rec).status_code ||
                                                                     ', Processed Staging ID: ' || IO_XPTO_in_cost_change(rec).staging_id ||
                                                                     ', Processed Cost Change ID: ' || IO_XPTO_in_cost_change(rec).cost_change_id ||
                                                                     ' | ' || IO_XPTO_in_cost_change(rec).error_message,
                                                  i_entry_status  => L_O_status_code,
                                                  i_entry_user_id => NULL,
                                                  o_error_desc    => L_O_error_desc) =
             FALSE THEN
            --
            RAISE PROGRAM_ERROR;
            --
          END IF;
          --
      END LOOP;
      --
        IF XPTO_API_AUDIT_UTILS.WRITE_AUDIT_REC(i_api_name      => L_program_name,
                                                i_api_procedure => 'UPLD_COSTCHANGE',
                                                i_entry_text    => 'End of process!',
                                                i_entry_status  => O_status_code,
                                                i_entry_user_id => NULL,
                                                o_error_desc    => O_error_desc) =
           FALSE THEN
          --
          RAISE PROGRAM_ERROR;
          --
        END IF;
      --
      RETURN;
      --
    EXCEPTION
      --
      WHEN PROGRAM_ERROR THEN
        --
        O_status_code := 255;
        --
        O_error_desc := L_O_error_desc;
        --
        RETURN;
        --
      WHEN OTHERS THEN
        --
        O_status_code := 255;
        --
        O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                                 I_txt_1 => SQLERRM,
                                                 I_txt_2 => L_program,
                                                 I_txt_3 => TO_CHAR(SQLCODE));
        RETURN;
        --
    END UPLD_COSTCHANGE;
    ------------------------------------------------------------------------------------------
END XPTO_API_COSTCHANGE;
```

Figura 69- Procedimento UPLD_COSTCHANGE *API Cost Change* (parte 4)

Teste Unitário

Cenário Positivo – Invocar o procedimento *UPLD_COSTCHANGE* tendo como *input*: uma *cost change* ao nível da *location type* com evento *Upsert* não tendo no sistema uma *cost change* com a mesma chave; uma cost change ao nível da *location type* com evento *Upsert* e tendo no sistema uma *cost change* com a mesma chave, sendo que para esta modificou-se o *cost value*; uma *cost change* ao nível da *location type* com evento *Cancel* e tendo no sistema uma *cost change* para a mesma chave.

Para executar este teste foi necessário criar uma *script* de teste para poder simular a entrada de dados vinda dos sistemas externos. A *script* usada foi a seguinte:

```
DECLARE
  lc_status_code NUMBER;
  lc_error_msg   VARCHAR2(1000);
  --
  lc_cost_tbl XPTO_IN_COST_CHANGES_TBL;
  lc_cost_obj XPTO_IN_COST_CHANGE_OBJ;
  --
  lc_cost2_tbl XPTO_IN_COST_CHANGE_DTL_TBL;
  lc_cost2_obj XPTO_IN_COST_CHANGE_DTL_OBJ;
BEGIN
  --
  --HEADER
  --
  lc_cost_tbl := XPTO_IN_COST_CHANGES_TBL();
  --
  --1st Cost Change for testing - header
  --
  lc_cost_obj                          := XPTO_IN_COST_CHANGE_OBJ;
  lc_cost_obj.event_type               := 'UPSERT';
  lc_cost_obj.cost_change              := null;--W--35077; --S--35048;
  lc_cost_obj.cost_change_description  := null;
  lc_cost_obj.reason                   := 14;
  lc_cost_obj.active_date              := to_date('2022-01-04', 'YYYY-MM-DD');
  lc_cost_obj.status                   := 'A';
  lc_cost_obj.cost_change_origin       := null;
  lc_cost_obj.collaborator_id           := 'CC_UT';
  lc_cost_obj.status_code              := null;
  lc_cost_obj.error_message            := null;
  --
  --DETAIL
  --
  lc_cost2_tbl := XPTO_IN_COST_CHANGE_DTL_TBL();
  --
  --1st Cost Change for testing - detail
  --
  lc_cost2_obj                    := XPTO_IN_COST_CHANGE_DTL_OBJ;
  lc_cost2_obj.supplier           := 2006200037;
  lc_cost2_obj.origin_country_id  := null;
  lc_cost2_obj.item               := 777011111;
  lc_cost2_obj.loc_type           := 'S';
  lc_cost2_obj.location           := null;
  lc_cost2_obj.cost_change_type   := 'F';
  lc_cost2_obj.cost_change_unit_cost   := 12;
  lc_cost2_obj.recalculate_order  := null;
  lc_cost2_obj.delivery_country_id := null;
  --
  --extends detail
  --
  lc_cost2_tbl.extend();
  lc_cost2_tbl(lc_cost2_tbl.count) := lc_cost2_obj;
  --
  --initialize table detail in header
  --
  lc_cost_obj.in_cost_change_dtl_tbl := lc_cost2_tbl;
  --
  --extends header
  --
  lc_cost_tbl.extend();
  lc_cost_tbl(lc_cost_tbl.last) := lc_cost_obj;
  --
```

Figura 70- *Script* teste para o Cenario 1, com 3 *cost changes* (parte 1)

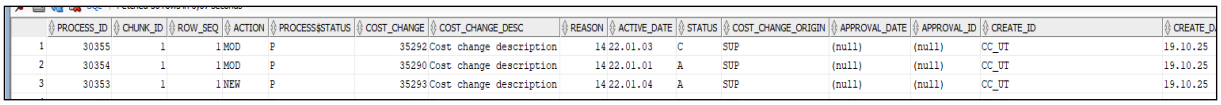
```
 --2st Cost Change for testing - header
--
lc_cost_obj                          := XPTO_IN_COST_CHANGE_OBJ;
lc_cost_obj.event_type               := 'UPSERT';
lc_cost_obj.cost_change              := null;--W--35077; --S--35048;
lc_cost_obj.cost_change_description  := null;
lc_cost_obj.reason                   := 14;
lc_cost_obj.active_date              := to_date('2022-01-01', 'YYYY-MM-DD');
lc_cost_obj.status                   := 'A';
lc_cost_obj.cost_change_origin       := null;
lc_cost_obj.collaborator_id          := 'CC_UT';
lc_cost_obj.status_code              := null;
lc_cost_obj.error_message            := null;
--
--DETAIL
--
lc_cost2_tbl := XPTO_IN_COST_CHANGE_DTL_TBL();
--
--2st Cost Change for testing - detail
--
lc_cost2_obj                    := XPTO_IN_COST_CHANGE_DTL_OBJ;
lc_cost2_obj.supplier           := 2006200037;
lc_cost2_obj.origin_country_id  := null;
lc_cost2_obj.item               := 777011111;
lc_cost2_obj.loc_type           := 'S';
lc_cost2_obj.location           := null;
lc_cost2_obj.cost_change_type   := 'F';
lc_cost2_obj.cost_change_unit_cost  := 22;
lc_cost2_obj.recalculate_order  := null;
lc_cost2_obj.delivery_country_id := null;
--
--extends detail
--
lc_cost2_tbl.extend();
lc_cost2_tbl(lc_cost2_tbl.count) := lc_cost2_obj;
--
--initialize table detail in header
--
lc_cost_obj.in_cost_change_dtl_tbl := lc_cost2_tbl;
--
--extends header
--
lc_cost_tbl.extend();
lc_cost_tbl(lc_cost_tbl.last) := lc_cost_obj;
```

Figura 71- *Script* teste para o Cenario 1, com 3 *cost changes* (parte 2)

```
    --3st Cost Change for testing - header
    --
    lc_cost_obj                          := XPTO_IN_COST_CHANGE_OBJ;
    lc_cost_obj.event_type               := 'CANCEL';
    lc_cost_obj.cost_change              := null;--W--35077; --S--35048;
    lc_cost_obj.cost_change_description  := null;
    lc_cost_obj.reason                   := 14;
    lc_cost_obj.active_date              := to_date('2022-01-03', 'YYYY-MM-DD');
    lc_cost_obj.status                   := 'C';
    lc_cost_obj.cost_change_origin       := null;
    lc_cost_obj.collaborator_id          := 'CC_UT';
    lc_cost_obj.status_code              := null;
    lc_cost_obj.error_message            := null;
    --
    --DETAIL
    --
    lc_cost2_tbl := XPTO_IN_COST_CHANGE_DTL_TBL();
    --
    --3st Cost Change for testing - detail
    --
    lc_cost2_obj                    := XPTO_IN_COST_CHANGE_DTL_OBJ;
    lc_cost2_obj.supplier           := 2006200037;
    lc_cost2_obj.origin_country_id  := null;
    lc_cost2_obj.item               := 777011111;
    lc_cost2_obj.loc_type           := 'S';
    lc_cost2_obj.location           := null;
    lc_cost2_obj.cost_change_type   := 'F';
    lc_cost2_obj.cost_change_unit_cost  := 11;
    lc_cost2_obj.recalculate_order  := null;
    lc_cost2_obj.delivery_country_id := null;
    --
    --extends detail
    --
    lc_cost2_tbl.extend();
    lc_cost2_tbl(lc_cost2_tbl.count) := lc_cost2_obj;
    --
    --initialize table detail in header
    --
    lc_cost_obj.in_cost_change_dtl_tbl := lc_cost2_tbl;
    --
    --extends header
    --
    lc_cost_tbl.extend();
    lc_cost_tbl(lc_cost_tbl.last) := lc_cost_obj;



    --
    XPTO_API_COSTCHANGE.UPLD_COSTCHANGE(lc_cost_tbl,
                                        true,
                                        lc_status_code,
                                        lc_error_msg);
    --
END;
```

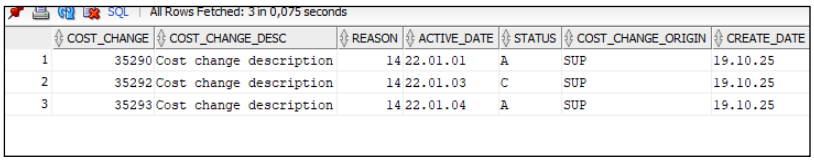
Figura 72- *Script* teste para o Cenario 1, com 3 *cost changes* (parte 3)

Após executar a *script* de teste na base de dados, podemos observar que os registos foram inseridos na tabela de *staging* e posteriormente foram carregados nas tabelas finais de *cost changes* no RMS, tal como poder-se-á observar nas tabelas abaixo.



PROCESS_ID	CHUNK_ID	ROW_SEQ	ACTION	PROCESS$STATUS	COST_CHANGE	COST_CHANGE_DESC	REASON	ACTIVE_DATE	STATUS	COST_CHANGE_ORIGIN	APPROVAL_DATE	APPROVAL_ID	CREATE_ID	CREATE_D
1 30355	1	1	MOD	P	35292	Cost change description	14	22.01.03	C	SUP	(null)	(null)	CC_UT	19.10.25
2 30354	1	1	MOD	P	35290	Cost change description	14	22.01.01	A	SUP	(null)	(null)	CC_UT	19.10.25
3 30353	1	1	NEW	P	35293	Cost change description	14	22.01.04	A	SUP	(null)	(null)	CC_UT	19.10.25

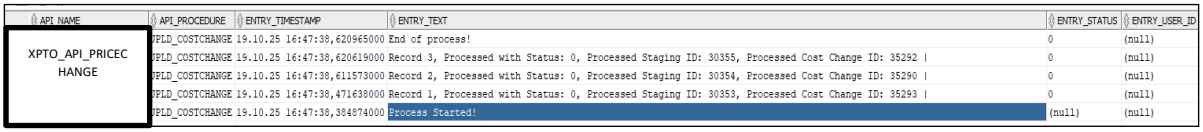
Tabela 17 - *Staging Cost Changes (Header)*



SQL | All Rows Fetched: 3 in 0,075 seconds

COST_CHANGE	COST_CHANGE_DESC	REASON	ACTIVE_DATE	STATUS	COST_CHANGE_ORIGIN	CREATE_DATE
1 35290	Cost change description	14	22.01.01	A	SUP	19.10.25
2 35292	Cost change description	14	22.01.03	C	SUP	19.10.25
3 35293	Cost change description	14	22.01.04	A	SUP	19.10.25

Tabela 18 - Tabela *final Cost Changes*



API_NAME	API_PROCEDURE	ENTRY_TIMESTAMP	ENTRY_TEXT	ENTRY_STATUS	ENTRY_USER_ID
XPTO_API_PRICECHANGE	UPLD_COSTCHANGE	19.10.25 16:47:38,620965000	End of process!	0	(null)
	UPLD_COSTCHANGE	19.10.25 16:47:38,620619000	Record 3, Processed with Status: 0, Processed Staging ID: 30355, Processed Cost Change ID: 35292 |	0	(null)
	UPLD_COSTCHANGE	19.10.25 16:47:38,611573000	Record 2, Processed with Status: 0, Processed Staging ID: 30354, Processed Cost Change ID: 35290 |	0	(null)
	UPLD_COSTCHANGE	19.10.25 16:47:38,471638000	Record 1, Processed with Status: 0, Processed Staging ID: 30353, Processed Cost Change ID: 35293 |	0	(null)
	UPLD_COSTCHANGE	19.10.25 16:47:38,384874000	Process Started!	(null)	(null)

Tabela 19 - XPTO_API_AUDIT

Como podemos ver na tabela XPTO_API_AUDIT, que faz um balanço de toda a execução, registo a registo, mostrando o respetivo estava e a mensagem de erro, quando aplicável, podemos ver que todos os registos foram inseridos com sucesso, retorno como estado "0".

Posto isto podemos concluir que o teste foi positivo!

Cenário 2 Positivo – Invocar o procedimento *UPLD_COSTCHANGE* tendo como *input*: Uma única *cost change* ao nível *location type* "B" (*Both*) com evento *Upsert* e não existe no sistema uma *cost change* para a mesma chave. Executar o procedimento em modo síncrono!

Para executar este teste foi necessário criar uma *script* de teste para poder simular a entrada de dados vinda dos sistemas externos. A *script* usada foi a seguinte:

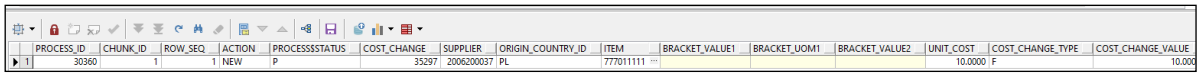
```
DECLARE
  lc_status_code NUMBER;
  lc_error_msg   VARCHAR2(1000);
  --
  lc_cost_tbl XPTO_IN_COST_CHANGES_TBL;
  lc_cost_obj XPTO_IN_COST_CHANGE_OBJ;
  --
  lc_cost2_tbl XPTO_IN_COST_CHANGE_DTL_TBL;
  lc_cost2_obj XPTO_IN_COST_CHANGE_DTL_OBJ;
BEGIN
  --
  --HEADER
  --
  lc_cost_tbl := XPTO_IN_COST_CHANGES_TBL();
  --
  --1st Cost Change for testing - header
  --
  lc_cost_obj                          := XPTO_IN_COST_CHANGE_OBJ;
  lc_cost_obj.event_type               := 'UPSERT';
  lc_cost_obj.cost_change              := null;--W--35077; --S--35048;
  lc_cost_obj.cost_change_description  := null;
  lc_cost_obj.reason                   := 14;
  lc_cost_obj.active_date              := to_date('2022-08-04', 'YYYY-MM-DD');
  lc_cost_obj.status                   := 'A';
  lc_cost_obj.cost_change_origin       := null;
  lc_cost_obj.collaborator_id          := 'CC_UT';
  lc_cost_obj.status_code              := null;
  lc_cost_obj.error_message            := null;
  --
  --DETAIL
  --
  lc_cost2_tbl := XPTO_IN_COST_CHANGE_DTL_TBL();
  --
  --1st Cost Change for testing - detail
  --
  lc_cost2_obj                    := XPTO_IN_COST_CHANGE_DTL_OBJ;
  lc_cost2_obj.supplier           := 2006200037;
  lc_cost2_obj.origin_country_id  := null;
  lc_cost2_obj.item               := 777011111;
  lc_cost2_obj.loc_type           := 'S';
  lc_cost2_obj.location           := null;
  lc_cost2_obj.cost_change_type   := 'F';
  lc_cost2_obj.cost_change_unit_cost  := 10;
  lc_cost2_obj.recalculate_order  := null;
  lc_cost2_obj.delivery_country_id := null;
  --
  --extends detail
  --
  lc_cost2_tbl.extend();
  lc_cost2_tbl(lc_cost2_tbl.count) := lc_cost2_obj;
  --
  --initialize table detail in header
  --
  lc_cost_obj.in_cost_change_dtl_tbl := lc_cost2_tbl;
  --
  --extends header
  --
  lc_cost_tbl.extend();
  lc_cost_tbl(lc_cost_tbl.last) := lc_cost_obj;
  --
  --
  --
  XPTO_API_COSTCHANGE.UPLD_COSTCHANGE(lc_cost_tbl,
                                      true,
                                      lc_status_code,
                                      lc_error_msg);
  --
END;
```

Figura 73- *Script* teste para o Cenario 2, com 1 *cost change*

Após executar a script de teste na base de dados, podemos observar que os registos foram inseridos na tabela de *staging* e posteriormente foram carregados nas tabelas finais de *cost changes* no RMS, tal como poder-se-á observar nas tabelas abaixo.

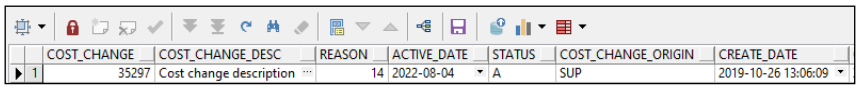
	PROCESS_ID	CHUNK_ID	ROW_SEQ	ACTION	PROCESSSSTATUS	COST_CHANGE	COST_CHANGE_DESC	REASON	ACTIVE_DATE	STATUS	COST_CHANGE_ORIGIN	APPROVAL_DATE	APPROVAL_ID
1	30360	1	1	NEW	P	35297	Cost change description	14	2022-08-04	A	SUP		

Tabela 20 - *Staging Cost Changes (header)*



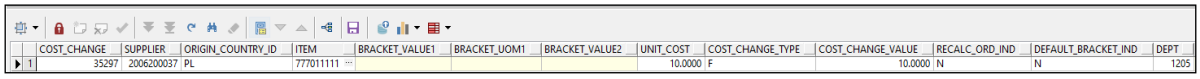
PROCESS_ID	CHUNK_ID	ROW_SEQ	ACTION	PROCESSSSTATUS	COST_CHANGE	SUPPLIER	ORIGIN_COUNTRY_ID	ITEM	BRACKET_VALUE1	BRACKET_UOM1	BRACKET_VALUE2	UNIT_COST	COST_CHANGE_TYPE	COST_CHANGE_VALUE
1 30360	1	1	NEW	P	35297	2006200037	PL	777011111				10.0000	F	10.000

Tabela 21 - *Staging Cost Changes (Detail)*



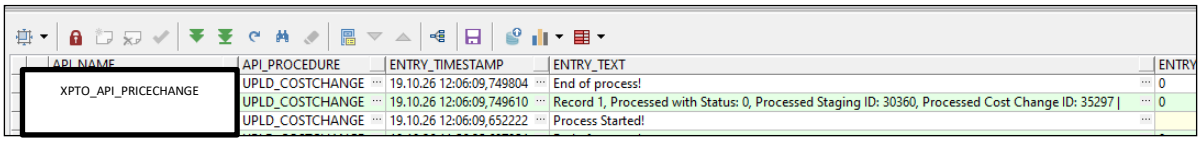
COST_CHANGE	COST_CHANGE_DESC	REASON	ACTIVE_DATE	STATUS	COST_CHANGE_ORIGIN	CREATE_DATE
1 35297	Cost change description	14	2022-08-04	A	SUP	2019-10-26 13:06:09

Tabela 22 - *Final Cost Change (Header)*



COST_CHANGE	SUPPLIER	ORIGIN_COUNTRY_ID	ITEM	BRACKET_VALUE1	BRACKET_UOM1	BRACKET_VALUE2	UNIT_COST	COST_CHANGE_TYPE	COST_CHANGE_VALUE	RECALC_ORD_IND	DEFAULT_BRACKET_IND	DEPT
1 35297	2006200037	PL	777011111				10.0000	F	10.0000	N	N	1205

Tabela 23 - *Final Cost Change (Detail)*



API_NAME	API_PROCEDURE	ENTRY_TIMESTAMP	ENTRY_TEXT	ENTRY
XPTO_API_PRICECHANGE	UPLD_COSTCHANGE	19.10.26 12:06:09,749804	End of process!	0
	UPLD_COSTCHANGE	19.10.26 12:06:09,749610	Record 1, Processed with Status: 0, Processed Staging ID: 30360, Processed Cost Change ID: 35297	0
	UPLD_COSTCHANGE	19.10.26 12:06:09,652222	Process Started!	

Tabela 24 - XPTO_API_AUDIT UPLD_COSTCHANGE

Este procedimento foi executado em modo síncrono, logo era expectável termos os registos inseridos na *staging* e posteriormente processados com sucesso para as tabelas finais de *cost change* do RMS. Tal como podemos observar nas tabelas, os registos foram inseridos não só na *staging* como também nas tabelas finais de *cost changes* do RMS. Como se pode observar na tabela XPTO_API_AUDIT, o registo foi processado com sucesso retornando estado 0 para a execução.

Posto isto podemos concluir que o teste foi positivo!

Cenário 3 Positivo – Invocar o procedimento *UPLD_COSTCHANGE* tendo como *input*: Uma única *cost change* ao nível *location type* "S" (*Store*) com evento *Cancel* e existe no sistema uma *cost change* para a mesma chave. Executar o procedimento em modo assíncrono!

Para executar este teste foi necessário criar uma *script* de teste para poder simular a entrada de dados vinda dos sistemas externos. A *script* usada foi a seguinte:

```
DECLARE
  lc_status_code NUMBER;
  lc_error_msg   VARCHAR2(1000);
  --
  lc_cost_tbl XPTO_IN_COST_CHANGES_TBL;
  lc_cost_obj XPTO_IN_COST_CHANGE_OBJ;
  --
  lc_cost2_tbl XPTO_IN_COST_CHANGE_DTL_TBL;
  lc_cost2_obj XPTO_IN_COST_CHANGE_DTL_OBJ;
BEGIN
  --
  --HEADER
  --
  lc_cost_tbl := XPTO_IN_COST_CHANGES_TBL();
  --
  --1st Cost Change for testing - header
  --
  lc_cost_obj                          := XPTO_IN_COST_CHANGE_OBJ;
lc_cost_obj.event_type               := 'CANCEL';
lc_cost_obj.cost_change              := 75061;--W--35077; --S--35048;
lc_cost_obj.cost_change_description  := null;
lc_cost_obj.reason                   := 14;
lc_cost_obj.active_date              := to_date('2023-09-10', 'YYYY-MM-DD');
lc_cost_obj.status                   := 'C';
lc_cost_obj.cost_change_origin       := null;
lc_cost_obj.collaborator_id          := 'CC_UT';
lc_cost_obj.status_code              := null;
lc_cost_obj.error_message            := null;
  --
  --DETAIL
  --
lc_cost2_tbl := XPTO_IN_COST_CHANGE_DTL_TBL();
  --
  --1st Cost Change for testing - detail
  --
lc_cost2_obj                    := XPTO_IN_COST_CHANGE_DTL_OBJ;
lc_cost2_obj.supplier           := 2006200037;
lc_cost2_obj.origin_country_id  := null;
lc_cost2_obj.item               := 777011111;
lc_cost2_obj.loc_type           := 'S';
lc_cost2_obj.location           := null;
lc_cost2_obj.cost_change_type   := 'F';
lc_cost2_obj.cost_change_unit_cost  := 14;
lc_cost2_obj.recalculate_order  := null;
lc_cost2_obj.delivery_country_id := null;
  --
  --extends detail
  --
  lc_cost2_tbl.extend();
  lc_cost2_tbl(lc_cost2_tbl.count) := lc_cost2_obj;
  --
  --initialize table detail in header
  --
  lc_cost_obj.in_cost_change_dtl_tbl := lc_cost2_tbl;
  --
  --extends header
  --
  lc_cost_tbl.extend();
  lc_cost_tbl(lc_cost_tbl.last) := lc_cost_obj;
  --
  --
  --
  XPTO_API_COSTCHANGE.UPLD_COSTCHANGE(lc_cost_tbl,
                                      false,
                                      lc_status_code,
                                      lc_error_msg);
  --
END;
```

Figura 74- *Script* teste para o Cenario 3, com 1 *cost change*

Antes de executar a script, a tabela final de *cost changes* do RMS contém os seguintes dados que serão usados para realizar este teste:

	COST_CHANGE	COST_CHANGE_DESC	REASON	ACTIVE_DATE	STATUS	COST_CHANGE_ORIGIN	CREATE_DATE
1	75061	Cost change description	14	2023-09-10	A	SUP	2019-11-13 15:47:29

Tabela 25 - Final de Cost Change

Após executar a *script* de teste na base de dados, podemos observar que os registos foram inseridos temporariamente numa *queue*, não na *staging*, e posteriormente foram carregados nas tabelas finais de *cost changes* no RMS, tal como poder-se-á observar nas tabelas abaixo.

	COST_CHANGE	COST_CHANGE_DESC	REASON	ACTIVE_DATE	STATUS	COST_CHANGE_ORIGIN	CREATE_DATE
1	75061	Cost change description	14	2023-09-10	C	SUP	2019-11-13 15:47:29

Tabela 26 - Staging Cost Changes (header)

	COST_CHANGE	SUPPLIER	ORIGIN_COUNTRY_ID	ITEM	LOC_TYPE	LOC	BRACKET_VALUE1	BRACKET_UOM1	BRACKET_VALUE2	UNIT_COST	COST_CHANGE_TYPE	COST_CHANGE_VALUE	RECALC_ORD_IND	DEFAULT_
1	75061	2006200037	PL	777011111	S	300610				14.0000	F	14.0000	N	N
2	75061	2006200037	PL	777011111	S	300611				14.0000	F	14.0000	N	N
3	75061	2006200037	PL	777011111	S	400610				14.0000	F	14.0000	N	N
4	75061	2006200037	PL	777011111	S	400611				14.0000	F	14.0000	N	N
5	75061	2006200037	PL	777011111	S	500610				14.0000	F	14.0000	N	N
6	75061	2006200037	PL	777011111	S	500611				14.0000	F	14.0000	N	N
7	75061	2006200037	PL	777011111	S	600610				14.0000	F	14.0000	N	N
8	75061	2006200037	PL	777011111	S	600611				14.0000	F	14.0000	N	N
9	75061	2006200037	PL	777011111	S	700610				14.0000	F	14.0000	N	N
10	75061	2006200037	PL	777011111	S	700611				14.0000	F	14.0000	N	N
11	75061	2006200037	PL	777011111	S	1600610				14.0000	F	14.0000	N	N
12	75061	2006200037	PL	777011111	S	1600611				14.0000	F	14.0000	N	N
13	75061	2006200037	PL	777011111	S	1700610				14.0000	F	14.0000	N	N

Tabela 27 - Staging Cost Changes (Detail)

Como podemos ver, para a *cost change* inicial em estado *Approve*, ela passou para o estado cancelada para todas as *stores* que dela faziam parte.

Como tal, podemos concluir que o teste foi positivo!

XXRMS418 *Item Attribute By Date* (PI3)

Package de Base de Dados

```
CREATE OR REPLACE PACKAGE XPTO_API_ITEM_CABD_IN_SQL AS

   ----------------------------------------------------------------------------
   /*****************************************************************************/
   /* CREATE DATE - JANUARY 2020                                                */
   /* CREATE USER - JOAO VILAVERDE                                              */
   /* PROJECT     - XPTO                                                        */
   /* DESCRIPTION - PACKAGE TO XXRMS418 - API Wrapper to support upload         */
   /*               operations of item attributes to be managed by date         */
   /*               into the RMS.                                               */
   /*****************************************************************************/
   --
   STATUS_SUCCESS          NUMBER(3) := 0;
   STATUS_ERROR            NUMBER(3) := 255;
   STATUS_PROC_ERROR       NUMBER(3) := 2;
   STATUS_N_FOUND          NUMBER(3) := 1;
   --
   ----------------------------------------------------------------------------
   --Procedure Name : UPLD_ITEM_BYDATE_TO_STG
   --Purpose        : Support upload operations of item attributes to be managed
   --                 by date into the RMS.
   --                 XPTO_IN_ITEM_ATTR_BYDATE_TBL - Table type based on XPTO_IN_ITEM_ATTR_BYDATE_OBJ
   ----------------------------------------------------------------------------
   PROCEDURE UPLD_ITEM_BYDATE_TO_STG(IO_ITEM_ATTRS_BYDATE IN OUT XPTO_ITEM_ATTR_BYDATE_TBL,
                                     O_STATUS_CODE         OUT    NUMBER,
                                     O_ERROR_DESC          OUT    VARCHAR2);
   ----------------------------------------------------------------------------------------------------
END XPTO_API_ITEM_CABD_IN_SQL;
/
```

Figura 75- XPTO_API_ITEM_CABD_IN_SQL.pks

```
FUNCTION UPSERT_IBD(IO_item_attrs_bydate IN OUT XPTO_ITEM_ATTR_BYDATE_OBJ,
                    I_delete             IN     BOOLEAN,
                    O_status_code        OUT    NUMBER,
                    O_error_desc         OUT    VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_PROCEDURE     VARCHAR2(30) := 'UPSERT_IBD';
  L_PROGRAM       VARCHAR2(70) := G_PROGRAM || '.' || L_PROCEDURE;
  L_record_exist  VARCHAR2(1);
  --
BEGIN
  --
  -- Check if the item attribute by date record already exists in XPTO_ITEM_ATTR_BYDATE table
  --
  OPEN C_record_exist(IO_item_attrs_bydate.item,
                      IO_item_attrs_bydate.effective_date,
                      IO_item_attrs_bydate.attribute_id,
                      IO_item_attrs_bydate.attribute_type);
  FETCH C_record_exist
    INTO L_record_exist;
  CLOSE C_record_exist;
  --
  -- If doesnt exist in XPTO_ITEM_ATTR_BYDATE table its a Create
  --
  IF L_record_exist IS NULL AND NOT I_delete THEN
    --
    IF NOT CREATE_IBD(IO_item_attrs_bydate,
                      TRUE,
                      O_status_code,
                      O_error_desc) THEN
      --
      RETURN TRUE;
      --
    END IF;
    --
  ELSIF L_record_exist IS NOT NULL AND I_delete THEN
    --
    IF NOT CREATE_IBD(IO_item_attrs_bydate,
                      FALSE,
                      O_status_code,
                      O_error_desc) THEN
      --
      RETURN TRUE;
      --
    END IF;
    --
  ELSE
    --
    -- If exists in XPTO_ITEM_ATTR_BYDATE table its an Update
    --
    IF NOT UPDATE_IBD(IO_item_attrs_bydate,
                      O_status_code,
                      O_error_desc) THEN
      --
      RETURN TRUE;
      --
    END IF;
    --
  END IF;
  --
  RETURN TRUE;
  --
EXCEPTION
  --
  WHEN OTHERS THEN
    --
    O_status_code := STATUS_ERROR;
    O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                              I_txt_1 => SQLERRM,
                                              I_txt_2 => L_program,
                                              I_txt_3 => TO_CHAR(SQLCODE));
    --
    RETURN FALSE;
  --
END UPSERT_IBD;
```

Figura 76- Função *Upsert API Item Attribute by Date*

```
677   FUNCTION DELETE_IBD(IO_item_attrs_bydate IN OUT XPTO_ITEM_ATTR_BYDATE_OBJ,
678                       O_status_code        OUT    NUMBER,
679                       O_error_desc         OUT    VARCHAR2)
680     RETURN BOOLEAN IS
681     --
682     L_PROCEDURE VARCHAR2(30) := 'DELETE_IBD';
683     L_PROGRAM   VARCHAR2(70) := G_PROGRAM || '.' || L_PROCEDURE;
684     L_record_exist      VARCHAR2(1);
685     L_O_error_desc      VARCHAR2(1000);
686     L_O_status_code     VARCHAR2(1);
687     L_get_vdate  PERIOD.VDATE%TYPE;
688     --
689   BEGIN
690     --
691     -- Check if the item attribute by date record already exists in XPTO_ITEM_ATTR_BYDATE table
692     --
693     OPEN C_record_exist(IO_item_attrs_bydate.item,
694                         IO_item_attrs_bydate.effective_date,
695                         IO_item_attrs_bydate.attribute_id,
696                         IO_item_attrs_bydate.attribute_type);
697     FETCH C_record_exist
698       INTO L_record_exist;
699     CLOSE C_record_exist;
700     --
701     IF L_record_exist IS NOT NULL THEN
702       --
703       -- Copy from staging into historic table
704       --
705       IF NOT UPSERT_IBD(IO_item_attrs_bydate,
706                         TRUE,
707                         L_O_status_code,
708                         L_O_error_desc) THEN
709         --
710         O_status_code    := L_O_status_code;
711         O_error_desc     := L_O_error_desc;
712         --
713         RETURN FALSE;
714         --
715       END IF;
716       --
717       -- Get vdate
718       --
719       L_get_vdate := get_vdate;
720       --
721       -- Delete record from staging
722       --
723       DELETE FROM XPTO_item_attr_bydate
724        WHERE item           = IO_item_attrs_bydate.item
725          AND attribute_type = IO_item_attrs_bydate.attribute_type
726          AND attribute_id   = IO_item_attrs_bydate.attribute_id
727          AND (effective_date = IO_item_attrs_bydate.effective_date
728           OR effective_date  > L_get_vdate)
729          AND status         = G_DEFAULT_STATUS; --exclude item attribute by date already process or in error
730       --
731     ELSE
732       --
733       O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'ET_REC_NOT_FOUND',
734                                                 I_txt_1 => SQLERRM,
735                                                 I_txt_2 => L_program,
736                                                 I_txt_3 => TO_CHAR(SQLCODE));
737       --
738       O_status_code := STATUS_N_FOUND;
739       --
740       RETURN TRUE;
741       --
742     END IF;
743     --
744     O_status_code := STATUS_SUCCESS;
745     --
746     RETURN TRUE;
747     --
748   EXCEPTION
749     --
750     WHEN OTHERS THEN
751       --
752       O_status_code := STATUS_ERROR;
753       O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
754                                                 I_txt_1 => SQLERRM,
755                                                 I_txt_2 => L_program,
756                                                 I_txt_3 => TO_CHAR(SQLCODE));
757       --
758       RETURN FALSE;
759       --
760     --
761   END DELETE_IBD;
```

Figura 77- Função *Delete API Item Attribute by Date*

```
424   FUNCTION CREATE_IBD(IO_item_attrs_bydate IN OUT XPTO_ITEM_ATTR_BYDATE_OBJ,
425                       I_staging_table      IN BOOLEAN,
426                       O_status_code        OUT    NUMBER,
427                       O_error_desc         OUT    VARCHAR2)
428     RETURN BOOLEAN IS
429     --
430     L_PROCEDURE      VARCHAR2(30) := 'CREATE_IBD';
431     L_PROGRAM        VARCHAR2(70) := G_PROGRAM || '.' || L_PROCEDURE;
432     L_check_item_edf VARCHAR2(1);
433     L_get_vdate      PERIOD.VDATE%TYPE;
434     --
435     -- Cursor to check all items that exist in Staging that have effective date > vdate
436     --
437     CURSOR C_check_item_edf (i_vdate DATE) IS
438       SELECT DISTINCT 'Y'
439         FROM XPTO_item_attr_bydate
440        WHERE item           = IO_item_attrs_bydate.item
441          AND attribute_id   = IO_item_attrs_bydate.attribute_id
442          AND attribute_type = IO_item_attrs_bydate.attribute_type
443          AND (effective_date = IO_item_attrs_bydate.effective_date
444           OR  effective_date  > i_vdate);
445     --
446   BEGIN
447     --
448     -- IF TRUE insert in the staging table
449     -- IF FALSE copy from staging into historic table
450     --
451     IF I_staging_table
452       THEN
453         --
454         -- Insert in the staging table
455         --
456        INSERT INTO XPTO_item_attr_bydate
457         (item,
458          attribute_type,
459          attribute_id,
460          effective_date,
461          value,
462          status,
463          process_date,
464          error_message,
465          create_datetime,
466          last_update_datetime,
467          last_update_id)
468       VALUES
469         (IO_item_attrs_bydate.item,
470          IO_item_attrs_bydate.attribute_type,
471          IO_item_attrs_bydate.attribute_id,
472          IO_item_attrs_bydate.effective_date,
473          IO_item_attrs_bydate.value,
474          G_DEFAULT_STATUS,
475          NULL,
476          IO_item_attrs_bydate.error_message,
477          sysdate,
478          sysdate,
479          USER);
480        --
481       ELSE
482         --
483         -- Get vdate
484         --
485         L_get_vdate := get_vdate;
486         --
487         -- Cursor to check if exist records that are in Staging table that have effective date > vdate.
488         --
489         OPEN C_check_item_edf(L_get_vdate);
490         FETCH C_check_item_edf
491           INTO L_check_item_edf;
492         CLOSE C_check_item_edf;
493
494         --
```

Figura 78- Função *Create API Item Attribute by Date* (parte 1)

```
        IF L_check_item_edf IS NOT NULL THEN
          --
          -- Copy from staging into historic table
          --
          INSERT INTO XPTO_item_attr_bydate_hist
           (item,
            attribute_type,
            attribute_id,
            effective_date,
            value,
            status,
            process_date,
            error_message,
            create_datetime,
            last_update_datetime,
            last_update_id)
           SELECT item,
                  attribute_type,
                  attribute_id,
                  effective_date,
                  value,
                  status,
                  process_date,
                  error_message,
                  create_datetime,
                  last_update_datetime,
                  last_update_id
             FROM XPTO_item_attr_bydate
            WHERE attribute_id    = IO_item_attrs_bydate.attribute_id
              AND item            = IO_item_attrs_bydate.item
              AND attribute_type  = IO_item_attrs_bydate.attribute_type
              AND (effective_date = IO_item_attrs_bydate.effective_date
               OR effective_date  > L_get_vdate);
          --
        ELSE
          --
          O_status_code := STATUS_N_FOUND;
          O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'XPTO_IBD_EFDT_LESS_VDT',
                                                    I_txt_1 => SQLERRM,
                                                    I_txt_2 => L_program,
                                                    I_txt_3 => TO_CHAR(SQLCODE));
          --
          RETURN TRUE;
          --
        END IF;
        --
      END IF;
      --
    O_status_code := STATUS_SUCCESS;
    --
    RETURN TRUE;
    --
  EXCEPTION
    --
    WHEN OTHERS THEN
      --
      O_status_code := STATUS_ERROR;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                                I_txt_1 => SQLERRM,
                                                I_txt_2 => L_program,
                                                I_txt_3 => TO_CHAR(SQLCODE));
      --
      RETURN FALSE;
    --
  END CREATE_IBD;
```

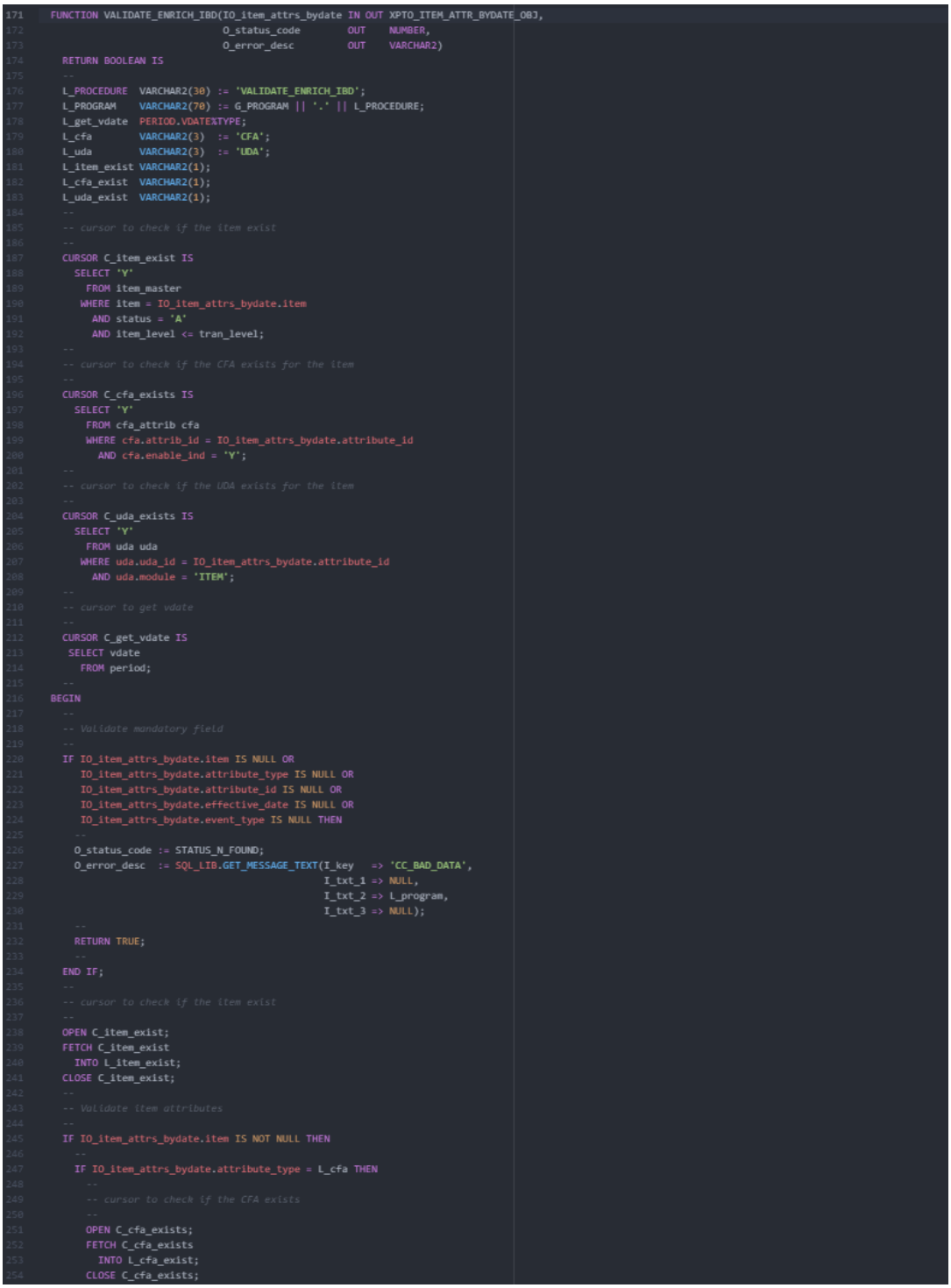
Figura 79- Função *Create API Item Attribute by Date* (parte 2)

```
561  FUNCTION UPDATE_IBD(IO_item_attrs_bydate IN OUT XPTO_ITEM_ATTR_BYDATE_OBJ,
562                      O_status_code        OUT    NUMBER,
563                      O_error_desc         OUT    VARCHAR2)
564    RETURN BOOLEAN IS
565    --
566    L_PROCEDURE VARCHAR2(30) := 'UPDATE_IBD';
567    L_PROGRAM   VARCHAR2(70) := G_PROGRAM || '.' || L_PROCEDURE;
568    --
569  BEGIN
570    --
571    -- Update the value of the item attribute by date
572    --
573    UPDATE XPTO_item_attr_bydate
574    SET value                = IO_item_attrs_bydate.value,
575        status               = G_DEFAULT_STATUS,
576        last_update_datetime = sysdate
577    WHERE item               = IO_item_attrs_bydate.item
578      AND effective_date     = IO_item_attrs_bydate.effective_date
579      AND attribute_id       = IO_item_attrs_bydate.attribute_id
580      AND attribute_type     = IO_item_attrs_bydate.attribute_type
581      AND status             NOT IN (G_DEFAULT_PROCESS);
582    --
583    O_status_code := STATUS_SUCCESS;
584    --
585    RETURN TRUE;
586    --
587  EXCEPTION
588    --
589    WHEN OTHERS THEN
590      --
591      O_status_code := STATUS_ERROR;
592      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
593                                                I_txt_1 => SQLERRM,
594                                                I_txt_2 => L_program,
595                                                I_txt_3 => TO_CHAR(SQLCODE));
596      --
597      RETURN FALSE;
598    --
599  END UPDATE_IBD;
```

Figura 80- Função *Update API Item Attribute by Date*

```
106  FUNCTION SYSTEM_VALIDATIONS_IBD(IO_item_attrs_bydate IN OUT XPTO_ITEM_ATTR_BYDATE_TBL,
107                                  O_status_code        OUT    NUMBER,
108                                  O_error_desc         OUT    VARCHAR2)
109    RETURN BOOLEAN IS
110    --
111    L_PROCEDURE VARCHAR2(30)     := 'SYSTEM_VALIDATIONS_IBD';
112    L_PROGRAM   VARCHAR2(70)     := G_PROGRAM || '.' || L_PROCEDURE;
113    L_max_sync_no_rec_upld_rec   XPTO_API_SETTINGS.MAX_SYNC_UPLD_REC%TYPE;
114    --
115    /* CURSOR to get the settings of the API */
116    CURSOR C_api_settings IS
117      SELECT max_sync_upld_rec
118        FROM XPTO_api_settings
119       WHERE api_name = 'XPTO_API_ITEM_CABD_IN_SQL';
120    --
121  BEGIN
122    --
123    OPEN C_api_settings;
124    FETCH C_api_settings
125      INTO L_max_sync_no_rec_upld_rec;
126    CLOSE C_api_settings;
127    --
128    IF L_max_sync_no_rec_upld_rec IS NULL THEN
129       --
130       O_status_code := STATUS_ERROR;
131       --
132       O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'MAX_SYNC_NO_REC_NULL',
133                                                I_txt_1 => NULL,
134                                                I_txt_2 => L_program,
135                                                I_txt_3 => NULL);
136       --
137       RETURN TRUE;
138       --
139     END IF;
140     --
141     IF L_max_sync_no_rec_upld_rec < IO_item_attrs_bydate.count THEN
142       --
143       O_status_code := STATUS_N_FOUND;
144       --
145       O_error_desc := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'XPTO_MAX_SYNC_LESS_IBD',
146                                                I_txt_1 => NULL,
147                                                I_txt_2 => L_program,
148                                                I_txt_3 => NULL);
149       --
150       RETURN TRUE;
151       --
152     END IF;
153    --
154    RETURN TRUE;
155    --
156  EXCEPTION
157    --
158    WHEN OTHERS THEN
159      --
160      O_status_code := STATUS_ERROR;
161      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
162                                                I_txt_1 => SQLERRM,
163                                                I_txt_2 => L_PROGRAM,
164                                                I_txt_3 => TO_CHAR(SQLCODE));
165      --
166      RETURN FALSE;
167      --
168    --
169  END SYSTEM_VALIDATIONS_IBD;
```

Figura 81- Função System_Validations_IBD *API Item Attribute by Date*



```
FUNCTION VALIDATE_ENRICH_IBD(IO_item_attrs_bydate IN OUT XPTO_ITEM_ATTR_BYDATE_OBJ,
                             O_status_code        OUT    NUMBER,
                             O_error_desc         OUT    VARCHAR2)
  RETURN BOOLEAN IS
  --
  L_PROCEDURE  VARCHAR2(30) := 'VALIDATE_ENRICH_IBD';
  L_PROGRAM    VARCHAR2(70) := G_PROGRAM || '.' || L_PROCEDURE;
  L_get_vdate  PERIOD.VDATE%TYPE;
  L_cfa        VARCHAR2(3)  := 'CFA';
  L_uda        VARCHAR2(3)  := 'UDA';
  L_item_exist VARCHAR2(1);
  L_cfa_exist  VARCHAR2(1);
  L_uda_exist  VARCHAR2(1);
  --
  -- cursor to check if the item exist
  --
  CURSOR C_item_exist IS
    SELECT 'Y'
      FROM item_master
     WHERE item = IO_item_attrs_bydate.item
       AND status = 'A'
       AND item_level <= tran_level;
  --
  -- cursor to check if the CFA exists for the item
  --
  CURSOR C_cfa_exists IS
    SELECT 'Y'
      FROM cfa_attrib cfa
      WHERE cfa.attrib_id = IO_item_attrs_bydate.attribute_id
        AND cfa.enable_ind = 'Y';
  --
  -- cursor to check if the UDA exists for the item
  --
  CURSOR C_uda_exists IS
    SELECT 'Y'
      FROM uda uda
     WHERE uda.uda_id = IO_item_attrs_bydate.attribute_id
       AND uda.module = 'ITEM';
  --
  -- cursor to get vdate
  --
  CURSOR C_get_vdate IS
   SELECT vdate
     FROM period;
  --
BEGIN
  --
  -- Validate mandatory field
  --
  IF IO_item_attrs_bydate.item IS NULL OR
     IO_item_attrs_bydate.attribute_type IS NULL OR
     IO_item_attrs_bydate.attribute_id IS NULL OR
     IO_item_attrs_bydate.effective_date IS NULL OR
     IO_item_attrs_bydate.event_type IS NULL THEN
    --
    O_status_code := STATUS_N_FOUND;
    O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'CC_BAD_DATA',
                                              I_txt_1 => NULL,
                                              I_txt_2 => L_program,
                                              I_txt_3 => NULL);
    --
    RETURN TRUE;
    --
  END IF;
  --
  -- cursor to check if the item exist
  --
  OPEN C_item_exist;
  FETCH C_item_exist
    INTO L_item_exist;
  CLOSE C_item_exist;
  --
  -- Validate item attributes
  --
  IF IO_item_attrs_bydate.item IS NOT NULL THEN
    --
    IF IO_item_attrs_bydate.attribute_type = L_cfa THEN
      --
      -- cursor to check if the CFA exists
      --
      OPEN C_cfa_exists;
      FETCH C_cfa_exists
        INTO L_cfa_exist;
      CLOSE C_cfa_exists;
```

Figura 82- Função Validate_Enrich_IBD *API Item Attribute by Date* (parte 1)

```
        IF L_cfa_exist IS NULL THEN

          O_status_code := STATUS_N_FOUND;
          O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'XPTO_CFA_NOT_EXIST',
                                                    I_txt_1 => NULL,
                                                    I_txt_2 => L_program,
                                                    I_txt_3 => NULL);

           RETURN TRUE;

        END IF;

      ELSIF IO_item_attrs_bydate.attribute_type = L_uda THEN

        -- cursor to check if the UDA exists

        OPEN C_uda_exists;
        FETCH C_uda_exists
          INTO L_uda_exist;
        CLOSE C_uda_exists;

        IF L_uda_exist IS NULL THEN

          O_status_code := STATUS_N_FOUND;
          O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'XPTO_UDA_NOT_EXIST',
                                                    I_txt_1 => NULL,
                                                    I_txt_2 => L_program,
                                                    I_txt_3 => NULL);

           RETURN TRUE;

        END IF;

      ELSE

        O_status_code := STATUS_N_FOUND;
        O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'XPTO_ATTR_TYPE_N_EXIST',
                                                  I_txt_1 => NULL,
                                                  I_txt_2 => L_program,
                                                  I_txt_3 => NULL);

        RETURN TRUE;

      END IF;

      -- cursor to check if the effective date is in the future

      OPEN C_get_vdate;
      FETCH C_get_vdate
        INTO L_get_vdate;
      CLOSE C_get_vdate;

      IF IO_item_attrs_bydate.effective_date < L_get_vdate THEN

        O_status_code := STATUS_N_FOUND;
        O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'DATE_NOT_IN_FUTURE',
                                                  I_txt_1 => NULL,
                                                  I_txt_2 => L_program,
                                                  I_txt_3 => NULL);

         RETURN TRUE;

      END IF;

    ELSE

      O_status_code := STATUS_N_FOUND;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'ET_ITEM_NOT_EXIST',
                                                I_txt_1 => NULL,
                                                I_txt_2 => L_program,
                                                I_txt_3 => NULL);

      RETURN TRUE;

    END IF;

    O_status_code := STATUS_SUCCESS;

    RETURN TRUE;

  EXCEPTION

    WHEN OTHERS THEN

      O_status_code := STATUS_ERROR;
      O_error_desc  := SQL_LIB.GET_MESSAGE_TEXT(I_key   => 'PACKAGE_ERROR',
                                                I_txt_1 => SQLERRM,
                                                I_txt_2 => L_PROGRAM,
                                                I_txt_3 => TO_CHAR(SQLCODE));

      RETURN FALSE;

  END VALIDATE_ENRICH_IBD;
```

Figura 83- Função Validate_Enrich_IBD *API Item Attribute by Date* (parte 2)

Teste Unitário

Cenário 1 Positivo – Invocar o procedimento *UPLD_ITEM_BYDATE_TO_STG* tendo como *input* dois registos com evento *Upsert* e *attribute_ID* "UDA" com attribute_ID que existe no RMS.

Para a realização deste cenário 1, foi necessária a criação de uma *script* de teste, sendo ela:

```
DECLARE
  --
  lc_status_code NUMBER;
  lc_error_msg   VARCHAR2(1000);
  --
  lc_itbd_tbl XPTO_ITEM_ATTR_BYDATE_TBL;
  lc_itbd_obj XPTO_ITEM_ATTR_BYDATE_OBJ;
  --
BEGIN
  --
  --HEADER
  --
  lc_itbd_tbl := XPTO_ITEM_ATTR_BYDATE_TBL();
  --
  --1st item attribute by date
  --
  lc_itbd_obj                := XPTO_ITEM_ATTR_BYDATE_OBJ();
  lc_itbd_obj.ITEM           := '777010702';
  lc_itbd_obj.ATTRIBUTE_TYPE := 'UDA';
  lc_itbd_obj.ATTRIBUTE_ID   := '255';
  lc_itbd_obj.EFFECTIVE_DATE := to_date('05/01/2020', 'DD/MM/YYYY');
  lc_itbd_obj.VALUE          := 'L';
  lc_itbd_obj.EVENT_TYPE     := 'UPSERT';
  lc_itbd_obj.STATUS         := null;
  lc_itbd_obj.ERROR_MESSAGE  := null;
  --
  lc_itbd_tbl.extend();
  lc_itbd_tbl(lc_itbd_tbl.last) := lc_itbd_obj;
  --
  --2st item attribute by date
  --
  lc_itbd_obj                := XPTO_ITEM_ATTR_BYDATE_OBJ();
  lc_itbd_obj.ITEM           := '777010702';
  lc_itbd_obj.ATTRIBUTE_TYPE := 'UDA';
  lc_itbd_obj.ATTRIBUTE_ID   := '255';
  lc_itbd_obj.EFFECTIVE_DATE := to_date('17/01/2020', 'DD/MM/YYYY');
  lc_itbd_obj.VALUE          := 'J';
  lc_itbd_obj.EVENT_TYPE     := 'UPSERT';
  lc_itbd_obj.STATUS         := null;
  lc_itbd_obj.ERROR_MESSAGE  := null;
  --
  lc_itbd_tbl.extend();
  lc_itbd_tbl(lc_itbd_tbl.last) := lc_itbd_obj;


  --
  XPTO_API_ITEM_CATTR_BYDTE.UPLD_ITEM_BYDATE_TO_STG(lc_itbd_tbl,
                                                    lc_status_code,
                                                    lc_error_msg);
  --
  dbms_output.put_line('lc_status_code: '   || lc_status_code);
  dbms_output.put_line('lc_error_message: ' || lc_error_msg);
  --
END;
```

Figura 84- *Script* Teste Cenário 1 *API Item Attribute by Date*

Após executar a script de teste, temos como resultado a inserção dos registos na *staging* com sucesso, tal como poder-se-á visualizar nas tabelas abaixo.

	ITEM	ATTRIBUTE_TYPE	ATTRIBUTE_ID	EFFECTIVE_DATE	VALUE	STATUS	PROCESS_DATE	ERROR_MESSAGE
1	777010702	UDA	255	05/01/2020	L	N		
2	777010702	UDA	255	17/01/2020	J	N		
3	777010702	UDA	1	10/10/0200	A	N		
4	777010702	UDA	1	12/10/0200	A	N		
5	777010703	UDA	1	12/10/0200	A	N		
6	777010703	UDA	1	10/10/0200	A	P		
7	777010703	UDA	1	16/10/0200	A	P		
8	777010702	UDA	1	15/10/0200	A	P		
9	777010702	UDA	255	08/01/2020	B	N		
10	777010702	UDA	255	12/01/2020	C	N		

Tabela 28 - *Staging Item Attribute by Date*

E	API_PROCEDURE	ENTRY_TIMESTAMP	ENTRY_TEXT	ENTRY_STATUS	ENTRY_USER_ID
API_ITEM_CABD_IN_SQL	UPLD_ITEM_BYDATE_TO_STG	09-JAN-20 03.47.24.212588 PM	End of process!	0	
API_ITEM_CABD_IN_SQL	UPLD_ITEM_BYDATE_TO_STG	09-JAN-20 03.47.24.212466 PM	Record 2, Processed with Status: 0 |	0	
API_ITEM_CABD_IN_SQL	UPLD_ITEM_BYDATE_TO_STG	09-JAN-20 03.47.24.212055 PM	Record 1, Processed with Status: 0 |	0	
API_ITEM_CABD_IN_SQL	UPLD_ITEM_BYDATE_TO_STG	09-JAN-20 03.47.24.210383 PM	Process Started!		

Tabela 29 - XPTO_API_AUDIT UPLD_ITEM_BYDATE_TO_STG (cenário 1)

Posto isto, poder-se-á concluir que o testo foi positivo.

Cenário 2 Positivo – Invocar o procedimento *UPLD_ITEM_BYDATE_TO_STG* tendo como *input* um registo com evento *Upsert* e attribute_ID "CFA" com attribute_ID que existe no RMS.

Para a realização deste cenário 2, foi necessária a criação de uma *script* de teste, sendo ela:

```
DECLARE
   --
   lc_status_code NUMBER;
   lc_error_msg   VARCHAR2(1000);
   --
   lc_itbd_tbl XPTO_ITEM_ATTR_BYDATE_TBL;
   lc_itbd_obj XPTO_ITEM_ATTR_BYDATE_OBJ;
   --
BEGIN
   --
   --HEADER
   --
   lc_itbd_tbl := XPTO_ITEM_ATTR_BYDATE_TBL();
   --
   --1st item attribute by date
   --
   lc_itbd_obj                 := XPTO_ITEM_ATTR_BYDATE_OBJ();
   lc_itbd_obj.ITEM            := '777010702';
   lc_itbd_obj.ATTRIBUTE_TYPE  := 'CFA';
   lc_itbd_obj.ATTRIBUTE_ID    := '814';
   lc_itbd_obj.EFFECTIVE_DATE  := to_date('07/01/2020', 'DD/MM/YYYY');
   lc_itbd_obj.VALUE           := 'A';
   lc_itbd_obj.EVENT_TYPE      := 'UPSERT';
   lc_itbd_obj.STATUS          := null;
   lc_itbd_obj.ERROR_MESSAGE   := null;
   --
   lc_itbd_tbl.extend();
   lc_itbd_tbl(lc_itbd_tbl.last) := lc_itbd_obj;
   --
   XPTO_API_ITEM_CATTR_BYDTE.UPLD_ITEM_BYDATE_TO_STG(lc_itbd_tbl,
                                                     lc_status_code,
                                                     lc_error_msg);
   --
   dbms_output.put_line('lc_status_code: '   || lc_status_code);
   dbms_output.put_line('lc_error_message: ' || lc_error_msg);
   --
END;
```

Figura 85- *Script* Teste Cenario 2 *API Item Attribute by Date*

Após executar a script de teste, temos como resultado a inserção dos registos na *staging* com sucesso, tal como poder-se-á visualizar nas tabelas abaixo.

	ITEM	ATTRIBUTE_TYPE	ATTRIBUTE_ID	EFFECTIVE_DATE	VALUE	STATUS	PROCESS_DATE	ERROR_MESSAGE
1	777010702	UDA	255	05/01/2020	L	N		
2	777010702	UDA	255	17/01/2020	J	N		
3	777010702	CFA	814	07/01/2020	A	N		
4	777010702	UDA	1	10/10/0200	A	N		
5	777010702	UDA	1	12/10/0200	A	N		
6	777010703	UDA	1	12/10/0200	A	N		
7	777010703	UDA	1	10/10/0200	A	P		
8	777010703	UDA	1	16/10/0200	A	P		
9	777010702	UDA	1	15/10/0200	A	P		
10	777010702	UDA	255	08/01/2020	B	N		
11	777010702	UDA	255	12/01/2020	C	N		

Tabela 30 - Staging Item Attribute by Date

E	API_PROCEDURE	ENTRY_TIMESTAMP	ENTRY_TEXT	ENTRY_STATUS	ENTRY_USER_ID
API_ITEM_CABD_IN_SQL	UPLD_ITEM_BYDATE_TO_STG	09-JAN-20 04.04.32.489232 PM	End of process!	0	
API_ITEM_CABD_IN_SQL	UPLD_ITEM_BYDATE_TO_STG	09-JAN-20 04.04.32.489093 PM	Record 1, Processed with Status: 0 |	0	
API_ITEM_CABD_IN_SQL	UPLD_ITEM_BYDATE_TO_STG	09-JAN-20 04.04.32.454382 PM	Process Started!		

Tabela 31 - XPTO_API_AUDIT UPLD_ITEM_BYDATE_TO_STG (cenário 2)

Posto isto, poder-se-á concluir que o testo foi positivo.