Mestrado em Engenharia Informática

Dissertação/Estágio

Relatório Final

Real-Time Warehousing

Nickerson Fonseca Ferreira

nickerson@student.dei.uc.pt

Orientador:

Pedro Nuno San-Bento Furtado

Data: 03 de Julho de 2013

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

Com o crescimento acelerado da quantidade de informação gerada dentro das organizações, as bases de dados continuaram seu processo evolutivo, surgindo novas práticas para manipular, de forma viável, uma quantidade cada vez maior de dados. Um exemplo deste tipo de prática é chamado *Data Warehouse* (DW). Porém, este tipo de sistema possui uma limitação relacionada ao tempo de atualização dos respectivos dados. A cada dia que passa a demanda por dados atualizados em tempo real vem se tornando uma realidade no contexto de DW. O objetivo deste trabalho é a criação de uma arquitetura que permita aos utilizadores do DW acessar não somente os dados históricos, mas também dados recentes, caracterizando um contexto de tempo real. Esta solução é denominada *Real-Time Data Warehouse* (RTDW). Além de sugerir esta nova arquitetura, este projeto irá mensurar a viabilidade da abordagem supracitada através de testes realizados numa plataforma desenvolvida para este fim.

Palavras-Chave

Informação, Base de dados, *Data Warehouse*, *Real-Time Data Warehouse*

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

BD	Base de dados
BI	*Business Inteligence*
CDC	*Change Data Capture*
CSV	*Comma-Separated Values*
D-DW	*Dynamic Data Warehouse*
DP	*Data Package*
DPA	*Data Processing Area*
DS	*Data Source*
DW	*Data Warehouse*
EJB	*Enterprise Java Beans*
ELT	*Extract-Load-Transform*
ETL	*Extract-Transform-Load*
ETLR	*Extract-Transform-Load-Refresh*
IINM	*Independent Instance No Migration*
IIWM	*Independent Instance With Migration*
ISB	*Input Stream Buffer*
IT	*Information Technology*
JDBC	*Java Database Connectivity*
LP	*Landing Pad*
MV	*Materialized View*
ODI	*Oracle Data Integrator*
OLAP	*Online Analytical Processing*
QW	*Query Workload*
RTDW	*Real-Time Data Warehouse*
S-DW	*Static Data Warehouse*
SF	*Scale Factor*
SOA	*Service-Oriented Architecture*
SQL	*Structured Query Language*

SSB	*Star Schema Benchmark*
SSB-RT	*Star Schema Benchmark Real-time*
TPC	*Transaction Processing Performance Council*
XML	*Extensible Markup Language*

Capítulo 1
Introdução

Com o aumento da competitividade entre as empresas, é crescente a demanda por soluções que auxiliem seus administradores na tomada de decisões corretas e no menor espaço de tempo. Essas soluções utilizam um grande volume de dados históricos armazenados e carecem de uma constante atualização, no menor período de tempo. É onde se encontra o problema. Essas bases de dados que são utilizadas para sistemas de apoio à decisão são conhecidas como *Data Warehouse* (DW). Como esses DW possuem uma enorme quantidade de dados, sua atualização se torna bastante demorada, impossibilitando que esse processo seja realizado constantemente.

Atualmente, um conceito relativamente novo chama a atenção da comunidade de pesquisadores, conhecido como *Real-Time Data Warehouse* (RTDW). Esta nova solução é composta por componentes que permitem que os utilizadores possuam acesso aos dados históricos, além dos dados mais recentes, por exemplo, de minutos atrás.

Segundo relatórios publicados em (GARTNER, 2012), a receita gerada por sistemas de *Business Inteligence* (BI), sistemas de análise de dados e gerenciamento de performance foi de U$ 12, 2 bilhões no ano de 2011. O lucro obtido em sistemas de BI aumentou cerca de 16% em 2011, ou seja, várias empresas do mercado de IT (*Information Tecnology*) estão investindo bastante em novas ideias e tecnologias. Esse mercado em constante crescimento é uma das principais motivações para a realização de pesquisas e desenvolvimento de soluções que atendam a respectiva demanda. Empresas como bancos ou administradoras de cartão de crédito, por exemplo, estão utilizando soluções deste tipo para detecção de fraudes, já que a necessidade de possuir os dados incessantemente atualizados é de extrema importância.

1.1. Definições Preliminares

O propósito deste tópico é dar ao leitor uma breve definição de determinados termos técnicos, com a finalidade de que este, mesmo que não tenha domínio do assunto, saiba do que se trata.

Um *Data Warehouse* (DW) é uma base de dados com registros históricos, possuindo uma massa de dados muito grande. As bases de dados são utilizadas para suporte de apoio à decisão e também são conhecidas como base de dados OLAP (*Online Analytical Processing*).

O termo *Real-Time Data Warehouse* (RTDW) é usado para representar os DW que possuem seus dados atualizados constantemente, já que os DW tradicionais não suportam tal atividade.

1.2. Objetivos

1.2.1. Objetivos Gerais

O objetivo deste trabalho é o desenvolvimento de uma solução que seja capaz de oferecer ao utilizador a informação mais atualizada numa base de dados tipo *Data Warehouse.* Além disso, o custo da adição periódica dos dados mais recentes esteja dentro de um nível aceitável, possibilitando aos utilizadores acesso às informações de forma rápida e simples.

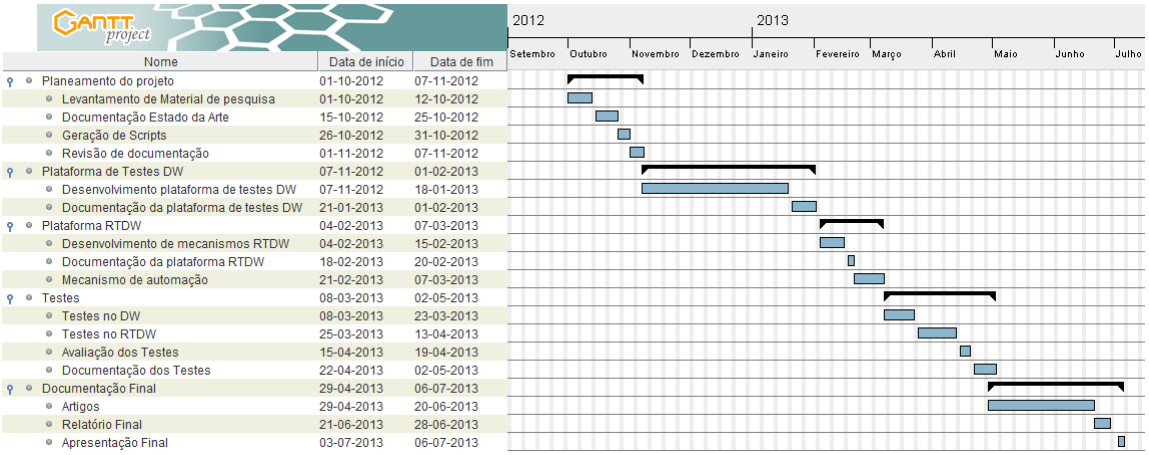
1.2.2. Objetivos Específicos

- Definir um *benchmark* para RTDW.
- Desenvolvimento de uma plataforma de testes, implementando o *benchmark* definido, para avaliar as falhas da arquitetura tradicional de um *Data Warehouse* (DW) num contexto de *Real-Time Data Warehouse* (RTDW);
- Elaboração de uma proposta de arquitetura que se adeque à situação de RTDW;

- Testes sobre a arquitetura tradicional de um DW e na arquitetura proposta de RTDW;
- Avaliação dos resultados obtidos nos testes realizados.

1.3. Cronograma

Com o propósito de organizar as fases do projeto, listar e definir os prazos dos *milestones* do mesmo, um cronograma foi elaborado e apresentado abaixo na Figura 1.



Nome	Data de início	Data de fim
Planeamento do projeto	01-10-2012	07-11-2012
Levantamento de Material de pesquisa	01-10-2012	12-10-2012
Documentação Estado da Arte	15-10-2012	25-10-2012
Geração de Scripts	26-10-2012	31-10-2012
Revisão de documentação	01-11-2012	07-11-2012
Plataforma de Testes DW	07-11-2012	01-02-2013
Desenvolvimento plataforma de testes DW	07-11-2012	18-01-2013
Documentação da plataforma de testes DW	21-01-2013	01-02-2013
Plataforma RTDW	04-02-2013	07-03-2013
Desenvolvimento de mecanismos RTDW	04-02-2013	15-02-2013
Documentação da plataforma RTDW	18-02-2013	20-02-2013
Mecanismo de automação	21-02-2013	07-03-2013
Testes	08-03-2013	02-05-2013
Testes no DW	08-03-2013	23-03-2013
Testes no RTDW	25-03-2013	13-04-2013
Avaliação dos Testes	15-04-2013	19-04-2013
Documentação dos Testes	22-04-2013	02-05-2013
Documentação Final	29-04-2013	06-07-2013
Artigos	29-04-2013	20-06-2013
Relatório Final	21-06-2013	28-06-2013
Apresentação Final	03-07-2013	06-07-2013

Figura 1: **Gráfico de Gantt com o cronograma do projeto.**

Na imagem acima, as tarefas de planejamento, definição de arquitetura e os levantamentos de requisitos foram realizados no início do primeiro semestre. Após esta fase, ainda foi possível iniciar o desenvolvimento da plataforma de testes e documentação.

As atividades planejadas para o segundo semestre tiveram o foco na arquitetura RTDW (desenvolvimento e documentação), nos testes que são necessários para avaliação da solução proposta e no relatório e apresentação final deste projeto.

1.4. Publicações

Este trabalho teve caráter essencialmente científico. No âmbito deste trabalho foram produzidos alguns artigos científicos, os quais foram submetidos para conferências e periódicos. Os artigos produzidos e submetidos foram:

Conferência

Nickerson Ferreira, Pedro Furtado, "Near Real-Time with traditional Data Warehouse Architectures: Factors and how-to", submetido para IDEAS 2013.

Nickerson Ferreira, Pedro Furtado, "Transforming a Normal Warehouse for Realtime 24/7 Operation ", submetido para DOLAP 2013.

Periódicos

Nickerson Ferreira, Pedro Furtado, "A Real-Time Data Warehouse Architecture: Proposal and Evaluation", submetido para Information Systems.

Nickerson Ferreira, Pedro Furtado, "A Survey on Real-Time Data Warehouse", finalizado e aguardando ser submetido para Data & Knowledge Engineering.

Nickerson Ferreira, Pedro Furtado, "SSB-RT: A Real-Time Data Warehouse Benchmark", finalizado e aguardando ser submetido para IEEE Transactions on Knowledge and Data Engineering.

1.5. Estrutura do Trabalho

Este trabalho foi dividido em seis capítulos, onde cada um aborda assuntos particulares de cada atividade realizada.

O capítulo 1 apresenta a introdução deste trabalho, bem como os objetivos, cronograma, publicações realizadas e a apresentação da metodologia de desenvolvimento utilizada.

O capítulo 2 discorre aspectos das soluções existentes de RTDW. Foram estudados diversos trabalhos acadêmicos e apresentados neste capítulo, onde é destacada a inovação ou técnica utilizada na subsecção que for mais adequado.

No capítulo 3 é apresentada a arquitetura RTDW e detalhes dos mecanismos desenvolvidos para que fosse possível uma integração de dados em tempo real e online. Aqui é apresentado um *overview* do funcionamento da solução, detalhes de cada mecanismo desenvolvido e todas as estratégias de *Merger* definidas.

O capítulo 4 descreve detalhes do benchmark desenvolvido neste trabalho, o SSB-RT. Neste capítulo são descritos quais os elementos principais de um benchmark para avaliar sistemas de DW de tempo real, as configurações necessárias e também os principais testes possíveis de se realizar com o uso do SSB-RT.

No capítulo 5 é responsável pela apresentação do setup experimental e avaliação dos testes realizados com a utilização de uma ferramenta que implementa o benchmark SSB-RT. Neste capítulo estão descritas as avaliações realizadas sob a arquitetura tradicional do DW fazendo integração de dados em tempo real (ou quase) e a avaliação realizada sob a arquitetura proposta e detalhada no capítulo 3.

Finalizando, o capítulo 6 possui as conclusões relacionadas ao trabalho e futuras possibilidades de trabalhos de investigação.

1.6. Metodologia de desenvolvimento

Este tópico irá descrever com detalhes a metodologia de desenvolvimento adotada neste projeto, incluindo todas as fases.

A metodologia adotada neste projeto foi uma adaptação do Scrum que utiliza como modelo de processo de *software* o modelo iterativo e incremental. Criado no inicio dos anos 90, o Scrum é um *framework* bastante usado na produção e na manutenção de software, ou seja, é um conjunto de técnicas e processos que auxiliam, de forma estrutural, o desenvolvimento de produtos. A metodologia pode ser empregada para gestão de produção de qualquer tipo de produto, não sendo aplicada somente para *software* (SCHWABER and SUTHERLAND, 2011). O Scrum é classificado como uma metodologia de desenvolvimento ágil ou leve, possuindo seus fundamentos baseados nas teorias empíricas de controle de processos, isto quer dizer que o conhecimento é construído a partir da experiência e de fatos já ocorridos.

O framework é constituído pelas equipes Scrum associadas a papéis, eventos, artefatos e regras. Estes componentes possuem um objetivo específico e de suma importância para o uso e o sucesso deste conjunto de processos mencionados em (SCHWABER and SUTHERLAND, 2011). Os principais componentes do Scrum estão listados a seguir:

- *Time Scrum*: denominação para equipe que está trabalhando no projeto.
    - *Product Owner*: pessoa responsável pela gestão do desenvolvimento do produto.
    - Equipe de desenvolvimento: equipe de profissionais que trabalham no projeto com a finalidade de desenvolver o produto.
    - *Scrum Master*: pessoa responsável por garantir que o Scrum seja aplicado da melhor forma.
- *Sprint:* é o componente principal do Scrum. Geralmente possui a duração de um mês, ou menos, e ao fim de cada *Sprint* uma versão incremental e funcional do produto é criada.
    - Reunião de planeamento: reunião para identificar o trabalho que será realizado na *Sprint*. Para *Sprints* de um mês, é sugerida uma reunião de oito horas, caso a *Sprint* seja menor, esta reunião deve ser proporcionalmente menor.
    - Reunião Diária: reunião realizada com o intuito de avaliar o progresso do desenvolvimento e definir quais as próximas tarefas previstas até a próxima reunião.
    - Revisão da *Sprint*: reunião cujo propósito é avaliar o incremento desenvolvido e adaptar o *Backlog* do produto, se necessário. O tempo sugerido para esta reunião é de quatro horas para uma *Sprint* de um mês.
    - Retrospectiva da *Sprint*: reunião para avaliar o trabalho da *Time Scrum* e realizar melhorias que serão aplicadas nas próximas *Sprints*.

- Artefatos:
    - *Backlog* do Produto: lista ordenada de todos os requisitos do produto. O *Product Owner* é o responsável pela manutenção deste artefato.
    - *Backlog* da *Sprint*: lista de requisitos que serão desenvolvidos na *Sprint*. Este artefato também inclui um plano de entrega do incremento a ser desenvolvido.

- Incremento: é a soma de todos os itens do *Backlog* do produto que já foram desenvolvidos na *Sprint* atual e nas anteriores.

A Figura 2 ilustra o funcionamento do processo Scrum com seus elementos.

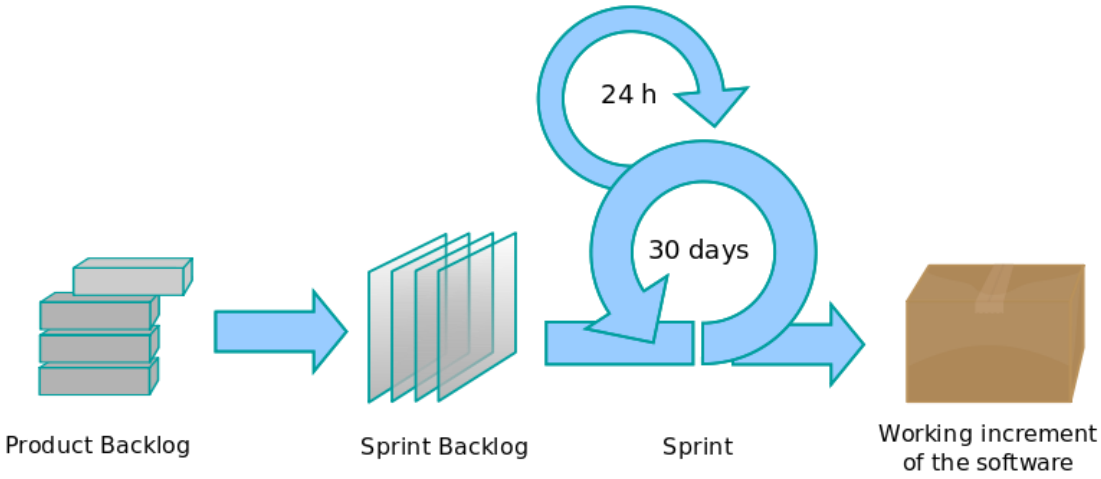


Figura 2: **O processo Scrum (WIKI, 2013).**

Todos os detalhes de como esta metodologia foi utilizada neste projeto podem ser vistos no Apêndice A.

1.7. Análise e gestão de Riscos

O risco, num contexto de projeto, é algo que pode acontecer e suas consequências podem interferir no sucesso do projeto, podendo atribuir ganho ou perda ao mesmo. O risco é composto por dois elementos: a probabilidade de ocorrência e as implicações ou impactos que acarretarão (COOPER et. al, 2005).

A gestão de riscos é um procedimento essencial no desenvolvimento de um produto. A partir da aplicação deste processo é possível minimizar o risco de não atingir os objetivos esperados do projeto e das pessoas envolvidas no desenvolvimento do produto (*stackholders*). Com o uso do gerenciamento de riscos também é possível identificar e tomar proveito das oportunidades que surgem ao longo do projeto. O processo irá auxiliar o gerente do projeto a atribuir prioridades, alocar recursos (humanos, por exemplo) e tomar decisões corretas para que o mesmo avance para atingir

seus objetivos. A gestão de riscos também auxilia o gerente em projetos futuros, onde ele pode tomar decisões com base em ações realizadas anteriormente para minimizar riscos semelhantes no projeto atual (COOPER et. al, 2005).

O processo de análise e gestão de riscos utilizados neste projeto foi baseado no modelo visto em (COOPER et. al, 2005). A Figura 3 mostra o modelo do processo de gestão de riscos e os elementos que o compõem.

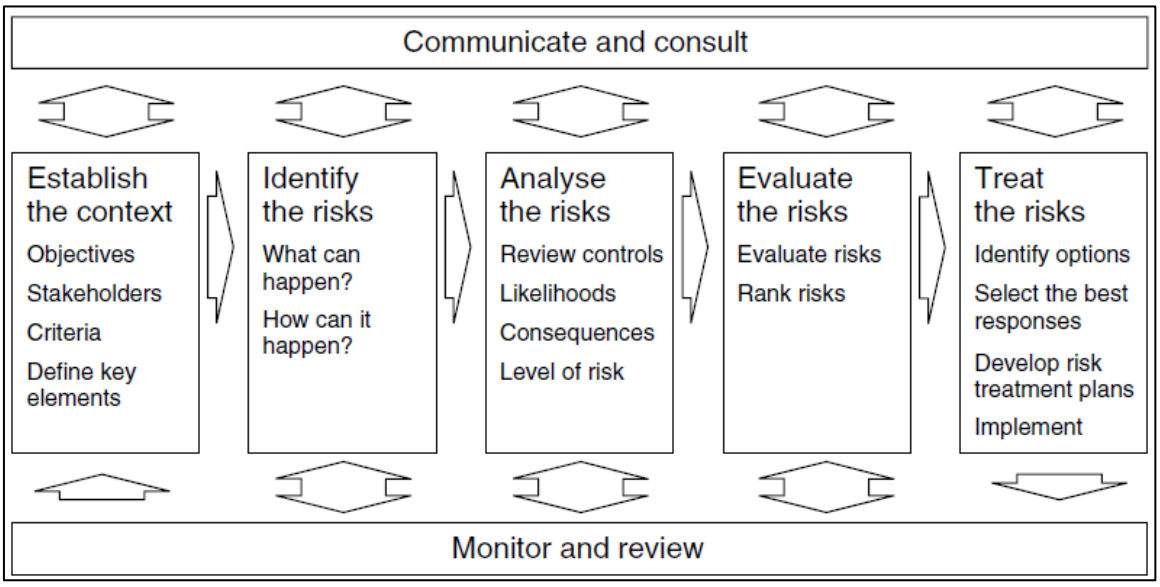


Figura 3: Modelo do processo de gestão de riscos (COOPER et. al, 2005).

Com base na imagem apresentada anteriormente é possível identificar 7 fases sugeridas nesse processo:

- Identificar o contexto: a definição do escopo do projeto e o critério de sucesso são essenciais para identificação do que pode ser considerado risco para o mesmo.
- Identificar riscos: é o procedimento que identifica o que, quando e como algo pode acontecer. A identificação dos riscos foi concretizada através de *brainstorms* ao longo do projeto.
- Analisar riscos: baseado nas informações obtidas nos passos anteriores, este procedimento irá identificar a frequência que os eventos podem ocorrer e o quão suas consequências podem afetar o andamento do projeto.

- Avaliar riscos: neste ponto os riscos serão classificados e priorizados, com o propósito de identificar qual a sequência de tratamento dos riscos e se ele é ou não tolerável.
- Tratar riscos: estabelece e implementa medidas para proceder caso ocorra determinado risco.
- Comunicação: elemento essencial em todo decorrer do processo. O gerente do projeto deve estar muito bem informado sobre o estado atual dos riscos e entendê-los para que o processo de gestão seja realizado com sucesso.
- Revisão: outro elemento que deve ser realizado em todo o ciclo de vida do projeto. Com a revisão constante, aumenta a possibilidade de identificar novos risco e administrá-los de forma progressiva.

Maiores detalhes relacionados à gestão de riscos (ex. levantamento e classificação dos riscos) podem ser vistos no Apêndice A.

Capítulo 2
Estado da Arte

Neste Capítulo analisamos os vários trabalhos relacionados ao projeto e que serviram como base teórica para o desenvolvimento deste.

2.1. Processo de Integração dos dados na DW tradicional

Nesta secção, será explicada a arquitetura tradicional de uma *Data Warehouse* (DW) e os processos que são usados para a manutenção dos dados. Inicialmente, será explicada a arquitetura típica utilizada numa *Data Warehouse*, que possui uma ou mais estrelas base, que apresentam os dados de perspectivas diferentes. Estas DW também podem possuir uma gama de visões de dados agregados, em que uma das finalidades é a diminuição da quantidade de dados, reduzindo o tempo de resposta das consultas.

Na arquitetura apresentada na Figura 4, existe a tarefa de extração dos dados das fontes (E). Estas fontes podem ser bases de dados operacionais, ficheiros XML, CVS, entre outros. Após a obtenção dos dados, torna-se necessário que sejam transformados (T), com o objetivo de adequá-los ao formato das estrelas contidas na DW. Com os dados já devidamente formatados e consistentes, o carregamento (L) é realizado nas estrelas da DW. Por conveniência, ainda foi definida a separação do carregamento (L) em duas partes: o carregamento em si (L) e o refrescamento dos dados agregados (R). O processo completo de integração dos dados foi denominado ETLR.

Além disso, foi considerada a indexação, eliminação e recriação de índices, quando tal se aplica. O carregamento de dados transformados implica na inserção destes dados nas estrelas do esquema de base de dados. A existência de eventuais índices atrasa consideravelmente o processo de inserção, pelo que tradicionalmente é comum remover-se os índices no inicio do processo de carregamento e recriá-los somente após esta tarefa e o refrescamento de dados na DW.

Tendo em conta que a gestão dos dados na DW utilizando chaves estrangeiras é onerosa, algumas implementações não aplicam essas restrições e assumem que a integridade dos dados seja garantida no momento da transformação, na área de estágio.

Outra perspectiva de uma DW, além do processo de ETLR, são as sessões que submetem as pesquisas e recebem os resultados retornados por estas, denominado *Query Workload* (QW).

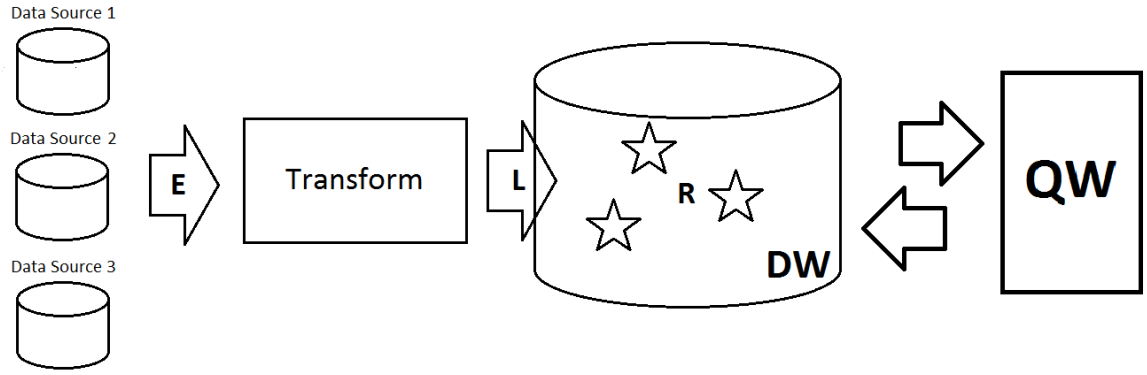


Figura 4: Arquitetura típica de uma DW e a representação de seus principais processos (ETLR).

2.2. Requisitos de uma Data Warehouse de tempo real

Atualmente, várias indústrias necessitam de informações mais atuais possíveis, logo, surge a necessidade cada vez maior de atualizar as informações contidas nas bases de dados de *Data Warehouse* (DW), tendo em vista que este tipo de sistema já não é entendido apenas para planejamento estratégico a longo prazo, mas também para decisões e operações rápidas, possivelmente em tempo real.

As DW tradicionais assumem carregamentos periódicos, com operações pesadas que não se coadunam com análises e pesquisas simultâneas. Por isso, definem períodos *offline* de carregamento, por exemplo, diário, semanal ou mensal. Dada a necessidade do mercado pelas informações atualizadas, se possível em tempo real, surge a necessidade de reduzir a latência existente entre a criação dos dados nas bases de dados operacionais e a atualização destes dados nas DW. Para que o processo de integração de dados de uma DW (descrito na secção anterior) permitisse atingir um nível de integração em tempo real, a arquitetura deve contemplar alguns requisitos básicos relacionados a esta característica.

Os autores de (BRUCKNER et. al, 2002) definem basicamente três características relevantes para uma *Data Warehouse* de tempo real, que são:

- Integração contínua de dados: permitir que novos dados, vindos de diferentes fontes de dados, sejam integrados (*Extract - Transoform - Load - Refresh*) em tempo real ou quase real.
- Mecanismos de suporte a decisão: esses mecanismos devem auxiliar os utilizadores em suas tomadas de decisão realizando recomendações com base em regras pré-definidas, por exemplo.
- Alta disponibilidade: o sistema deve ser capaz de permitir acesso às análises em qualquer momento.

2.3. Soluções para cada processo do ETLR para tempo real

Como citado anteriormente, para que a DW tenha seus dados atualizados constantemente e em períodos reduzidos, é preciso que todas as atividades do processo de integração (ETLR) sejam capazes de atingir esse ritmo de tempo real. Nesta secção serão vistas soluções encontradas para cada uma das atividades e como os autores se propõem a resolver esse problema. Além disso, serão vistas soluções de arquiteturas que permitem esta integração em tempo real.

2.3.1. Extração e Transformação dos dados

A técnica mais utilizada na extração de dados para um sistema RTDW é conhecida como *Change Data Capture* (CDC). Esta técnica é responsável pela identificação de dados que devem ser integrados ao DW.

Em (SHI et. al, 2008) os autores propõem um *framework* com as seguintes características para conseguir integrar dados em tempo real: avaliação prévia dos dados das fontes para verificar integridade, evitando carregamentos desnecessários; estabelecimento de prioridades de carregamento, por forma a carregar prioritariamente dados mais importantes; agendamento de transformação e carregamentos, a fim de otimizar a utilização dos recursos.

Os autores de (RAM and DO, 2000) analisaram as variadas formas de extração de dados fontes (*triggers, log-based, timestamp-based*, etc.) e constataram que o melhor entre elas é a chamada Op-Delta (*Operation Delta*). Este método (*log-based*) registra operações, ao invés de registrar os dados alterados, aplicando essas operações de forma

alterada sobre a DW para integrar as modificações. Esta solução torna a extração mais rápida e menos intrusiva.

Outro caso de uso do CDC pode ser visto em (GUERRA and ANDREWS, 2011). Neste caso, os autores indicam como a melhor alternativa de utilização desta técnica, a extração com base em *log* do motor da base de dados. A solução sugerida pelos autores é utilizar uma ferramenta que realiza a leitura do ficheiro de *log*, identificando quais os dados que são relevantes para a atualização, avançando para o próximo passo, chamando a função de transformação. Os autores ainda sugerem alternativas, tais como: o uso de gatilhos, filas de mensagens e o uso de um campo com TIMESTAMP.

Em (JÖRG and DESSLOCH, 2010) os autores citam soluções para o uso do CDC. Uma solução apresentada pelos autores foi para as fontes de dados que possuem sistemas legados ou que por algum motivo fazem o armazenamento dos dados no sistema de arquivos, técnica chamada de *Snapshot sources*. A utilização de recursos como *locks* em tabelas ou ficheiros foi outra opção listada pelos autores desse artigo.

Os autores de (ZHU et. al, 2008) propõem uma solução diferente das demais. Neste artigo, os autores utilizaram um CDC com base em *Web Services*. O sistema inclui diversos subsistemas e um coletor de dados. Cada subsistema possui um *Web Service* servidor que pode receber requisições através da internet. Então, o coletor de dados utiliza uma função para capturar os dados alterados existentes em cada um dos subsistemas, acessando estes *Web Services*.

Além dos artigos vistos, também existem soluções comerciais que se propõem a realizar a extração de informação das fontes de dados, vistos em (ORACLE, 2013), (ETI, 2013) e (INFORMATICA, 2013).

Já relacionado às transformações dos dados, os autores de (GOLDENGATE, 2009) apresentam duas soluções para essa tarefa. A primeira solução sugere que as transformações sejam aplicadas aos dados na própria fonte de dados, por meio do uso de *scripts* ou alguma ferramenta que realiza ETL. A solução seguinte sugere que esta atividade seja atribuída ao próprio DW, ou seja, realizando o carregamento antes das transformações. Este processo é conhecido como ELT (*Extract - Load - Transform*).

Outra solução para otimização do processo de transformação dos dados pode ser vista em (VASSILIADIS and SIMITSIS, 2008). Os autores recomendam que esta tarefa, como também a extração, sejam realizadas por outra aplicação que seja escalável, com isso existindo a possibilidade de paralelizar o processamento dessas operações, obtendo um *throughput* maior.

Em (WAAS et. al, 2012), os autores sugerem outra solução com base em vistas materializadas, em que se segue uma estratégia parecida com ELT, sendo os dados carregados em bruto para "*Landing Pads*", e a lógica das transformações é colocada em vistas (organizadas em pilha) e realizadas sob demanda, ou seja, ao receber uma requisição de análise ou consulta.

Os autores em (POLYZOTIS et. al, 2007) sugerem uma otimização na fase de transformação, relacionada à forma de junção de tabelas (*join*), chamada por eles *MeshJoin*. Este algoritmo utiliza *streams* e faz varreduras sequenciais na base de dados, diminuindo o acesso a disco e aumentando a performance da operação de junção.

Outro trabalho efetuado para otimizar transformações pode ser visto em (YU, 2006), cujos autores propõem uma opção que usa apenas vistas normais e vistas materializadas para realizar a limpeza e outras transformações nos dados.

2.3.2. Carregamento e refrescamento de vistas

Uma das operações que representam maior custo na integração de novos dados é o carregamento (*Load*). Com o objetivo de diminuir esse custo, considerado alto, foram sugeridas diversas formas de realizar tal operação.

Uma estratégia de carregamento encontrada para permitir integração em tempo real na DW foi vista em (SANTOS and BERNARDINO, 2008). Os autores deste artigo apresentaram uma solução no que concerne a estrutura da DW. Eles sugerem a criação de tabelas temporárias que irão armazenar os dados mais recentes, além de manter as estrelas originais do esquema da DW. Estas tabelas temporárias possuem os mesmos atributos das originais, sendo que não possuem nenhum tipo de chaves estrangeiras, chaves primárias, restrições ou índices, permitindo um elevado ritmo de carregamento.

Em (ZUTERS, 2011) os autores também utilizam a lógica de criar tabelas temporárias, mas adicionam uma partição para diferentes níveis de informação, por exemplo, uma para a última hora e uma para o último dia, sendo atualizadas de acordo com sua periodicidade. Outro fator importante é que as partições serão atualizadas se, e somente se, não houver consultas a correr.

Outra solução foi estudada em (NGUYEN and TJOAV, 2003), cujos autores propõem a construção de uma ferramenta, tendo como base *Data Stream* que permita *freshness* dos dados e integração contínua, além de outros critérios avaliados pelos autores. A inserção de dados neste tipo de estrutura é muito rápida. A problemática desta

solução é que os resultados das consultas são aproximados, pelo fato do elevado custo necessário para produzir os resultados exatos, quando se utilizam *data stream* contínuos.

Os autores de (ZHU et. al, 2008) indicam uma solução com base em filas e caches, que possui um gestor de filas que agrupa os dados. A partir do momento que a fila é preenchida completamente, os dados são carregados. Além disso, os autores criaram uma estrutura hierárquica de caches, onde cada um armazena informações com um determinado período de atualização. Os dados passam por todos os níveis de cache até serem carregados na DW, que possui todos os dados históricos.

No tocante ao refrescamento de dados agregados, o artigo (CONDIE et. al, 2010) apresenta uma solução para otimização desta tarefa. Esta solução consiste no uso de funções *Map Reduce* para agregar os dados, essas funções são usadas em duas fases. A primeira fase é responsável pela execução da função *Map*, que irá extrair as chaves e registros. Em seguida, realiza um particionamento dessas informações com base nas chaves dos registros. A segunda fase realiza a junção dos registros que possuem a mesma chave e passa o resultado para a função *Reduce*, assemelhando-se ao *group by* utilizado em bases de dados.

Em (WAAS et. al, 2012) os autores propõem uma alternativa ousada para o refrescamento das vistas materializadas, em que a manutenção destas vistas é realizada sob demanda. Ao receber uma requisição de consulta, o sistema verifica se a vista está atualizada, caso contrário, o refrescamento da vista é realizado e, então, a consulta é executada. O desempenho desta solução é garantido através de um algoritmo que não necessita de re-computar todos os dados, apenas na diferença existente nas tabelas base.

O trabalho de (SCHALLEHN et. al, 2001) apresenta técnicas de agrupamento e agregação que podem ser usadas nas DW. Os autores criaram uma extensão da linguagem SQL (FraQL) e realizam agrupamentos por similaridade ou definições criadas pelos utilizadores.

2.4. Arquiteturas RTDW

Iniciando pelo artigo (VASSILIADIS and SIMITSIS, 2008), os autores sugerem uma arquitetura que possui alguns mecanismos que garantem o refrescamento constante dos dados na DW. Esta arquitetura possui basicamente cinco camadas: 1) camada onde é realizada a extração dos dados das fontes; 2) camada que realiza a sincronização dos

dados extraídos para uma camada intermediária, denominada *Data Processing Area* (DPA). 3) Nesta camada é onde são realizadas as transformações. 4) A camada que realiza a sincronização entre a DPA e a *Data Warehouse* (DW). 5) camada de armazenamento (DW).

Os autores de (ZHU et. al, 2008) recomendam uma solução com base numa arquitetura SOA (*Service Oriented Architecture*). A solução dos autores evidencia um mecanismo coletor que acessa *Web Services* hospedados em subsistemas, extraindo os dados e os armazenando em *caches,* que são organizados de forma hierárquica e possui os dados de determinados períodos de atualização. Após os dados passarem por todos os *caches*, então são armazenados na DW, que é o mecanismo que armazena todos os dados históricos. Esta solução conta também com um componente que realiza a junção das informações contidas nos *caches*.

A arquitetura apresentada em (SANTOS and BERNARDINO, 2008) conta com tabelas temporárias que armazenam os dados mais recentes e que são integrados de forma contínua e rápida, já que não possui restrições ou índices. A atualização das tabelas base da DW é feita de maneira tradicional. A junção dos dados é realizada com base na alteração da consulta original, onde a consulta final faz duas consultas (tabela base e tabela temporária) e a união (UNION ALL) dos resultados.

Os autores de (WAAS et. al, 2013) optaram por escolher uma estratégia de integração diferente da convencional (ETL). A estratégia utilizada foi ELT (*Extract - Load - Transform*). Os dados extraídos das fontes são logo inseridos numa área de estágio, denominada *Landing Pad* (LP). Outro mecanismo existente na arquitetura possui várias vistas materializadas (*Materialized Views* - MV) e foi criado para atender às consultas submetidas pelos utilizadores. As MV são refrescadas sob demanda, isto é, ao receber a requisição de uma consulta.

No artigo visto em (ZUTERS, 2011), foi apresentada uma arquitetura que utiliza múltiplas partições que armazenam as informações referentes a um determinado período de atualização, exemplo: uma hora e outra para um dia. Entretanto, a atualização de uma partição para outra é realizada assumindo períodos *offline*.

Em (BRUCKNER et. al, 2002), os autores recomendam uma arquitetura que aumenta o desempenho do processo ETL movendo os dados de diferentes níveis sem a necessidade de armazená-los em ficheiros. Para garantir esse desempenho, os autores utilizaram os *ETLets* e um conjunto de componentes EJB (*Enterprise Java Beans*), que

extraem os dados utilizando conectores JDBC de alta velocidade, transformando e convertendo para o formato XML.

Os autores de (THOMSEN et. al, 2008) apresentam uma solução para RTDW que garante um excelente desempenho na integração contínua de dados com consultas a correr em simultâneo. Para isto, os autores criaram um produtor de dados (*driver* JDBC especializado) que disponibiliza as informações para os utilizadores sob demanda. Objetivando um bom desempenho, os autores criaram uma estrutura em memória. Também foi criado um mecanismo consumidor para as consultas.

A indústria também está investindo nesse conceito, soluções para RTDW de grandes empresas podem ser vistas no mercado. Em (ORACLE, 2012) os autores utilizam ferramentas Oracle, tais como o Oracle Data Integrator (ODI) e o Oracle GoldenGate, que garantem integração de dados em tempo real. Outra solução comercial pode ser vista em (VERTICA, 2013), onde os criadores sugerem uma arquitetura hibrida, ou seja, armazenamento em disco e em memória. O armazenamento em memória possui baixa latência, é segmentado e possui um alto ritmo de inserção e os dados são migrados para o armazenamento em disco de forma assíncrona. Já o armazenamento em disco é realizado de forma tradicional.

Capítulo 3

Proposta de arquitetura de Data Warehouse em tempo real (RTDW)

Neste capítulo é apresentada a proposta de arquitetura RTDW. Diversamente da arquitetura comum, esta nova arquitetura possui elementos que permitem a atualização dos dados de forma constante e contínua, com a finalidade de manter a base de dados OLAP (*Online Analytical Processing*) com as informações mais recentes.

Os três componentes principais da arquitetura (4,5 e 6), ilustrados na Figura 5, são o *Dynamic Data Warehouse* (D-DW), o *Static Data Warehouse* e o *Merger*. Além disso, assumimos alguns elementos de suporte necessários para a extração e transformação dos dados em tempo real (2 e 3), o *Input Stream Buffer* (ISB) e o T-ETL.

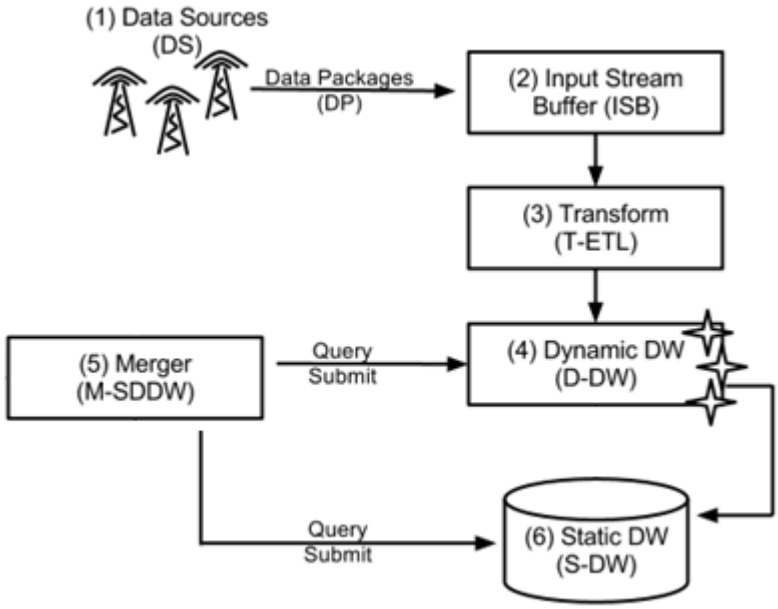


Figura 5: Arquitetura proposta para o RTDW.

3.1. *Overview* do sistema

Inicialmente, os dados são extraídos com certa frequência dos *Data Sources* (DS). Este intervalo de tempo pode ser definido através de configuração de ferramenta. Não enquadra-se como objetivo deste trabalho definir como os dados são extraídos das fontes.

Uma forma comum de extração e que garante tempo real nesta fase é a utilização de *Change Data Capture* (CDC), como visto em (ZHU et. al, 2008), (ORACLE, 2012), (SHI et. al, 2008), (RAM and DO, 2000), (GOLDENGATE, 2009).

Após esta extração, os dados são guardados numa estrutura denominada ISB (*Input Stream Buffer*). O ISB recebe os pacotes de dados (*Data Packages* - DP) até atingir um número de linhas ou determinado tempo, definido pelo administrador do DW. Desta forma, o ISB gera *mini-batches* para transformação. O ISB pode ser configurado para armazenar dados em memória ou ficheiros em disco, conforme o tamanho dos dados e outros fatores como persistência.

Outro componente típico é o T-ETL. Ele é um componente independente, responsável pelas transformações dos dados extraídos das fontes. Por se tratar de um componente independente, assume-se o mecanismo possa ser implementado de diversas formas diferentes, com o fito de otimizar o processo de transformações. Algumas formas de otimização dessa atividade podem ser revistas em (VASSILIADIS and SIMITSIS, 2008).

Numa arquitetura de DW tradicional, os carregamentos de dados são pesados e efetuados *offline* (sem consultas a correr), devido a vários fatores: a necessidade de eliminar os índices e outras restrições (por exemplo, chaves estrangeiras) antes carregar os dados, recriando-as posteriormente, com o objetivo de otimizar ao máximo o carregamento dos dados; a necessidade de refrescar as vistas materializadas, sumários e todas as estruturas envolvidas na DW; problemas de performance se tivermos concorrência entre pesquisas e carregamento de dados.

Conforme proposto neste trabalho, as dificuldades são resolvidas, com o intuito de permitir a integração de dados online e em tempo real. Os dados são carregados num componente pequeno, eficiente, possuindo uma capacidade de carregamento muito alta, o D-DW. Este componente foi criado para permitir um ritmo de integração extremamente elevado, possuindo apenas os dados recentes (por exemplo, um dia). Por ter uma quantidade de dados razoavelmente pequena, também possui um bom desempenho para processar as pesquisas de forma rápida. Opcionalmente, o D-DW pode ser colocado em memória.

Mas uma DW é um armazém de dados históricos, detentor de uma quantidade significativa de dados. Logo, o processamento desses dados se torna uma tarefa com um alto custo. Uma solução é possuir este armazém de dados históricos em outro componente, o S-DW. Dessa forma, evita-se a simultaneidade das consultas com o

carregamento de dados sobre a parte pesada do sistema. A integração dos dados da D-DW na S-DW é feita após um determinado período, definido pelo administrador da DW. O processo é realizado de forma tradicional e em períodos *offline* (por exemplo, à noite). Ao fim da integração o D-DW é esvaziado. Dessa forma, o D-DW permite ter os dados recentes, integrados em real-time, além de pesquisas executadas rapidamente. Já o S-DW tem a grande quantidade de dados históricos e as pesquisas sobre estes.

Com o auxílio de um terceiro elemento, denominado *Merger* (M-SDDW), onde este identifica quais os dados a pesquisa necessita: somente dados recentes (D-DW), somente históricos (S-DW) ou de ambos (D-DW + S-DW). Caso a pesquisa necessite de dados históricos e recentes, o *Merger* recupera os dados das duas DWs e realiza a junção destes.

O esquema da D-DW é semelhante ao existente na S-DW. Para efeitos de otimizar a integração dos dados na D-DW, o esquema de tabelas utilizado nesta base de dados não possui nenhum índice ou chaves estrangeiras. Como a quantidade de dados é pequena, se comparada com a S-DW, não são criadas tabelas de agregação e sumários na D-DW, o que elimina a necessidade de refrescamento constante dessas vistas e sumários. Em vez disso, são criadas vistas não materializadas correspondentes.

O S-DW é equivalente a uma DW tradicional, armazenando os dados históricos e sua atualização é dependente do tempo definido pelo administrador da DW. O esquema adotado no S-DW é o modelo em estrela, amplamente utilizado em projetos de DW. Tipicamente, também possui um conjunto de índices (b-tree e bitmap), chaves estrangeiras, restrições de integridade de dados, vistas de dados agregados, entre outros. O carregamento dos dados no S-DW segue o processo tradicional, onde normalmente os índices são eliminados no início do processo e recriados no fim, por uma questão de eficiência do carregamento. Nesse carregamento são feitas a agregação dos dados e a atualização nas tabelas de agregação do S-DW. Posteriormente, o D-DW é totalmente limpo, não restando nenhuma informação e os índices do S-DW são, então, recriados.

Com o emprego desta arquitetura, é possível garantir a integração constante de dados na DW e o acesso aos dados mais recentes.

3.2. Static Data Warehouse (S-DW)

Numa base de dados do tipo DW a quantidade de dados existente pode ser expressivo, podendo chegar a casa dos Terabytes. Logo, a atualização dessa BD com dados novos possui um custo elevado, o que compromete as capacidades de tempo real. Para resolver este problema, separamos a DW em dois componentes de armazenamento, sendo o S-DW o componente com enorme quantidade de dados históricos, estes novos dados são integrados na S-DW de forma tradicional e *offline*, e o D-DW (detalhado na secção 3.2). O esquema desta base de dados é baseado no modelo estrela, proposto por Kimball e Caserta (KIMBALL and CASERTA, 2004). A Figura 6 mostra o conceito deste esquema.

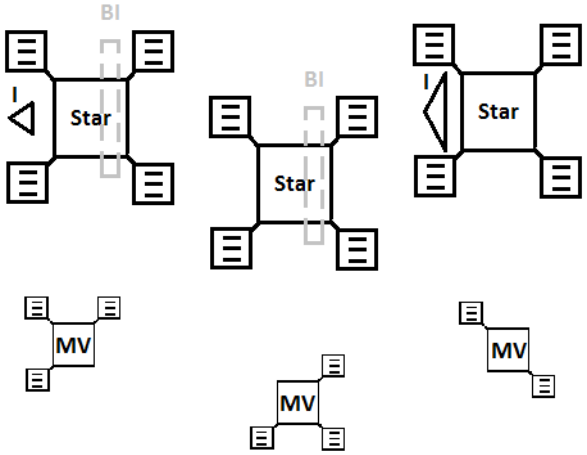


Figura 6: Modelo conceitual do esquema da base de dados S-DW.

Com base na figura acima, identifica-se os elementos que estão presentes na S-DW (os mesmos presentes em DW tradicionais), que contribuem para o elevado custo de sua atualização. Para garantir a integridade dos dados, existem as ligações entre as tabelas de fato e as dimensões (chaves estrangeiras), as restrições de valor único (chave primária), entre outras. A criação de índices na DW também é uma prática utilizada e detém o objetivo de diminuir o tempo de resposta das consultas realizadas. A S-DW possui vários índices, de diferentes tipos (I - b-tree e BI - bitmap). O custo de reorganização ou recriação destes índices é considerável. Outro elemento existente na S-DW são as tabelas de dados agregados (MV), que são responsáveis pelo armazenamento, de forma agrupada, dos dados presentes nas tabelas de fato.

3.3. Dynamic Data Warehouse (D-DW)

Ao contrário da S-DW, este componente contém uma quantidade pequena de dados. Trata-se dos dados de um determinado período recente, por exemplo, o último dia. As pesquisas sobre a D-DW são sempre rápidas, em termos absolutos, visto que a quantidade de dados é razoavelmente pequena. Por esse motivo, essa DW não possui necessidade de ter índices. A Figura 7 mostra, em alto nível, o esquema da D-DW.

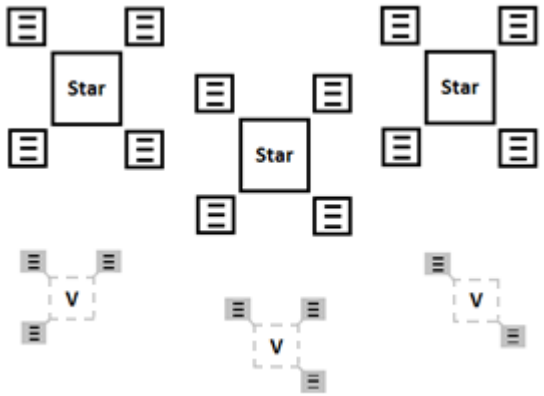


Figura 7: Modelo conceitual do esquema da base de dados D-DW.

Observando a figura acima, é notável que possui uma complexidade inferior à mostrada na S-DW. Podemos ver que não existem índices (de nenhum tipo) e foi assumido que a integridade dos dados para esta BD é realizada no componente T-ETL. Assim, não há necessidade da criação de nenhum tipo de restrição e ligações entre as tabelas de fato e dimensões. Com essa alteração no modelo da BD, conseguimos um aumento no desempenho do carregamento dos dados e, por possuir poucos dados, o impacto da falta dos índices na performance das consultas é pequeno.

Outro fator que difere entre os componentes, é o fato da D-DW não possuir as tabelas de dados agregados, ao contrário da S-DW. Para substituir esse componente, foram criadas vistas não materializadas (consultas) que simulam as agregações. Isto elimina a necessidade de refrescamento das vistas com os dados que estão constantemente a ser carregados (tempo real), porém existe um impacto relativamente pequeno (por possuir poucos dados) pelo fator da ausência dos índices.

3.4. Merger

Como a arquitetura RTDW proposta neste trabalho possui dois componentes de armazenamento, uma base de dados para os dados históricos (S-DW) e a outra para os dados recentes (D-DW), as consultas devem ser alteradas para realizar a junção dos resultados existentes nas duas DW, quando necessário. Esta operação é realizada pelo *Merger* e causa um *overhead* no tempo de resposta de uma consulta, se for comparada com a execução da mesma consulta apenas numa das DW. Com a finalidade de diminuir ao máximo este *overhead*, foram definidas algumas estratégias de merge, divididas em três categorias: *Single-instance, Independent-Instance with migration e Independent-instance, no migration*.

3.4.1. Single-Instance

Este conceito leva em consideração que ambas as DW (S-DW e D-DW) estejam numa única instância de BD. Com isso, há a necessidade de alterar, unicamente, a consulta, com o objetivo de selecionar os dados em ambas as DW e realizar a junção dos resultados utilizando um UNION ALL. Por utilizar essencialmente comandos SQL para realizar a junção, este método foi chamado SQL-*based*. A consulta original e o resultado da alteração da consulta pode ser visualizada abaixo.

Antes:

```
select sum(l_revenue), d_year, p_brand
from lineorder, date_dim, part, supplier
where l_orderdate = d_datekey and
l_partkey = p_partkey and
l_suppkey = s_suppkey and
p_brand = 'MFGR#2221' and
s_region = 'Nevada'
group by d_year, p_brand
order by d_year, p_brand;
```

Depois:

```
select sum(l_revenue), d_year, p_brand
from (select l_revenue, d_year, p_brand
      from SDW.lineorder, SDW.date_dim, SDW.part, SDW.supplier
      where l_orderdate = d_datekey and
      l_partkey = p_partkey and
      l_suppkey = s_suppkey and
      p_brand = 'MFGR#2221' and
      s_region = 'Nevada')
UNION ALL
(select l_revenue, d_year, p_brand
      from DDW.lineorder, DDW.date_dim, DDW.part, DDW.supplier
      where l_orderdate = d_datekey and
      l_partkey = p_partkey and
      l_suppkey = s_suppkey and
      p_brand = 'MFGR#2221' and
      s_region = 'Nevada')
group by d_year, p_brand
order by d_year, p_brand;
```

Como esta categoria não utiliza mais de uma instância de BD, o próprio motor da base de dados se encarrega de realizar a operação de junção dos dados. O conceito de *Single-instance* pode ser conferido na Figura 8.

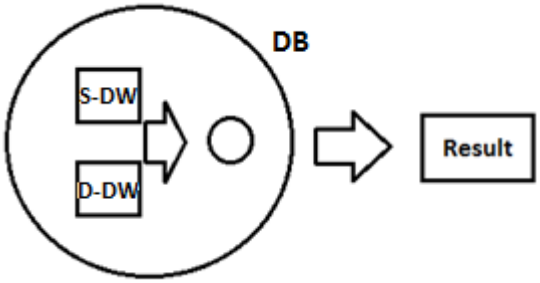


Figura 8: Modelo conceitual do Merger Single-instance.

3.4.2. Independent-Instance with migration

Neste caso, a S-DW e a D-DW estão em instâncias independentes e separadas, ou seja, estão em máquinas diferentes, com o fito de evitar a simultaneidade entre o carregamento de dados em tempo real e as consultas pesadas. Os dados resultantes de determinada consulta será migrado para a outra instância e depois a junção é concretizada (veja Figura 9).

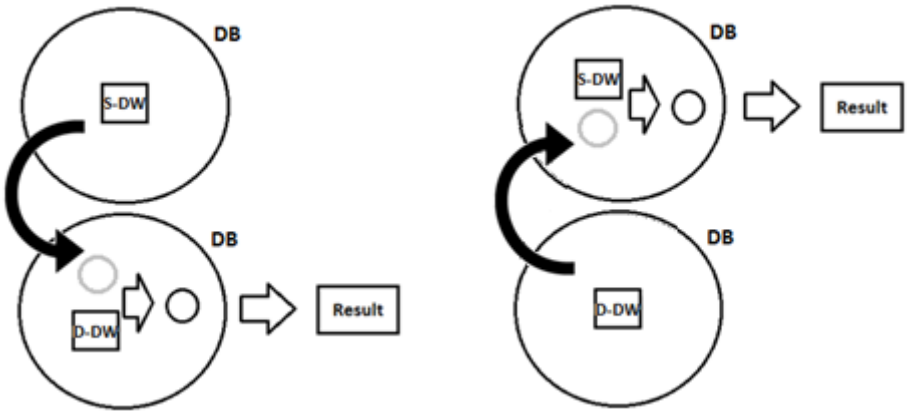


Figura 9: Merger Independent-instance with migration, modelo conceitual.

Nesta categoria é possível realizar a migração dos dados de diversas formas. Primeiro definimos uma forma que realiza a consulta em uma das bases de dados (D-DW ou S-DW) e depois, com base nos metadados do resultado obtido, é criada uma tabela temporária no outro componente, que armazenará os resultados da pesquisa. A inserção do resultado da pesquisa pode ser feita utilizando inserções uma a uma (*driver* JDBC em nossa plataforma Java) ou através de *bulk load*. Após a inclusão dos resultados, a junção dos resultados das duas consultas é concretizada e o resultado final é retornado. Este método pode migrar os dados tanto da S-DW para a D-DW como vice-versa. A migração dos resultados da D-DW para S-DW é plausível porque a quantidade de dados resultante da consulta sobre a D-DW é previsivelmente menor (por se tratar da representação dos dados mais recentes) que sobre a S-DW.

A migração dos dados também pode ser feita usando diversos meios de troca de dados, muitos deles disponibilizados pelos motores de base de dados. Na parte experimental foi testada a utilização de DBLinks, do Oracle. Este recurso foi criado para ligação de bases de dados heterogêneas, com o intento de facilitar a execução de consultas que acedam aos dados em máquinas distintas (DBLINK, 2013). Primeiramente, é necessária a criação do DBLink na instância da BD que irá receber a consulta. Este DBLink apontará para a outra instância da BD. Após essa operação, existirá uma ligação entre as duas instâncias, permitindo acesso aos dados. Com a ligação criada, basta alterar a consulta. Um exemplo de uma consulta alterada para realizar a junção pode ser analisado a seguir.

```
select sum(l_revenue), d_year, p_brand
from (select l_revenue, d_year, p_brand
      from lineorder, date_dim, part, SDW.supplier
      where
      l_orderdate = d_datekey and
      l_partkey = p_partkey and
      l_suppkey = s_suppkey and
      p_brand = 'MFGR#2221' and
      s_region = 'Nevada')
UNION ALL
(select l_revenue, d_year, p_brand
      from lineorder@ddw, date_dim@ddw, part@ddw, supplier@ddw
      where
      l_orderdate = d_datekey and
      l_partkey = p_partkey and
      l_suppkey = s_suppkey and
      p_brand = 'MFGR#2221' and
      s_region = 'Nevada')
group by d_year, p_brand
order by d_year, p_brand;
```

3.4.3. Independent-Instance no migration

Neste caso, em vez de migração dos dados de uma instância para outra, iremos utilizar um terceiro elemento que fará a junção dos resultados. A Figura 10 ilustra o Independent-instance, no migration.

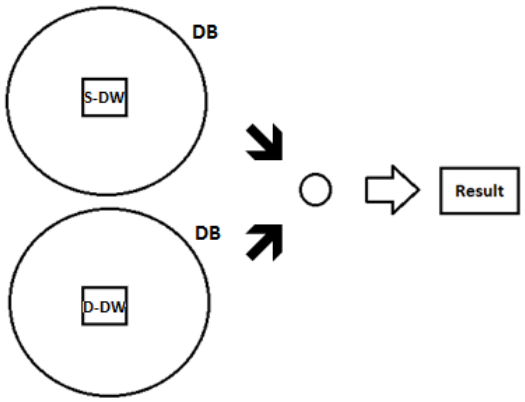


Figura 10: Independent-instance, no migration.

Nota-se que, as consultas são executadas nos componentes, independentemente e em paralelo. Os resultados destas consultas passam para um terceiro elemento (*Merger*). No protótipo desenvolvido para este trabalho, o *Merger* é uma base de dados em memória, por questões de eficiência.

3.5. Eficiência do Carregamento

A eficiência no carregamento de novos dados numa DW é um fator essencial, principalmente tratando-se de um RTDW, onde existem carregamentos periódicos e consultas a correr em paralelo. Esses dois processos, ambos com um elevado custo, não devem exercer influência entre eles. Aumentando a eficiência do carregamento de dados, o tempo de duração dessa operação diminui. Consequentemente há uma diminuição na interferência que exerce sobre as consultas.

Na arquitetura proposta neste trabalho, os carregamentos contínuos são realizados na D-DW e a maior quantidade de dados está presente na S-DW. Se ambos estiverem numa mesma máquina (S-DW e D-DW), os recursos estarão partilhados e existirá uma concorrência entre os processos executados. Logo, uma consulta efetuada no S-DW (custo elevado) irá influenciar o desempenho da integração constante dos dados no D-DW e vice-versa. Para sanar esse problema e otimizar a eficiência do carregamento periódico de novos dados e consultas, o S-DW e o D-DW são distribuídos em máquinas diferentes e independentes. Desta forma, não haverá interferências externas nos processos, sejam eles no carregamento de dados ou nas consultas.

Outra medida tomada para incrementar o desempenho dos carregamentos na D-DW é a criação do esquema da base de dados sem quaisquer recursos que possam interferir na performance dessa operação. Nessa instância da base de dados não são criadas nenhum tipo de restrições (NOT NULL, CHECK, FOREIGN KEY, entre outros), e também devido à pequena quantidade de dados os índices não serão necessários.

Além disso, outro fator que influencia na atualização dos dados é a estratégia de carregamento. Com a estratégia de *bulk load*, é possível o emprego de mini-batches de tamanhos variados, definidos pelo administrador da DW. Mas, se for necessário integrar linha a linha com enorme eficiência, também é admissível usar inserções diretas, o que possibilita a inclusão dos tuplos um a um ou em lotes. O tipo de estratégia aplicada a este nível depende da situação e da decisão de um administrador da base de dados.

3.6. Eficiência e freshness do Query Workload

As consultas realizadas em DW geralmente possuem um elevado custo, devido à quantidade de dados e processamento necessários. Essas consultas também possuem a necessidade de ter um menor tempo de resposta, não podendo sofrer influência de outros processos. Portanto, não é uma alternativa coerente colocar pesquisas e carregamentos a correr de forma simultânea.

A arquitetura sugerida neste trabalho possui um aspecto importante que é a separação das bases de dados D-DW e S-DW em instâncias, nomeadamente, em máquinas distintas e independentes. Com isso, possibilita-se a minimização do impacto do processo de carregamento de dados no tempo de resposta das consultas.

Outro ponto que esta arquitetura tende a otimizar é a quantidade de sessões simultâneas que o sistema tem capacidade de possuir. Com o aumento da eficiência de ambos os processos (carregamento e consultas), é possível elevar a quantidade de sessões que o sistema suporta mantendo, os tempos de resposta adequados.

Uma qualidade de uma solução RTDW é permitir o *freshness* nas consultas realizadas pelos utilizadores, ou seja, consentir o acesso às informações mais atuais. Com a utilização da arquitetura indicada, é possível garantir esta característica desejada para o *query workload* do DW.

3.7. Obtendo eficiência em tempo real nas transformações

Se for discutida a questão de atualização de bases de dados em tempo real, é importante lidar com a extração e transformação dos dados de origem de forma ágil. Com a finalidade de garantir uma atualização em tempo real de uma DW, é impreterível que estas operações também assegurem um *throughput* de dados comparável com a taxa de atualização da DW. Esta parte não constitui o cerne da arquitetura discutida neste trabalho, mas devido à sua importância, o debate torna-se indispensável.

Para extrair os dados, existem algumas técnicas utilizadas na indústria. Uma das mais conhecidas e indicadas é o *Change Data Capture* (CDC), como visto em (ZHU et. al, 2008), (ORACLE, 2012), (SHI et. al, 2008), (RAM and DO, 2000), (GOLDENGATE, 2009). Esta técnica é usada de várias formas (algorítmos, *triggers*, entre outros) e é

responsável pela identificação dos dados que são relevantes, bem como armazená-los num *log* para integração posterior.

Outro processo relevante na integração dos dados numa DW é a transformação. As transformações são um conjunto de operações realizadas sobre os dados de origem com o desígnio de limpar os dados e deixá-los no formato presente nas estrelas do DW. Este processo é particular e variável de acordo com a necessidade de quem implementa uma DW. Em (WYATT et. al, 2009) o autor lista uma série de tipos de transformações comuns neste processo. A otimização deste processo é um fator crucial quando se pensa em RTDW. Dependendo da quantidade e do nível de complexidade das transformações, este procedimento deve usar estratégias de otimização, com o objetivo de garantir um ritmo aceitável de saída de dados.

Como possíveis técnicas para otimização desse processo, tem-se a paralelização das transformações (VASSILIADIS and SIMITSIS, 2008), *lookups* realizados em memória ou até mesmo a execução de certas transformações no momento da extração, como visto em (GOLDENGATE, 2009). O foco deste trabalho não consiste em especificar detalhes dessas técnicas de otimização do processo de transformação, mas fortalecer a ideia de que os processos de extração e transformação devem garantir um ritmo de tempo real para a nossa arquitetura de integração e processamento.

Capítulo 4

Real-Time DW Benchmark (SSB-RT)

Os benchmarks de DW existentes atualmente não estão indicados para avaliar soluções para DW de tempo real. Por essa razão, neste capítulo propomos um *Benchmark* de *Real-time* DW que permite avaliar soluções RTDW, sendo útil em particular na avaliação da nossa proposta de arquitetura RTDW.

O Benchmark foi criado por alteração de um já existente para avaliação de DW normais. Aproveitamos o *Star Schema Benchmark* (SSB) (O'NEIL et. al, 2009) como fonte inicial, ao qual adicionamos mecanismos de ETL e outras partes necessárias para simular um ambiente de DW com integração em tempo real. Este *Benchmark* chama-se SSB-RT.

O SSB-RT é constituído por três partes principais: o ETL, a estrutura do DW e as sessões de pesquisa que correm um *Query Workload* (QW).

Para melhor entendimento dos processos existentes no SSB-RT, a Figura 11 mostra o diagrama de sequência com os passos realizados nesse processo.

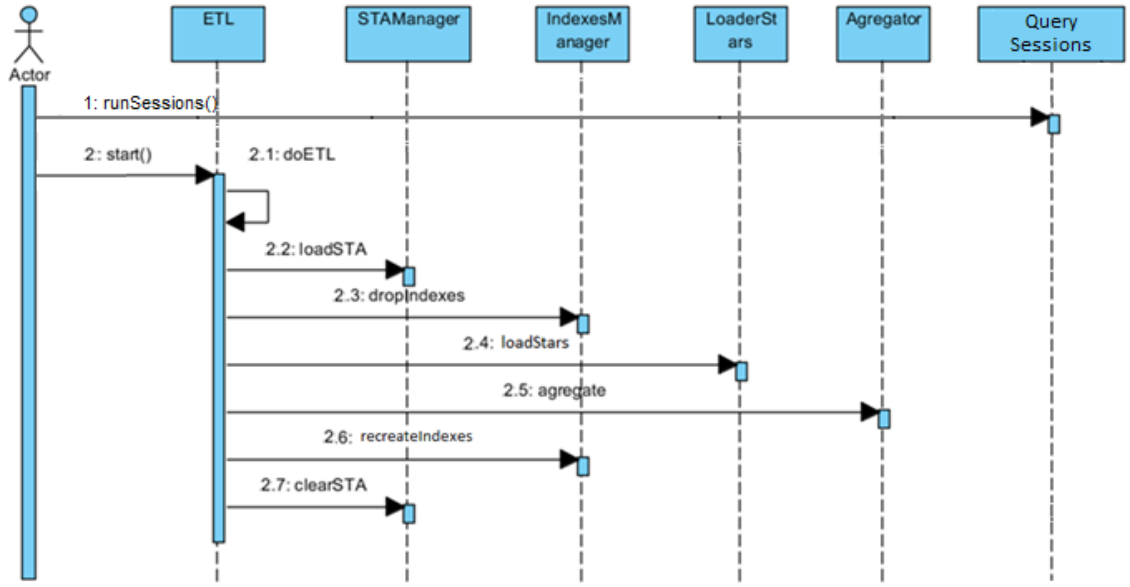


Figura 11: Diagrama de sequencia com os passos do processo ETL realizados pelo framework.

Este capítulo está estruturado da seguinte forma: a secção 4.1 apresenta a estrutura do DW. A secção 4.2 descreve as sessões de pesquisa e o QW. A secção 4.3 descreve os vários passos do processo ETL executado pelo Benchmark, e a secção 4.4

descreve as experiências e resultados que se pode obter com o SSB-RT. Finalizando o capítulo, a secção 4.11 descreve a forma de geração inicial dos dados para o SSB-RT.

4.1. Estrutura da *Data Warehouse*

Na base de dados em que o esquema do DW se encontra, foi utilizado um modelo desenvolvido abalizado nas informações contidas no projeto visto em (SSB), que segue o formato do modelo estrela, amplamente adotado em projetos de DW. Com o intuito de disponibilizar inúmeras vistas dos dados presentes no DW, foram criadas duas estrelas principais na base de dados: uma com base nas informações das vendas (*Orders*) por clientes (*Customer*) e tendo outras dimensões (*Date* e *Time*) ligadas a tabela de fato. A Figura 12 ilustra o modelo estrela desenvolvido para representação dos dados relacionados às vendas.

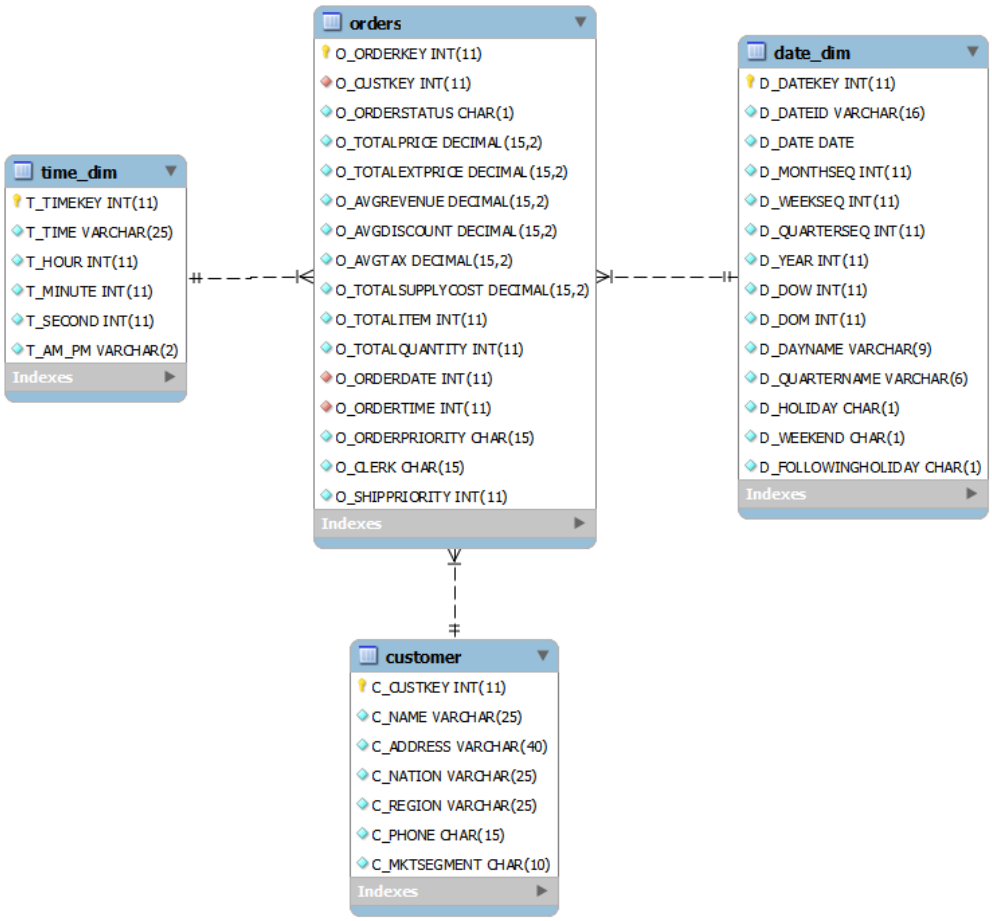


Figura 12: Esquema estrela que representa as vendas (*Orders*).

A segunda estrela foi instituída com o fito de representar as informações relativas aos produtos vendidos (itens das vendas), possuindo como tabela de fato "LINEORDER". A estrutura desta estrela é caracterizada com uma granularidade menor em relação à estrela apresentada anteriormente, possuindo várias outras tabelas de dimensões (*Customer*, *Supplier*, *Part*, *Date* e *Time*). Abaixo, a Figura 13 demonstra o esquema estrela para os dados relacionados aos itens das vendas.

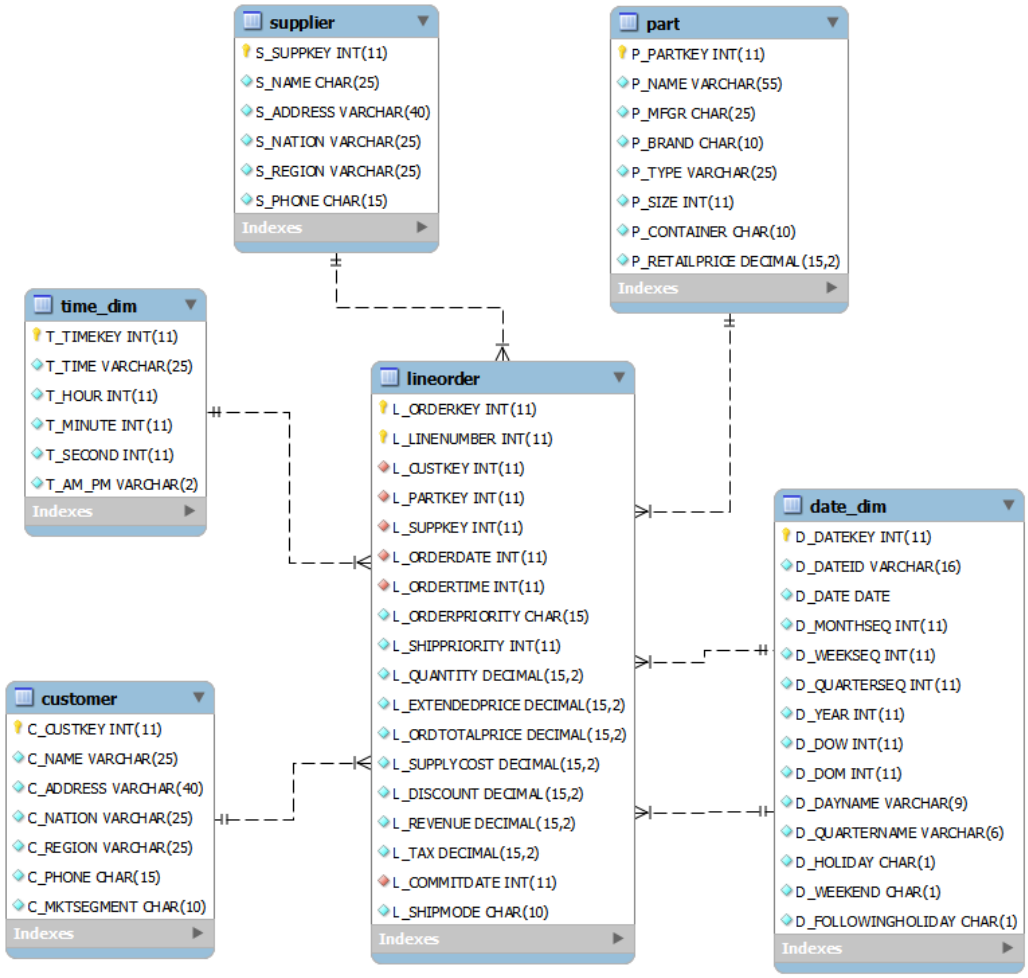


Figura 13: Esquema estrela que representa os itens das vendas (*LineOrder*).

Nas próximas secções serão apresentados os detalhes de cada uma das tabelas que compõem o modelo da base de dados.

4.1.1. Detalhamento das tabelas de fato

Nesta secção serão expostos pormenores referentes às tabelas de fato existentes na definição deste benchmark. Aqui são detalhadas todas as restrições, índices, chaves estrangeiras e primárias que existem nessas tabelas.

As tabelas de fato são os elementos principais de um esquema de base de dados em estrela. As tabelas possuem a grande maioria dos dados existentes numa DW. Como citado anteriormente, foram definidas duas tabelas de fato para a solução: Orders e Lineorder. Primeiramente, a tabela de fato Orders é detalhada. A Tabela 22, situada no Apêndice B, apresenta os atributos presentes na tabela e informações de chaves estrangeiras e primárias da tabela de fato Orders. As restrições (*constraints*) criadas para a tabela possuem a característica de garantir a integridade dos dados contidos na mesma. As restrições criadas para Orders são vistas na Tabela 1.

Restrição	**Tabela de Referência**	**Coluna(s)**
PRIMARY KEY	N/A	O_ORDERKEY
FOREIGN KEY CUSTKEY	Customer	C_CUSTKEY
FOREIGN KEY ORDERDATE	Date_dim	D_DATEKEY
FOREIGN KEY ORDERTIME	Time_dim	T_TIMEKEY

Tabela 1: Restrições da tabela Orders.

Além disso, foi designado um conjunto de índices, com o objetivo de otimizar o tempo de resposta das consultas e tornar o benchmark mais próximo da realidade, já que este recurso é muito utilizado em DW. Para esta tabela de fato (Orders) foi criado um total de 5 índices do tipo b-tree, que podem ser vistos na Tabela 2.

Índice	Coluna(s)	Tipo
PRIMARY KEY	O_ORDERKEY	b-tree
IN_O_CUSTKEY	O_CUSTKEY	b-tree
IN_O_ORDERDATE	O_ORDERDATE	b-tree
IN_O_ORDERTIME	O_ORDERTIME	b-tree
IN_O_ORDERDATETIME	O_ORDERDATE e O_ORDERTIME	b-tree

Tabela 2: Índices da tabela Orders.

A tabela de fato Lineorder possui uma menor granularidade (trata dos itens das encomendas). Portanto, a quantidade de dados contida na tabela é maior que na Orders. De forma semelhante à anterior, também foram criadas restrições, com o escopo de garantir a integridade dos dados. As restrições definidas para essa tabela podem ser vistas na Tabela 3.

Restrição	Tabela Referência	Coluna(s)
PRIMARY KEY	N/A	L_ORDERKEY e L_LINENUMBER
FOREIGN KEY CUSTKEY	Customer	C_CUSTKEY
FOREIGN KEY PARTKEY	Part	P_PARTKEY
FOREIGN KEY SUPPKEY	Supplier	S_SUPPKEY
FOREIGN KEY ORDERDATE	Date_dim	D_DATEKEY
FOREIGN KEY ORDERTIME	Time_dim	T_TIMEKEY

Tabela 3: Restrições da tabela Lineorder.

Além das restrições supracitadas, todas as colunas das tabelas de fato possuem uma restrição de integridade do tipo NOT NULL.

Como a quantidade de dados existente nesta tabela é maior, foi necessária a criação de mais índices do que na tabela Orders. Para esta tabela, foram estabelecidos índices dos tipos b-tree e bitmap, totalizando 13 índices, sendo 7 do tipo b-tree e os

outros 6 do tipo bitmap. Os índices criados para a tabela de fato Lineorder podem ser vistos na Tabela 4.

Restrição	**Coluna(s)**	**Tipo**
PRIMARY KEY	L_ORDERKEY e L_LINENUMBER	b-tree
IN_L_CUSTKEY	L_CUSTKEY	b-tree
IN_L_PARTKEY	L_PARTKEY	b-tree
IN_L_SUPPKEY	L_SUPPKEY	b-tree
IN_L_ORDERDATE	L_ORDERDATE	b-tree
IN_L_ORDERTIME	L_ORDERTIME	b-tree
IN_L_ORDERDATETIME	L_ORDERDATE e L_ORDERTIME	b-tree
BI_L_TAX	L_TAX	Bitmap
BI_L_DISCOUNT	L_DISCOUNT	Bitmap
BI_L_ORDERPRIORITY	L_ORDERPRIORITY	Bitmap
BI_L_SHIPPRIORITY	L_SHIPPRIORITY	Bitmap
BI_L_SHIPMODE	L_SHIPMODE	Bitmap
BI_L_QUANTITY	L_QUANTITY	Bitmap

Tabela 4: **Índices da tabela Lineorder.**

4.1.2. Detalhamento das tabelas de dimensão

Nesta secção as tabelas de dimensão definidas para este *benchmark* serão detalhadas, onde tais definições estão relacionadas com a forma de organização do modelo, à estrutura das tabelas, às restrições de integridade de dados, índices, entre outros.

As tabelas de dimensões representam as características de um dado evento. Por exemplo, os dados de um determinado cliente (nome, endereço, idade, etc.) se relaciona à tabela de fato através de uma chave estrangeira. Para o modelo adotado neste benchmark, foram definidas cinco tabelas de dimensão, cada uma contendo dados característicos. Essas dimensões podem ser vistas na Tabela 5.

Nome da tabela	Descrição
Customer	Todas as informações básicas dos clientes da empresa.
Supplier	Informações relacionadas aos fornecedores.
Part	Dados referentes aos produtos vendidos pela empresa.
Date_dim	Informações sobre as datas (Feriado, fim de semana, etc.).
Time_dim	Informações sobre o horário (ex. AM ou PM).

Tabela 5: **Tabelas de dimensão.**

Para estas tabelas só foram criadas uma restrição de chave primária em cada tabela. Não é necessária a criação de índices para as tabelas de dimensão. Maiores detalhes de cada atributo existente nas tabelas de dimensão podem ser vistos no Apêndice B.

4.1.3. Tabelas de dados agregados

O *benchmark* recomendado no trabalho sugere a criação de 12 sumários de dados agregados. Assim, após o carregamento, o *benchmark* irá agregar os dados existentes numa área de estágio. Em seguida, atualizar ou incluir os registros nas tabelas de agregação. Os atributos que são agrupados são os mesmos em todos estes sumários, variando apenas a chave primária que identifica os elementos únicos. Todos os atributos agregados são numéricos e foram aplicadas as seguintes funções de agrupamento: SUM, MIN, MAX, COUNT e AVG. Os atributos comuns a todos os sumários são detalhados na Tabela 6.

Campo	Descrição
L_QUANTITY	Agrupamento da quantidade de itens.
L_EXTENDEDPRICE	Agrupamento do valor dos itens.
L_SUPPLYCOST	Agrupamento do preço de custo do produto.
L_DISCOUNT	Agrupamento do desconto aplicado nos itens.
L_REVENUE	Agrupamento do valor vendido (aplicando desconto).
L_TAX	Agrupamento do valor das taxas.

Tabela 6: **Atributos agregados das vistas.**

Como dito anteriormente, todas as vistas possuem os atributos listados acima, mudando apenas as chaves primárias. Foram constituídos três grupos de vistas, de acordo com a perspectiva de análise (*Customer*, *Part e Supplier*), cada um gerando outras quatro vistas que representam diferentes níveis de granularidade (hora, dia, mês e ano), totalizando 12 vistas. Por exemplo, as vistas de *Customer* são:

1. *Customer* por hora: agregação é realizada levando em consideração todas as vendas por cliente em cada hora. Os identificadores dessa vista são: L_CUSTKEY, L_ORDERDATE e L_HOUR.
2. *Customer* por dia: agrupamento das informações por clientes realizadas em determinado dia. A chave primária para este sumário é: L_CUSTKEY, L_ORDERDATE.
3. *Customer* por mês: aumentando a granularidade, este sumário agrega as vendas por cliente em um determinado mês. Os identificadores para este sumário são: L_CUSTKEY, L_MONTH e L_YEAR.
4. *Customer* por ano: por último, e assumindo o maior grau de granularidade, o sumário que agrega as informações das vendas de clientes em um determinado ano. A chave primária para este sumário é: L_CUSTKEY e L_YEAR.

As demais vistas seguem o mesmo raciocínio, mudando apenas o atributo das tabelas de dimensões (SUPPKEY - Supplier e PARTKEY - Part).

A Figura 14 mostra um esquema das tabelas de dados agregados.

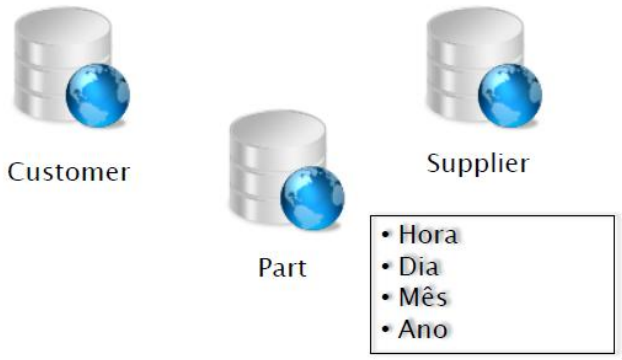


Figura 14: Modelo das Tabelas de dados agregados.

Com a utilização desta estrutura de tabelas agregadas, torna-se possível a realização de consultas, cujo resultado seja disponibilizado num espaço de tempo inferior, caso as consultas fossem executadas com base nos dados contidos nas tabelas de fato.

4.1.4. Tabelas auxiliares

Com a finalidade de auxiliar a computação dos dados agregados, foram instituídas duas tabelas auxiliares, uma para cada tabela de fato. A estrutura dessas tabelas são iguais as tabelas de fato, se tratando dos atributos. Essas tabelas foram denominadas Order_STA e Lineorder_STA.

Como essas tabelas possuem uma quantidade de dados relativamente pequena, apenas os dados utilizados na integração, não possuindo consultas que utilizam seus dados, não têm nenhum tipo de restrição ou índice, o que aumenta o desempenho no carregamento. No Apêndice B é possível ver os detalhes do layout dessas tabelas.

4.2. Query Workload e Query Sessions

Toda base de dados do tipo *Data Warehouse* deve prover uma série de consultas e análises para os utilizadores, para que possam tomar decisões mais eficazes. O *Query Workload* (QW) definido para este *benchmark* possui um conjunto de 13 consultas, que obtém os dados em diferentes perspectivas. O objetivo principal desse QW para o benchmark é a realização de uma carga de trabalho significativa, realizando tais consultas na DW.

O QW criado para o *benchmark* possui quatro consultas base. Essas consultas foram definidas com o propósito de avaliar os sistemas de DW usando pesquisas com diferentes níveis de seletividade. As consultas base são:

```
Q1. select sum(lo_extendedprice*lo_discount) as reve-nue
from lineorder, date
where lo_orderdate = d_datekey
and d_year = [YEAR] -- Specific values below
and lo_discount between [DISCOUNT] - 1
and [DISCOUNT] + 1 and lo_quantity < [QUANTITY];
```

A consulta base Q1 possui restrições apenas em uma das tabelas de dimensão. Esta consulta retorna o valor de rendimento do produto, depois de aplicar o desconto, para um determinado ano e entre um intervalo percentual de desconto. As outras consultas que derivam desta, alteram os valores para as variáveis apresentadas no exemplo acima e alguns predicados.

```
Q2. select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
and lo_partkey = p_partkey
and lo_suppkey = s_suppkey
and p_category = 'MFGR#12'
and s_region = 'AMERICA'
group by d_year, p_brand1
order by d_year, p_brand1;
```

Q2 já possui restrição em duas tabelas de dimensão. Esta consulta base já filtra os resultados com base em informações do fornecedor e da categoria do produto. Além disso, ela possui como resultado o valor do rendimento agrupado por ano e marca dos produtos.

```
Q3. select c_nation, s_nation, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey
and c_region = 'ASIA' and s_region = 'ASIA'
and d_year >= 1992 and d_year <= 1997
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;
```

A terceira consulta base (Q3) possui restrição em três tabelas de dimensão (Customer, Date_dim e Supplier) e agrupa o valor de rendimento (lo_revenue) por: nação do fornecedor, nação do cliente e ano.

```
Q4. select d_year, c_nation, sum(lo_revenue - lo_supplycost) as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and lo_partkey = p_partkey
and lo_orderdate = d_datekey
and c_region = 'AMERICA'
and s_region = 'AMERICA'
and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, c_nation
order by d_year, c_nation;
```

A consulta base Q4 possui algumas variações relacionadas aos resultados das consultas e seus predicados, mas no geral, possui no mínimo três restrições em tabelas de dimensões e retorna o agrupamento do lucro (lo_revenue - lo_supplycost) por pelo menos duas dimensões.

As consultas existentes, e detalhes relacionados a cada uma, podem ser vistos no Apêndice B.

Tendo em conta que os sistemas a testar são acessados por múltiplas sessões simultâneas, o *benchmark* simula acessos de vários utilizadores que fazem uso da plataforma. Neste caso, o *benchmark* dispara uma quantidade de *threads*, definida por configuração na ferramenta, a executar consultas de forma aleatória e simultânea. Cada *thread* efetua uma consulta por vez. Com base nessa simulação, é possível avaliar o desempenho de um sistema sob certa carga de *stress*.

Parâmetros de configuração

Parâmetro	Valores	Descrição
nSimultaneosSessions	Valor numérico inteiro	Valor informativo da quantidade de sessões de utilizadores que o *benchmark* criará.

Tabela 7: **Parâmetros de configuração para *query sessions*.**

4.3. Processo de Integração dos dados

O *benchmark* SSB-RT inclui simulação do processo de integração de dados. Nesta secção será discutido como o benchmark representa e simula os componentes desse processo, começando pelos dados de origem, que foram extraídos das fontes de dados (visto na secção 4.3.1), passando pelos processos de transformação (visto na secção 4.3.2), eliminação dos índices (secção 4.3.3), carregamento dos dados (secção 4.3.5), recriação dos índices (secção 4.3.6) e, por fim, a limpeza da área de estágio de refresh (secção 4.3.7). A Figura 15 mostra o fluxo dos passos que fazem parte do processo ETLR.

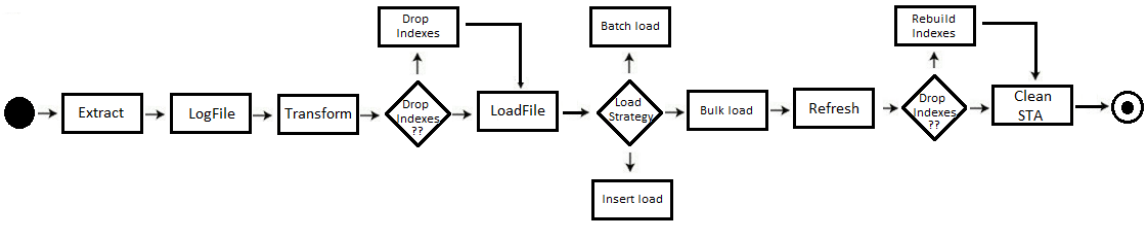


Figura 15: Fluxograma do processo ETLR.

4.3.1. Fonte de dados

As fontes de dados (*Data Sources* - DS) utilizadas tanto na arquitetura tradicional de um DW quanto numa RTDW, fornecem os dados que serão transformados e depois carregados na base de dados OLAP. Estas fontes podem ser de diversos tipos: bases de dados transacionais, ficheiros XML, ficheiros de *log*, entre outros.

Em razão do SSB reter um esquema derivado do TPC-H, o benchmark simula os dados extraídos das fontes usando dados gerados pelo DBGEN, a aplicação de geração de dados que acompanha o TPC-H. São gerados ficheiros de *log* que simulam os dados, depois da extração, incluindo a informação de cada encomenda (ORDER) e seus itens (LINEITEM). O tamanho e frequência de geração destes *logs* são configuráveis para testar batches de tamanhos variáveis.

Cada venda (ORDER) possui os itens (LINEITEM) relacionados a ela. Os campos existentes nas linhas do ficheiro são apresentados, em detalhes, no Apêndice B. A Figura 16 mostra o esquema da sequência de linhas existente nos ficheiros.

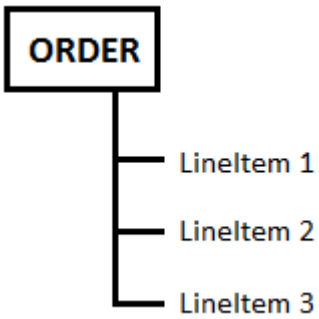


Figura 16: Esquema do ficheiro gerado através do DBGEN alterado.

Os dados gerados pela aplicação seguem o seguinte formato:

ORDER|1|370|O|172799.49|1996-01-02|5-LOW|Clerk#000000951|0|nstructions sleep furiously among |
LINEITEM|1|1552|93|1|17|24710.35|0.04|0.02|N|O|1996-03-13|1996-02-12|1996-03-22|DELIVER IN PERSON|TRUCK|egular courts above the|
LINEITEM|1|674|75|2|36|56688.12|0.09|0.06|N|O|1996-04-12|1996-02-28|1996-04-20|TAKE BACK RETURN|MAIL|ly final dependencies: slyly bold |

A Figura 17 representa um esquema conceitual correspondente aos dados apresentados acima.

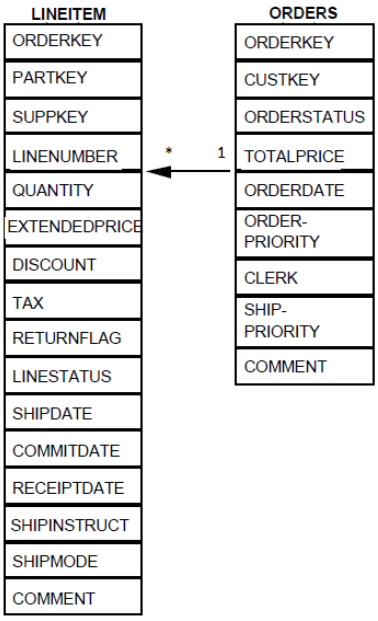


Figura 17: Modelo conceitual dos dados do ficheiro LogFile.

Parâmetros de configuração

Parâmetro	Valores	Descrição
logFileType	• Memory • Disk	Parâmetro que indica o tipo de armazenamento do *log* (memória ou disco).
logFileSize	Valor inteiro maior que zero	Tamanho medido em linhas para o LogFile que será lido pelo *benchmark*.
logFileDecision	• Size • Rate • Both	Este parâmetro indica qual a estratégia de entrada de dados do *benchmark*. Onde "Size" indica que o LogFile terá um tamanho definido, "Rate" informa que o benchmark recebe uma taxa de dados de entrada (ex. 1000 linhas/segundo) e executa o *benchmark* a cada período, ou "Both" que utiliza ambas as estratégias, se o LogFile não atingir o tamanho em determinado espaço de tempo o *benchmark* é executado mesmo assim.

Tabela 8: **Parâmetros de configuração para dados da fonte.**

4.3.2. Transformações

Para que os testes simulem um contexto do mundo real, o benchmark exercita um conjunto de transformações de uma *Data Warehouse*.

Os dados das fontes (secção 4.3.1) são lidos e em seguida realiza-se uma série de transformações a fim de adequar estas informações à estrutura do modelo estrela presente no DW. As transformações podem ser limpeza dos dados, *lookups*, aplicação de cálculos matemáticos, verificação de integridade, entre outros. As transformações definidas neste benchmark foram:

1. *Lookup* do custo de compra (*Supply Cost*): com base no identificador do produto (L_PARTKEY) e do fornecedor do produto (L_SUPPKEY), o *benchmark* busca o valor do custo do produto e atribui este valor à coluna correspondente da Lineorder (L_SUPPLYCOST).
2. Cálculo do lucro do produto (*Revenue*): aplica-se a fórmula (l_extendedprice*(100-l_discounnt))/100) e o resultado é atribuído à coluna referente ao valor na tabela Lineorder (L_REVENUE).
3. Realização do cálculo do total da venda: somatório do valor de todos os itens da encomenda e colocado na tabela Lineorder (L_ORDTOTALPRICE).
4. *Lookup* para a data da venda: com base na data (YYYY-MM-DD) existente no ficheiro, o benchmark deve buscar o identificador da tabela de dimensão Date_dim e atribuir esse valor ao campo da Lineorder (L_ORDERDATE).
5. *Lookup* para o horário da venda: operação semelhante a transformação 4, porém, o identificador que será recuperado pertence a tabela Time_dim e este valor será inserido na coluna L_ORDERTIME da tabela Lineorder.
6. *Lookup* para a data da entrega: Transformação semelhante ao item 4, seu valor é conferido à coluna L_COMMITDATE da tabela Lineorder.
7. Cálculo para o valor total estendido (*Extended Price*): soma de todos os valores estendidos (L_EXTENDEDPRICE) concernentes aos itens da encomenda e o resultado final é cominado à coluna referente da tabela Orders (O_TOTALEXTPRICE).
8. Cálculo para a média do valor líquido dos produtos da venda (*Revenue*): calcula-se a média deste valor entre todos os itens da venda e o resultado será inserido na tabela Orders (O_AVGREVENUE).
9. Cálculo para a média dos descontos dos produtos da venda: realiza-se a média dos descontos existentes nos itens da encomenda e o valor final é atribuído à coluna relativa da tabela Orders (O_AVGDISCOUNT).

10. Cálculo para a média dos impostos dos produtos da venda: média dos impostos dos itens e o valor final será inserido na tabela Orders (O_AVGTAX).
11. Cálculo para o total do custo dos produtos da venda (*Supply Cost*): soma do valor de custo de todos os itens da venda, atribuído à coluna O_TOTALSUPPLYCOST da tabela Orders.
12. Cálculo para o total das quantidades de produtos da venda: somatório das quantidades dos itens de determinada venda (O_TOTALQUANTITY).
13. Cálculo para quantidade de itens da venda: possui a contagem dos itens da venda (O_TOTALITEM).
14. *Lookup* da data da venda: transformação semelhante ao item 4 (O_ORDERDATE).
15. *Lookup* do horário da venda: transformação semelhante ao item 5 (O_ORDERTIME).

Finalizadas as transformações, esses dados são inseridos num buffer, que guarda as informações até atingir um tamanho limite ou ler o arquivo por completo. Posteriormente, a ferramenta criará os *batches files* para o carregamento dos dados na DW (um para cada tabela de fato), ou então, introduz as informações diretamente na DW (por exemplo, utilizando o *driver* JDBC, como na implementação realizada no trabalho). A estratégia de carregamento depende da configuração realizada antes da execução do *benchmark*. Existem três estratégias definidas no SSB-RT: *bulk load*, *batch load* e *insert* tuplo a tuplo (vistas na secção 4.3). Um exemplo do formato dos dados transformados pode ser visto abaixo, em que a primeira linha é referente aos dados para a tabela Orders e a segunda para a Lineorder.

```
287834718|370|O|172799.49|156|96767.7144|36|17|518398.47|386|746|33656|72007|5-LOW|Clerk#000000951|0|
287834718|1|370|1552|93|33656|24710.35|0.02|17|24710.35|1255.8|0.04|1255.8|805.90|0.02|33656|TRUCK|72007|
```

Existe a possibilidade de configurar o tamanho do *buffer* de transformação, podendo ajustá-lo conforme a necessidade.

Parâmetros de configuração

Parâmetro	Valores	Descrição
LoadFileType	• Memory • Disk	Parâmetro que indica o tipo de armazenamento do log (memória ou disco).
LoadFileSize	Número inteiro maior que zero	Número indicativo do tamanho do loadFile.
indexesDecision	• Full • onlyIndexes	Decisão de criação dos índices, onde "Full" cria índices e chaves estrangeiras e "onlyIndexes" somente os índices.

Tabela 9: **Parâmetros de configuração para transformação.**

4.3.3. Eliminação de Índices

Outro fator que pode ser configurado neste *benchmark* é a possibilidade de eliminação dos índices, permitindo avaliar a performance do carregamento com ou sem a extinção dos índices e das restrições. Configurando a ferramenta como desejado, é possível avaliar ambos os casos. Se o utilizador optar pela eliminação, todos os índices e restrições definidos para este *benchmark* serão eliminados antes do carregamento e, em seguida, repostos. Isso é feito para aumentar o desempenho no momento de carregar os dados nas estrelas. Caso contrário, os índices e chaves serão mantidos e o carregamento será realizado de forma menos eficiente.

Parâmetros de configuração

Parâmetro	Valores	Descrição
DropIndexes	• true • false	Indica ao benchmark se os índices serão eliminados ou não.

Tabela 10: **Parâmetros de configuração para a eliminação dos índices.**

4.3.4. Carregamento dos dados

O método de carregamento de novos dados é um fator importante no processo de refrescamento de uma DW. Este processo é responsável pela inclusão dos dados nas tabelas do esquema do DW. Podem ser escolhidos três métodos de carregamento: *Bulk load, batch load e insert* tuplo a tuplo. Com o *bulk load* o *benchmark* cria um ficheiro CSV contendo os dados que serão carregados e, posteriormente, é chamado um

aplicativo nativo do motor de base de dados para inserir os dados abrangidos no ficheiro. O *batch load* utiliza o *driver* do motor de base de dados (por exemplo, JDBC) para armazenar os comandos de inserção (em memória). Ao fim do processamento faz uma inserção de *batch*. O *insert* tuplo a tuplo também usa um *driver*, porém, cada comando de inserção é executado no momento da leitura da linha no *buffer*. A configuração dos métodos que o processo ETLR usará é configurável na própria ferramenta do *benchmark*.

Primeiro os dados são carregados nas tabelas auxiliares e, em seguida, nas estrelas da DW. O processo de carregamento é simples, porém, possui um alto custo.

Parâmetros de configuração

Parâmetro	Valores	Descrição
CallLoader	String não nula	String com o nome do comando que chama a aplicação de carregamento.
LoadStrategy	• BulkLoad • BatchLoad • Insert	Indica a estratégia de carregamento que o *benchmark* irá aplicar.

Tabela 11: **Parâmetros de configuração para carregamento dos dados.**

4.3.5. Refrescamento de Dados agregados

Uma DW pode ter uma série de vistas materializadas ou sumários, dependendo da necessidade. Estas vistas necessitam ser refrescadas, de acordo com a chegada de novos dados na DW. Muitas vezes o custo desta operação é extremamente elevado. Este benchmark inclui o refrescamento dessas vistas.

A operação é realizada após o carregamento dos dados e utiliza o auxílio de duas tabelas auxiliares. Primeiramente, a ferramenta muda o nome da tabela que possui os dados agregados. Em seguida, cria a nova tabela que irá armazenar os novos dados já agregados juntamente com os antigos. Por último, realiza uma consulta com o comando de inclusão que faz a agrupamento dos dados antigos e recentes e inclui na nova tabela. O algoritmo, em pseudo-código, desta operação é apresentado abaixo.

```
Begin
  Rename AGR_CUST_HOUR to AGR_CUST_HOUR_TEMP;
  Create table AGR_CUST_HOUR as
    Select sum(l_quantity_sum), min(l_quantity_min), Max(l_quantity_max) ...
    From (
          Select sum(l_quantity), min(l_quantity), Max(l_quantity) ...
          From lineorder
          UNION ALL
          Select * from AGR_CUST_HOUR_TEMP
    )
   group by l_custkey, l_orderdate, l_hour;
   drop table AGR_CUST_HOUR_TEMP;
end;
```

4.3.6. Recriação de índices

Caso os índices tenham sido eliminados anteriormente ao carregamento, após o refrescamento dos dados agregados, todos os índices e restrições, que foram extintos no início do procedimento de integração dos dados, são recriados. Devido à grande quantidade de dados existente nas tabelas de fato da DW, essa tarefa é considerada pesada, refletindo num elevado custo de execução, além da demora para a conclusão.

4.3.7. Limpeza da área de estágio de Refresh

Com o término do processo de integração dos dados, é realizada a limpeza de todas as tabelas que fazem parte da área de estágio. Existem duas tabelas auxiliares que fazem compõem esta área de estágio, uma para cada tabela de fato (vistas na secção 4.1.4). A área de estágio é utilizada apenas para ajudar na computação das vistas materializadas ou sumários existentes, logo, após a integração dos dados, torna-se necessário que os mesmos sejam excluídos, deixando as tabelas totalmente vazias. Com isso, na próxima vez que o processo de integração de dados for executado, a agregação será realizada apenas para estes novos dados.

4.4. Experiências e resultados

Um benchmark é desenvolvido com o fito de avaliar sistemas com base em diversos testes. Este *benchmark* (SSB-RT) possui um conjunto de testes e configurações que podem criar diversos cenários, objetivando avaliar a performance de um sistema de DW com integração de dados em tempo real ou quase real. Esta secção retém detalhes de

todos os cenários de testes possíveis neste *benchmark*, as configurações necessárias para representar os cenários e como os resultados são medidos e apresentados.

Utilizando o SSB-RT é possível simular e avaliar as seguintes situações:

- Integração constante e em tempo quase real: com a realização deste teste é possível avaliar se o DW possui capacidades de integração de dados em tempo quase real. Neste caso, configura-se a ferramenta informando qual a periodicidade que o processo de ETL será realizado, bem como o tempo total que o teste será executado.
- Integração de dados com a eliminação ou não dos índices: com este cenário, é possível informar (através da configuração do teste no *benchmark*) se os índices e chaves estrangeiras serão eliminados, possibilitando avaliar o impacto que os recursos exercem no processo ETL.
- Integração *online* versus integração *offline*: a concorrência da integração de dados processados juntamente com consultas também é um fator importante e definido neste *benchmark*. Com este cenário de teste é possível avaliar o impacto que o processo ETL sofre quando executado em simultâneo com consultas executadas. Caso o utilizador do SSB-RT deseje avaliar sem efetuar consultas, basta atribuir o valor zero para o parâmetro nSimultaneousSessions.
- Integração de dados utilizando tamanhos variados do logFile: também é possível avaliar o comportamento do processo ETL definido no SSB-RT para *logs* com diversos tamanhos. Basta listar todos os ficheiros que deseja avaliar e montar um array com o nome dos mesmos e executar o *benchmark*.
- Avaliação das consultas do QW: de forma semelhante ao cenário de teste que avalia o ETL *offline* e *online*, este caso serve para avaliar o inverso, ou seja, impacto que o processo ETL exerce nas consultas. Com isso, é possível identificar tal impacto analisando o tempo de resposta das consultas ao correr em simultâneo (ou não) com o processo ETL.
- Carregamento de dados: como dito anteriormente, é possível avaliar as estratégias de carregamento de dados. Configura-se a ferramenta

informando qual o método de carregamento é preferível. Por padrão, a configuração deve ser *bulk load*.

- Integração de dados em paralelo com diferentes quantidades de sessões de utilizadores: caso o utilizador tenha a necessidade de avaliar a integração *online* e diversas sessões, as quais efetuam consultas, basta configurar o parâmetro nSimultaneousSessions com a quantidade de sessões que desejar.

Finalmente, com o desígnio de avaliar os resultados, são medidos os tempos de execução (que pode ser medido em milisegundos ou em segundos) e o *throughput* (linhas/segundo) de cada tarefa do ETLR (E+T, Drop Indexes, Load STA, Load Stars, Refresh, Rebuild Indexes e Clean STA). Com estes testes e resultados obtidos é provável avaliar a performance de sistemas DW com integrações de dados em tempo quase real.

Configurações para os casos de testes

Caso de Teste	Configuração
Integração tempo real a cada 5 segundos e sem tempo para finalizar	• period = 5000; • timeRun = -1;
Integração a cada 5 segundos e com eliminação dos índices	• period = 5000; • dropIndexes = true;
Integração a cada 15 segundos e sem eliminação dos índices	• period = 15000; • dropIndexes = false;
Integração a cada 15 segundos e online	• period = 15000; • nSimultaneosSessions = 1;
Integração a cada 5 segundos e offline	• period = 5000; • nSimultaneosSessions = 0;
Integração a cada 5 segundos e com vários logFiles	• period = 5000; • logFiles [] = {"1","2","3"};
Integração a cada 5 segundos e utilizando a estratégia de carregamento *bulk load*	• period = 5000; • loadStrategy = "BULK";
Integração a cada 25 segundos e em simultâneo com 10 sessões de utilizadores	• period = 25000; • nSimultaneosSessions = 10;

Tabela 12: **Parâmetros de configuração para carregamento dos dados.**

4.5. Geração inicial de dados para DW

Com a finalidade de simular um ambiente mais aproximado da realidade, é necessário carregar dados iniciais no DW para aquisição de dados anteriores. Para realizar a tarefa de criação de dados para um carregamento inicial foi desenvolvido uma ferramenta que, com o auxílio do DBGEN do TPC-H (TPC-H, 2012), cria ficheiros com dados no formato das estrelas presentes na definição do SSB-RT.

Inicialmente, essa ferramenta de geração executa o DBGEN, gerando os ficheiros do TPC-H (.tbl). Após a geração desses ficheiros, os dados passam por uma série de transformações, que os adequam para o formato desejado, armazenando-os em ficheiros com a extensão "ssbtbl". O tamanho dos dados pode ser definido pelo utilizador, que informa o *Scale Factor* (SF) que desejar.

Posteriormente a geração, o utilizador deverá apenas carregar os ficheiros empregando a aplicação disponibilizada pelo fabricante do motor de base de dados que esteja usando, por exemplo o SQL *Loader* da Oracle.

O *benchmark* SSB-RT também dispõe de outra ferramenta para criação inicial da estrutura da base de dados. Esta aplicação possui todos os scripts das tabelas, chaves estrangeiras e primárias, índices, restrições e vistas (materializadas ou não) definidas para o *benchmark.*

Capítulo 5
Avaliação da arquitetura proposta VS tradicional

Neste capítulo foram realizados diversos experimentos, com o objetivo de comprovar que a arquitetura típica de uma *Data Warehouse* não é indicada para um contexto de tempo real. Além disso, também foram feitos testes para avaliar o desempenho da arquitetura proposta no trabalho, bem como as diversas formas de efetuar a junção dos resultados das consultas executadas na D-DW e na S-DW, quando necessário.

5.1. Avaliação dos fatores que influenciam as capacidades real-time de uma DW tradicional

Nesta secção serão apresentados detalhes dos resultados obtidos nos testes, sob uma arquitetura tradicional de DW, com integração contínua (tempo quase real) de dados, bem como a identificação e análise dos fatores que influenciam a performance.

5.1.1. Impacto de Load e Refresh Online no tempo de resposta das pesquisas

Quando se pretende que uma DW funcione em tempo quase real, pode-se avaliar influência de carregamento (L) e refrescamento (R) online. Com esta finalidade, foram avaliados o impacto das operações de carregamento e atualização de dados no tempo de resposta de cada consulta. A tarefa (L ou R) foi executada constantemente (em ciclo) e cada consulta do *Query Workload* (13 consultas) a correr 15 vezes, para obtenção de tempos de execução estatisticamente fiáveis. Foram descartados os 5 resultados mais distantes da média e calculado a média final. A Figura 18 mostra os resultados obtidos, em que é visível um impacto significativo no tempo de respostas das consultas. Vale salientar que, o desvio padrão destes resultados foi sempre inferior a 5%.

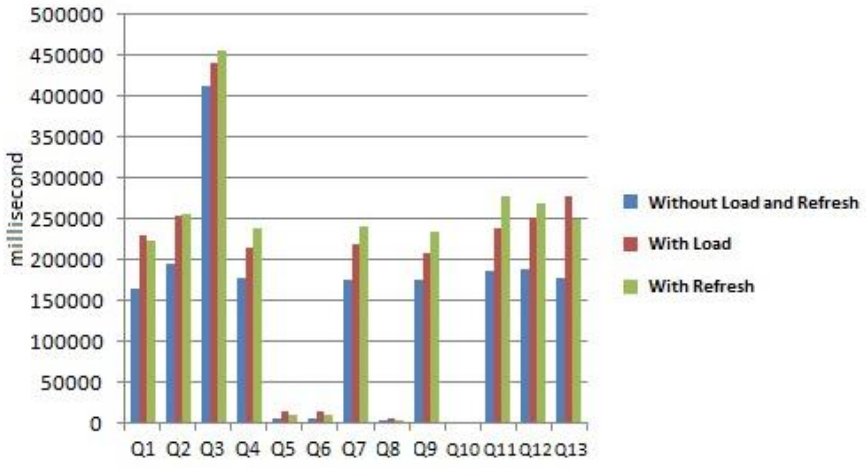


Figura 18: Gráfico com resultados do impacto do load e refresh no tempo de resposta das pesquisas.

Com a análise dos resultados obtidos através dos testes, foi possível concluir que as consultas sofrem uma perda de eficiência, em média de 44% ao executar em concorrência com o processo de carregamento (mínimo de perda de eficiência de 7% na consulta Q3 e máximo de 111% na pesquisa Q5). Já a diminuição da performance com o refrescamento foi em média de 35% (mínimo de 8% na consulta Q10 e máximo de 51% na consulta Q8).

5.1.2. Análise do throughput total do ETLR (Online x Offline)

Alguns dos fatores que influenciam o desempenho do processo ETLR é o tamanho do *log* a ser carregado e as consultas executadas em paralelo. A análise realizada nesta secção apresentará por gráfico a evolução do *throughput* (linhas/seg) ao incrementar o tamanho do *log*, bem como os índices e consultas afetam esse parâmetro.

Os testes aqui efetuados foram configurados para três cenários diferentes: I) *Offline* com índices e chaves; II) *Offline* recriando os índices e chaves; III) *Online* com 10 sessões realizando consultas em simultâneo.

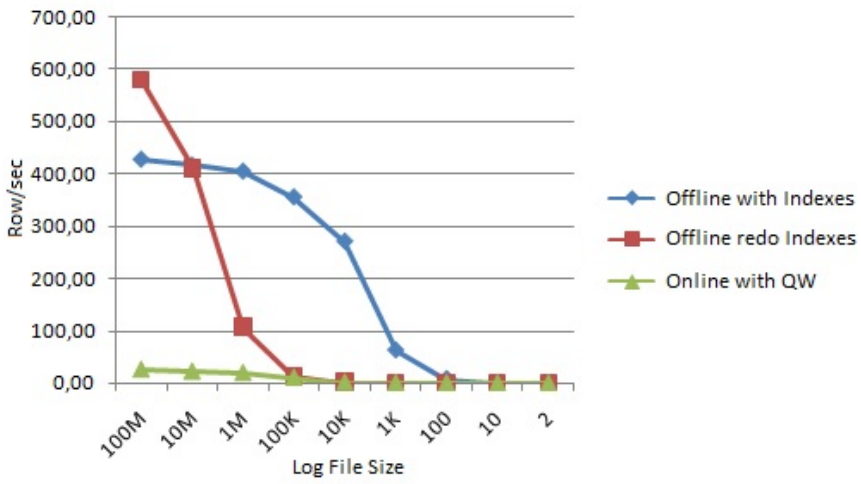


Figura 19: Gráfico da evolução do throughput do ETLR.

Analisando o gráfico, vislumbra-se que no cenário *offline* com os índices mantidos, o *throughput* é um pouco afetado, em razão do carregamento dos dados ser mais lento, porém, possui um bom resultado, não sendo melhor, unicamente, na situação de carregar 100M de linhas, em que destacou-se o *offline* com a recriação dos índices. Já o cenário *offline* com recriação de índices é bastante prejudicado pelo alto tempo de recriação dos mesmos, além das chaves estrangeiras, tornando-se viável apenas para uma grande quantidade de dados (acima dos 10M). Os resultados insatisfatórios obtidos foram no cenário *online* com consultas a correr simultaneamente. As consultas geraram um *overhead*, principalmente, no carregamento e no refrescamento dos dados. Esse *overhead* é enorme, traduzindo-se numa diminuição drástica do *throughput* de carregamento, revelando um problema muito relevante com abordagens *near real-time* de DW tradicionais. Além disso, tal *overhead* pode aumentar, de acordo com o número de sessões a correr ao mesmo tempo (detalhado na secção 5.3).

5.1.3. Impacto das sessões no desempenho do processo de atualização de uma DW

Uma base de dados DW recebe milhares de requisições de consultas realizadas pelos utilizadores constantemente. Essas consultas afetam o desempenho no processo ETLR, como visualizado anteriormente. Nesta secção é testado o impacto de diferentes quantidades de sessões executando consultas simultaneamente.

A configuração deste teste foi bastante simples. O benchmark foi executado para carregar as informações do *log*, contendo apenas 10 linhas, com a execução de consultas em paralelo. Inicialmente, não foi colocada nenhuma sessão, com o fito de obtermos os

tempos do ETLR sem qualquer influência de consultas. Posteriormente, foram adicionadas sessões, incrementando o número ao final de cada carregamento. A quantidade de sessões testadas foram: 5, 10, 50 e 100. Os resultados gerados no tempo total de duração do carregamento deste ficheiro pode ser visto na Figura 20.

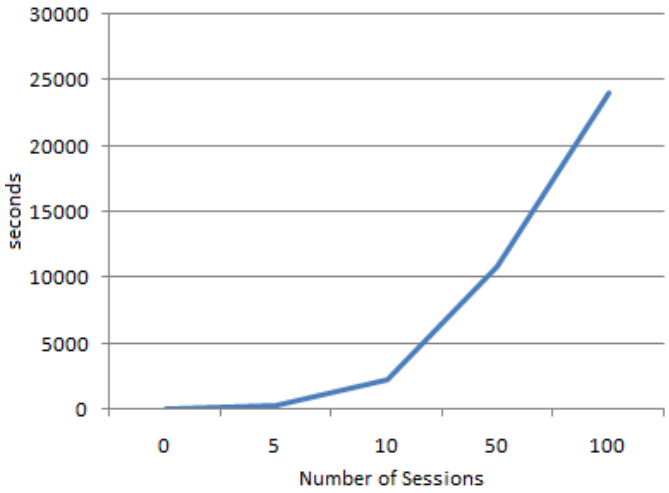


Figura 20: Gráfico de avaliação do impacto de sessões simultâneas a concorrer com o processo ETLR.

O impacto gerado pelas consultas na atualização de apenas 10 linhas foi negativo. Sem consultas a correr, a atualização dessas informações foi realizada em 12 segundos, ao colocar 5 sessões elevou-se para 291 segundos, um aumento de aproximadamente 2425%.

5.1.4. Impacto da estratégia de carregamento no processo ETLR (Offline)

Numa DW de tempo quase real a relação entre os tamanhos dos ficheiros de carregamento (*mini-batches*) e a performance assume particular importância, bem como a forma de carregar quando o tamanho do *batch* é pequeno ou grande (*bulk load*, *batch jdbc* e insert um a um). O objetivo desta análise é identificar a melhor estratégia de *load*, com o desígnio de otimizar o processo de ETLR na tentativa de atingir *near real-time* numa arquitetura de DW tradicional.

Foram colocadas em teste 3 estratégias: *Bulk load*, *Batch JDBC* e *Insert* um a um através do JDBC. Para este cenário de teste, o ficheiro a carregar tinha 10 milhões de linhas. Os resultados podem ser vistos na Figura 21.A e medidos em linhas/seg. Ademais, foi realizado um comparativo entre os métodos de *bulk load e batch jdbc* por várias quantidades de linhas. Neste caso, os testes foram realizados colocando o

processo de ETLR a correr sem eliminar os índices das tabelas de fato, além da ausência de consultas em simultâneo. A Figura 21.B mostra como o *throughput* total de carregamento varia com o tamanho dos dados a carregar.

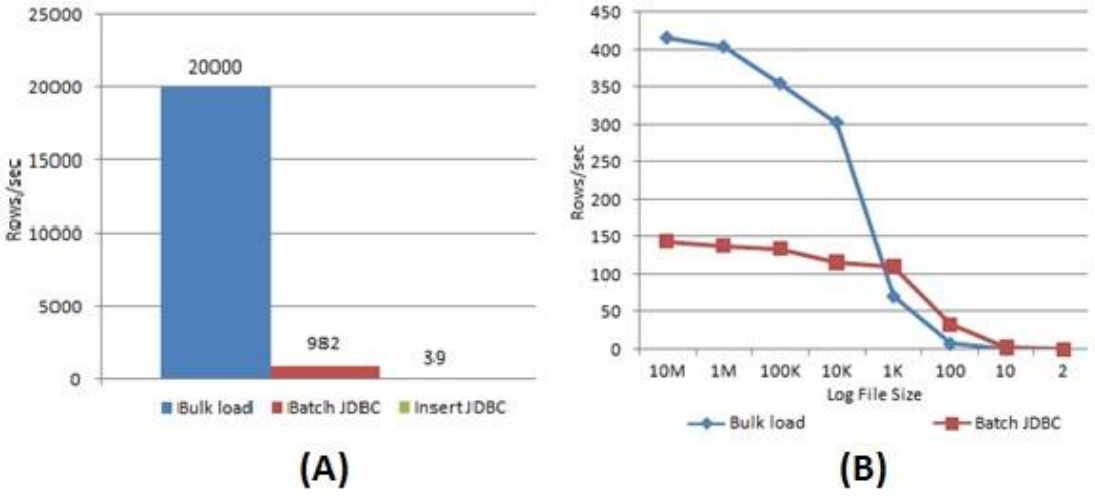


Figura 21: Gráfico (A) ilustra o throughput de cada estratégia de carregamento testada. Enquanto o gráfico (B) mostra o throughput total do processo de ETLR utilizando os métodos bulk load e batch jdbc em vários tamanhos de log.

Com base nos testes, foi possível avaliar que o carregamento realizado através do método *bulk load* é o mais indicado para um ficheiro que contenha muitas linhas. Com o auxílio do gráfico apresentado na Figura 21.B conclui-se que, em geral, a estratégia de *bulk load* realmente é a melhor opção para se utilizar quando se trata de ficheiros de log com número de linhas superior a 1000, obtendo *throughput* muito acima do outro método. Abaixo deste número de linhas, a estratégia de *batch jdbc* obteve uma ligeira vantagem.

5.1.5. Análise do custo de cada atividade do ETLR (Offline)

Nesta secção pretende-se analisar o custo de cada passo que é necessário no processo ETLR. As partes são: transformação e criação de ficheiro de carregamento (E + T); carregamento (que inclui o carregamento das tabelas auxiliares e carregamento das estrelas); refrescamento dos dados agregados e recriação dos índices, caso se tenha eliminado os índices para o carregamento mais eficiente. No carregamento das estrelas, consideramos os seguintes casos: sem índices; com índices e sem chaves estrangeiras; com índices e com chaves estrangeiras.

Os testes efetivados nesta fase possuem como propósito principal avaliar o impacto dos seguintes fatores no processo de ETLR:

- Recriação de índices: é feita a análise do impacto da recriação dos índices nos cenários de teste que realizam este procedimento.
- Chaves estrangeiras: Semelhante ao fator anterior, foi testado o quanto a recriação das chaves estrangeiras afetam a performance.
- Carregamento com e sem índices e/ou chaves estrangeiras: foi avaliado o tempo de inserção dos dados sem os índices e chaves estrangeiras, somente com índices e com os índices e chaves estrangeiras.
- Tamanho do ficheiro de *log*: foram realizados testes utilizando diversos tamanhos de ficheiros de origem para avaliar o comportamento com o crescimento destes ficheiros.

Inicialmente, foi realizado um teste para avaliar, de forma simples, os fatores que são mais significantes para estudo. Logo, foi escolhido um ficheiro de *log* grande (10M) e colocado o *benchmark* a correr para obter os tempos de cada um dos casos a se avaliar. Os resultados podem ser vistos na Figura 22.B, onde os gráfico mostram os tempos que cada um dos fatores levou para ser concluído com sucesso. A Figura 22.A foca nas operações de carregamento.

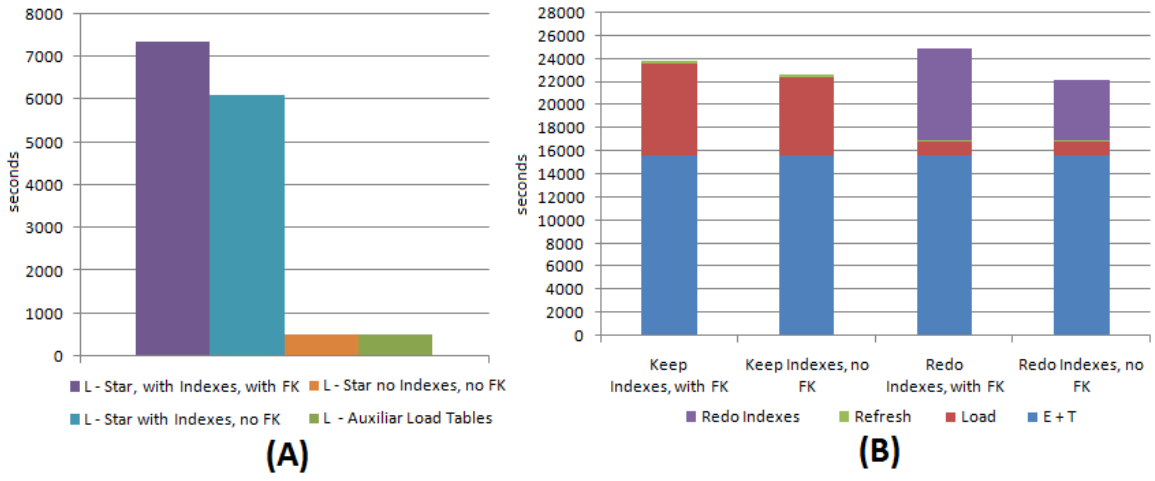


Figura 22: **O gráfico (A) mostra os tempos de carregamento em diversos cenários variando a configuração de manter os índices e chaves estrangeiras.Em (B), o gráfico mostra o tempo do processo ETLR completo variando os mesmos cenários vistos em (A).**

Com base nos resultados, pode ser visto que os principais fatores que influenciam a performance do processo ETLR são: a extração e transformação dos dados (E + T); o

carregamento das informações (L) quando realizada com os índices; a recriação dos índices. Vislumbra-se também, que as chaves estrangeiras influenciam de forma significativa a performance, tanto no carregamento dos dados, quanto na recriação dos índices (índices + chaves). Quando as chaves estrangeiras são retiradas, o carregamento dos dados nas estrelas demonstrou-se mais rápido aproximadamente 1000 segundos (16 minutos). O mesmo aconteceu ao tempo de recriação dos índices, ao retirar as chaves estrangeiras desta operação, o custo foi reduzido cerca de 2700 segundos (45 minutos).

5.1.6. Particionamento da tabela para obter near real-time com períodos Offline

Após a avaliação e utilizando uma arquitetura tradicional de DW, não foram obtidos bons resultados aplicando a um contexto de tempo quase real e *online*, ou seja, com consultas a correr, foi estudada a possibilidade de utilizar esta mesma arquitetura em tempo real (ou quase). Porém, assumindo que o carregamento de novos dados serão realizados em períodos *offline*, ou seja, a DW ficará sem acessos até que o processo de ETLR seja completamente finalizado.

Para garantir o sucesso destas experiências, foi necessário configurar o motor da base de dados para realizar o particionamento das tabelas de fato em pedaços pequenos. Definiu-se que, o tamanho de cada partição assumiria o tamanho de 1Gb. Após essa configuração, o benchmark foi posto a carregar um "*batch file*" de 10M de linhas, com o objetivo de comparar os resultados obtidos para a base de dados com partições do tamanho de 30Gb. Foram testados apenas 2 cenários nessa fase: I) Recriando os índices e chaves estrangeiras; II) Recriando apenas os índices, sem chaves estrangeiras. Os resultados podem ser vistos nos gráficos apresentados na Figura 23.

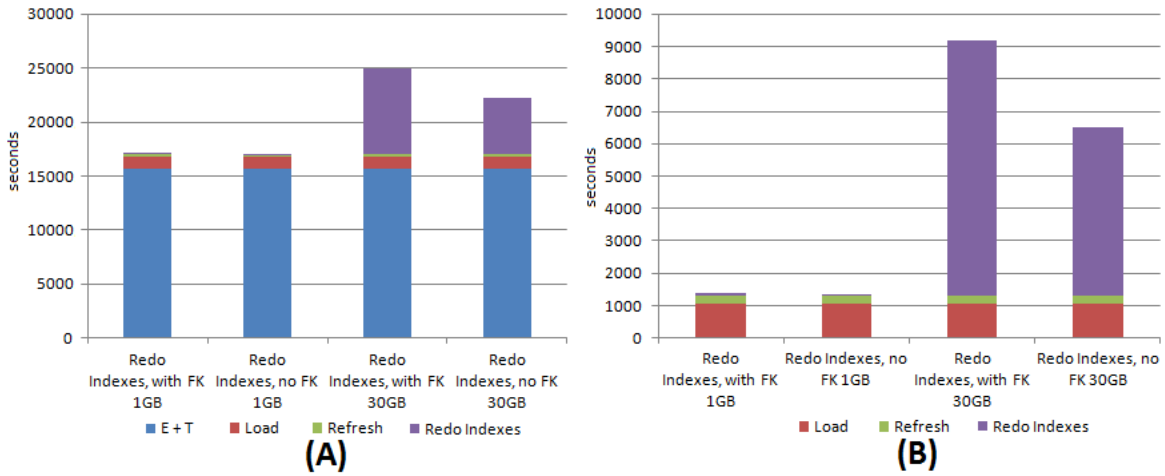


Figura 23: **O gráfico (A) mostra os tempos do ETLR para uma base de dados com partições de 30Gb e 1Gb, variando a configuração de manter os índices e chaves estrangeiras.Em (B), o gráfico mostra o tempo do processo LR (Load e Refresh) variando os mesmos cenários vistos em (A).**

Analisando a Figura 23.A, nota-se que o tempo gasto com a recriação dos índices e chaves estrangeiras na partição com 1Gb foi insignificante, comparado a recriação dos índices de uma partição com 30Gb. Se o tempo de ET for descartado, é possível visualizar que o grande problema da recriação dos índices foi reduzido absurdamente. Logo, se assumir um ficheiro de *log* menor, por exemplo, 100K linhas, é possível que o processo ETLR seja realizado em, aproximadamente, 4 minutos. Se assumir também, que o ET é realizado em outra máquina e não interfere a performance das consultas e vice-versa, é possível descartá-lo. Assim, pode-se concluir que o tempo que a DW necessita ficar *offline* para realizar a carga de batches com 100K de linhas é de, aproximadamente, 1,5 minutos. A Tabela 13 mostra os tempos obtidos no carregamento do *log* com 100k linhas.

Tempos para carregar 100k linhas (em segundos)	
E + T	156
Load	10
Refresh	4
Recriação de Índices	79
TOTAL	**249**

Tabela 13: **Tempos para carregamento de log contento 100k linhas.**

5.2. Avaliação da proposta de arquitetura RTDW

Para a execução dos testes, foi necessária a utilização de três computadores: um para a execução da aplicação Java, que faz a leitura do *log* e as transformações; e os outros que hospedarão as instâncias dos motores de base de dados (S-DW e D-DW). O computador utilizado para o aplicativo foi um Intel(R) Core(TM) i7 1,80GHz e 8GB de memória RAM. Para hospedar o D-DW, foi utilizado um computador com o processador Intel(R) Core(TM) 2 Quad 2,5GHz e 4GB de memória RAM. O servidor de base de dados S-DW é um Intel(R) Core(TM) i5 3,40GHz e com 16GB de memória RAM. O motor de base de dados escolhido foi o Oracle na versão 11g.

O segundo passo desta etapa experimental foi definir os cenários de testes que comprovem que a arquitetura sugerida é uma boa solução de RTDW. Foram listados 4 cenários, para os quais comparamos um DW tradicional com a nossa proposta de RTDW:

1. Consultas VS Consultas + L: neste cenário de teste iremos avaliar a performance das consultas a correr isoladamente e, posteriormente, verificarmos se a operação de carregamento (L) na D-DW gera alguma interferência quando colocada a correr de forma simultânea.
2. ETLR VS ETLR com Consultas: este cenário aborda testes que identificarão qual a influência que o processo de ETLR sofre ao correr em simultâneo com consultas, ou seja, em modo *online.*
3. ETLR com várias sessões: com base neste cenário de teste, será possível analisarmos o impacto que as sessões, que simulam acessos de utilizadores a correr consultas e análises na DW, geram sob o processo ETLR, quando postas a correr em simultâneo. Decidimos realizar este teste com diferentes quantidades de sessões: 0, 5, 10, 50 e 100 sessões a correr em paralelo.
4. Estratégias de *Merge*: este cenário foi pensado com o propósito de avaliar o desempenho de cada estratégia de *Merge,* indicando o impacto do carregamento de dados em cada um dos métodos. Com base nos resultados obtidos nestes testes, será possível identificar a melhor estratégia de *merge*.

Com o objetivo de obtermos dados estatísticos mais fiáveis, colocamos a correr por 15 vezes. Assim, calculamos a média geral do tempo de resposta dessa consulta e descartamos os 5 resultados mais distantes dessa média, calculamos, novamente, a média final do tempo de execução da pesquisa. Logo em seguida, calculamos o desvio padrão

nessa amostra de 10 tempos de resposta. Consideramos a seguinte fórmula para o cálculo do desvio padrão (S):

$$s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1}}$$

5.2.1. Análise nas estratégias de merge

Como visto anteriormente, o mecanismo responsável pela junção dos dados antigos com os dados recentes é chamado *Merger*. Foi comentado, também, que foram definidos diversos métodos e cenários de realização desta operação. Nesta secção foi feita a análise e identificação da melhor forma de realizar o *merge* das informações, bem como o estudo do impacto que a operação de carregamento exerce sobre a consulta que utiliza este mecanismo.

Para realização dos testes, foi escolhida uma consulta de forma aleatória entre o conjunto de pesquisas que não usavam datas na lista de filtros. A consulta escolhida foi a Q11. Com a realização destes testes, o desvio padrão máximo obtido foi de 5%.

Após obtermos as médias finais e analisar o desvio padrão, foi construído um gráfico, apresentado na Figura 24, para estudar os resultados obtidos.

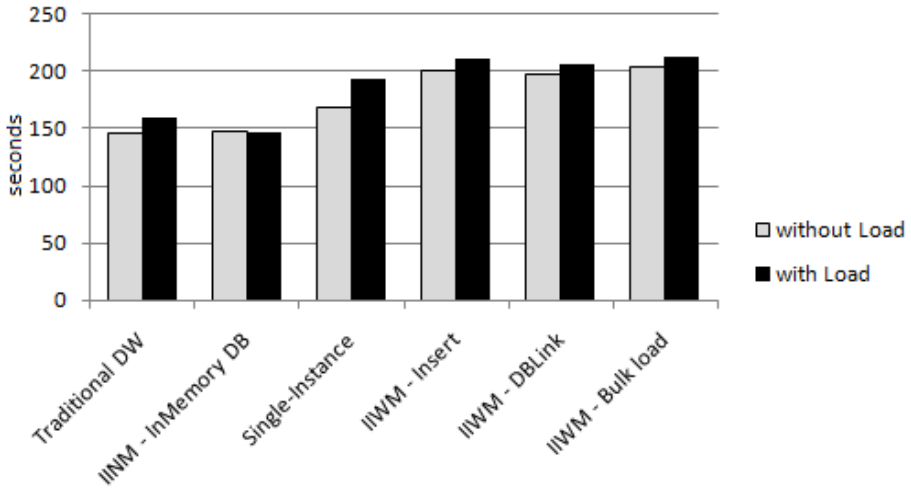


Figura 24: **Gráfico do tempo de resposta da consulta 11 utilizando os diversos métodos de junção dos resultados.**

Neste gráfico foram definidas duas formas de executar as consultas: com (*with Load*) e sem (*without Load*) carregamentos a correr em simultâneo. Além disso, foram testadas seis estratégias de *Merger*, que são:

1. *Traditional DW*: neste caso as consultas foram postas a correr apenas na S-DW, com o objetivo de simular as consultas a correr numa DW tradicional.
2. *Independent-instance, no migration* (IINM - *In memory* DB): para o caso deste cenário de testes, foi utilizado um terceiro componente responsável pela junção dos resultados. Neste caso, o motor de base de dados em memória H2. Primeiro as consultas são executadas, de forma paralela, nos dois componentes. Posteriormente, o resultado é inserido em tabelas criadas no H2, onde iremos executar uma consulta realizando a junção dos resultados.
3. *Single-instance*: aqui assumimos que os dois componentes (S-DW e D-DW) estão presentes na mesma instância da base de dados (mesma máquina).
4. *Independent-instance with migration* (IIWM - *Insert*): primeiro a consulta é realizada no D-DW e o seu resultado é migrado para uma tabela temporária no S-DW, onde corremos uma outra consulta que realiza a junção dos resultados. A inserção dos dados nessa tabela temporária é realizada através da execução de comandos insert.
5. *Independent-instance with migration* (IIWM - DBLink): para este cenário, utilizamos o recurso DBLink (Oracle) para realizar a migração dos dados de forma transparente e avaliamos o seu desempenho.
6. *Independent-instance with migration* (IIWM - *Bulk load*): a estratégia é a mesma do item 4, porém, a inserção é feita através de *bulk load*.

É possível analisar qual o melhor método testado. O primeiro (*Traditional* DW), serve como base para comparações, já que é a consulta a ser executada apenas no S-DW, sem realizar nenhuma junção. O *Merger* que obteve o melhor resultado foi o IINM - *In Memory* DB que utiliza um motor de base de dados em memória. Este método obteve o resultado igual ao de quando corremos a consulta somente no S-DW (*Traditional* DW). Comparando com os outros métodos, ele mais uma vez mostrou-se mais rápido (entre 20,8% e 49,3%) do que os demais métodos testados.

Quando analisamos o resultado da pesquisa a correr em simultâneo com o carregamento de dados, podemos ver que o desempenho do IINM - *In Memory* DB, ainda é o mais satisfatório, com uma perda na performance de apenas 1,38%.

5.2.2. Impacto do Load nas consultas

Com a identificação da melhor estratégia para realizar a junção dos resultados, iremos estudar o impacto do carregamento sobre o restante das consultas a correr em simultâneo. Mais uma vez, o desvio padrão obtido nesses testes foi muito pequeno, chegando no máximo de 9%. A Figura 25 apresenta o gráfico com os resultados obtidos.

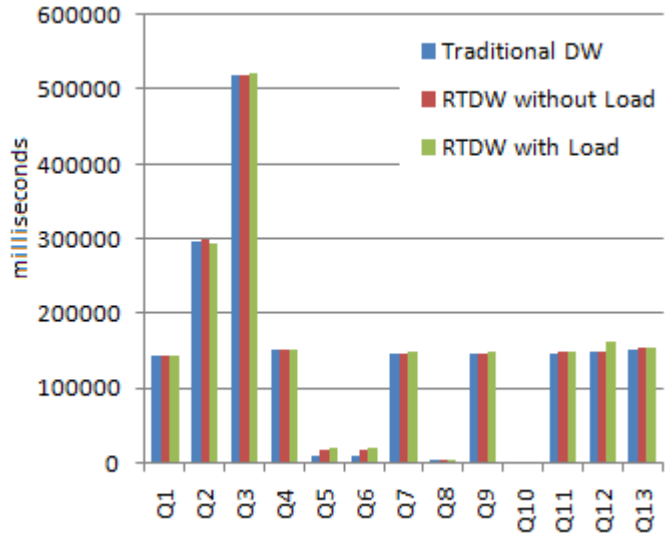


Figura 25: **Gráfico dos tempos de resposta das consultas.**

Nota-se que, o impacto tanto do merge, quanto do carregamento de dados é mínimo. Em 10 pesquisas, o impacto foi no máximo de 2%. Nas outras 3 pesquisas do QW, em que o tempo de resposta é muito pequeno, encontramos um impacto maior, porém, por se tratar de pesquisas rápidas (tempo de resposta em segundos ou até em milisegundos), não se torna um problema grave. Em outros termos, nossa solução permite a integração constante de dados (online), sem que as consultas sejam prejudicadas.

5.2.3. Impacto das consultas no processo ETLR

Nesta secção foi estudado como a execução de consultas prejudica o processo de ETLR, como o impacto na taxa de carregamento de dados.

Inicialmente, foi colocado o processo ETLR a correr, com diferentes tamanhos de *mini-batches*, e 10 sessões a executar consultas e análises simultaneamente. O objetivo desses testes é identificar o impacto que as consultas exercem sobre o ETLR, qual o

throughput obtido pelo nosso sistema e realizar uma comparação com uma DW tradicional. A Figura 26 mostra um gráfico com os resultados.

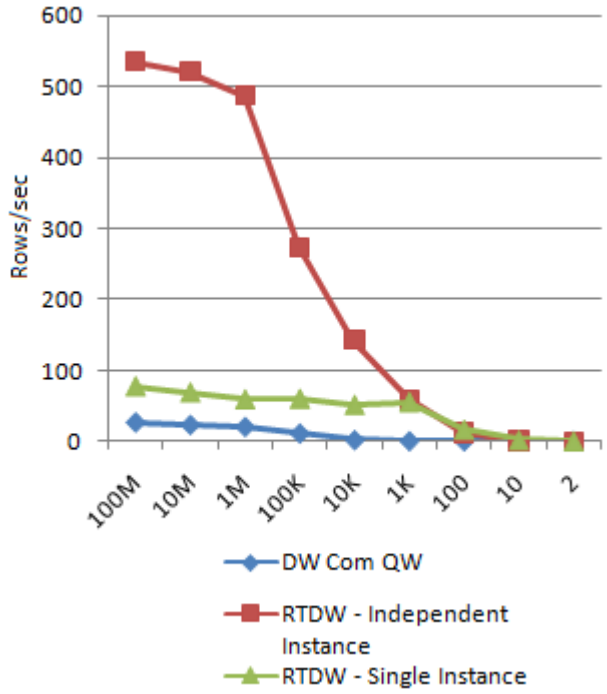


Figura 26: **Gráfico do throughput do processo ETLR.**

Enquanto nas DW tradicionais o *throughput* do processo ETLR online não chega a 30 linhas por segundo, nossa solução conseguiu atingir uma marca de mais de 500 linhas por segundo, um aumento de 1958%. Assumindo que os processos de extração e transformação sejam otimizados e distribuídos em várias máquinas para obter o máximo de performance, é feita uma análise do impacto das consultas sobre a taxa apenas do carregamento dos dados. A Figura 27 mostra os resultados.

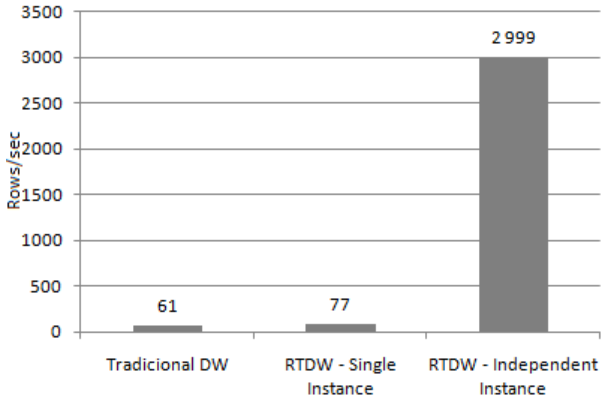


Figura 27: **Gráfico do throughput do processo de carregamento de dados.**

Observa-se um aumento no *throughput* de carregamento, quando colocamos os dois componentes de armazenamento (S-DW e D-DW) em máquinas separadas. Desta forma, garantimos uma taxa de carregamento alta, que garante o carregamento em tempo real.

5.2.4. Impacto das sessões no processo ETLR

Nessa secção, é analisado o impacto que a quantidade de sessões dos utilizadores exerce no processo de integração de dados em tempo real. Foi realizado um comparativo entre a arquitetura tradicional de uma DW e a arquitetura que propomos neste artigo. Os testes realizados para este objetivo seguiram a mesma sistemática para os dois casos, porém, pelo fato do tempo de execução na arquitetura tradicional ser muito elevado, foi escolhido um *mini-batch* com um tamanho menor do que o *mini-batch* escolhido para nossa arquitetura. Enquanto o tamanho na DW tradicional foi de apenas 10 linhas, para nossa arquitetura escolhemos um *mini-batch* com 100k linhas. Os resultados podem ser vistos na Figura 28.

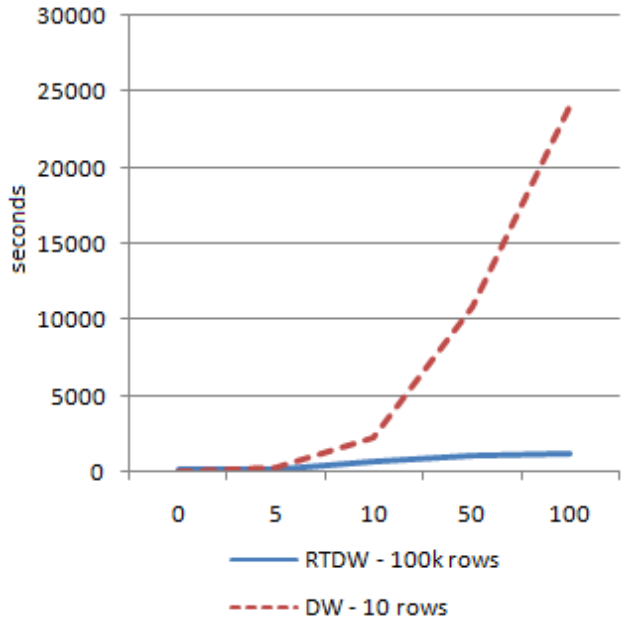


Figura 28: **Gráfico do tempo de execução do processo ETLR com diferentes quantidades de sessões a correr consultas em paralelo.**

Vislumbra-se que, mesmo com o *mini-batch* maior, a arquitetura apresentada se portou de forma satisfatória, obtendo tempos muito abaixo dos gerados na arquitetura

tradicional de uma DW. Com base nesses resultados, podemos concluir que nossa solução possui capacidade de suportar várias sessões de utilizadores.

Capítulo 6
Conclusões

Este trabalho propõe uma arquitetura de RTDW que garante um ritmo de integração em tempo real online. A integração de dados pode ser efetuada continuamente, enquanto os utilizadores submetem consultas e análises de forma simultânea, sem influenciar a performance da integração de dados e sem que esta influencie o desempenho das consultas. Para que isto seja possível, foram criados alguns mecanismos, tais como o *Merger*, o S-DW e o D-DW, os quais permitem atualizações e consultas que tenham acesso aos dados recentes e históricos.

No estado da arte foram vistos trabalhos relacionados às técnicas que permitem a extração de dados, transformação, carregamento e agregação, com um ritmo elevado, com o fito de atingir uma integração em tempo real. Também, foram estudados alguns trabalhos que propõem arquiteturas de RTDW e seus componentes.

Foi desenvolvido um benchmark, o SSB-RT, que define um conjunto de regras, com o objetivo de avaliar sistemas de DW, aplicando um contexto de tempo real, ou seja, com integração de dados contínua. Com base neste benchmark foi desenvolvida uma ferramenta de testes para avaliar as DW utilizadas neste projeto e comprovar que o SSB-RT possui capacidades de medir a eficiência destes sistemas.

A solução de arquitetura RTDW foi descrita, apresentando todos os detalhes e seus componentes. Os principais componentes desenvolvidos nesta arquitetura foram: o D-DW para armazenamento de dados recentes; o S-DW para armazenamento dos dados históricos; o *Merger* para realizar a junção dos dados presentes no S-DW e no D-DW. Definiu-se diversas estratégias de *Merger*, com o objetivo de identificar qual apresentaria um menor *overhead*. Além disso, foram realizadas otimizações no que diz respeito ao carregamento de dados recentes (D-DW), onde as tabelas deste mecanismo não possuem quaisquer restrições de integridade, nem índices.

Por fim, comprovou-se por meio dos testes que a arquitetura tradicional não possui capacidades de integração de dados em tempo real, quando são colocadas várias consultas a correr em simultâneo. Porém, se for efetuado um particionamento bem planeado e se houver pequenos períodos *offline* (sem consultas), é possível integrar dados

recentes nas DW tradicionais. Ressalta-se que foi comprovado que a arquitetura proposta neste trabalho desempenha de forma eficiente a integração de dados em tempo real e *online*, bem como o fato da junção dos dados (S-DW e D-DW) ter apresentado perda de performance insignificante. Outra conclusão, com base nestes testes, foi que a influência das consultas exercida sob o processo de integração, e vice-versa, também é pequena. Assim, é possível concluir que a arquitetura desenvolvida neste trabalho corresponde a uma boa solução de RTDW.

Além disso, como frutos deste trabalho foram produzidos quatro artigos científicos que foram submetidos em conferências e periódicos, como visto na secção 1.4. Estes ainda aguardam avaliação dos revisores.

Devido a facilidade de implantação desta arquitetura de RTDW, surgiu o interesse para futuras investigações, onde o objeto de estudo seria um framework para automação da implantação desta arquitetura RTDW em DW tradicionais de forma simples.

Apêndice A

Aplicação da Metodologia de desenvolvimento no projeto

Este apêndice é responsável pela apresentação de como a metodologia ágil de desenvolvimento Scrum foi aplicada neste trabalho, mostrando detalhes de como a equipe envolvida no projeto foi dividida, a lista de requisitos (plataforma de testes e arquitetura), entre outros. Outro assunto abordado é a aplicação do processo de gestão de riscos adotado neste projeto.

Aplicação do Scrum no projeto

Como dito anteriormente, a metodologia de desenvolvimento seguida neste projeto foi uma adaptação do Scrum, pois para o âmbito deste trabalho não seriam necessárias reuniões diárias, por exemplo.

A seguir, serão detalhados o *Backlog* do produto e as *Sprints* realizadas neste projeto.

Backlog do Produto

Nesta seção serão listados e especificados todos os requisitos funcionais e não funcionais do projeto, porém, é necessário apresentar alguns conceitos básicos sobre os mesmos.

Os requisitos são as descrições das funcionalidades que um determinado software possui, ou seja, detalham o comportamento ou restrição que o sistema possui em determinadas situações. Classificam-se como funcionais, não funcionais ou de domínio.

Os requisitos funcionais são as declarações dos serviços que o sistema possui, como o sistema deve se comportar, de acordo com determinadas entradas, e como o sistema deve agir em determinadas situações.

Já os requisitos não funcionais se aplicam às características dos serviços do sistema. São restrições dos serviços ou das funcionalidades oferecidas pelo sistema. Estes requisitos incluem restrições de tempo, performance, processo de desenvolvimento, entre outros.

Por fim, os requisitos de domínio são características ou restrições provenientes do domínio da aplicação. Por exemplo, se um sistema bancário for desenvolvido, existem várias restrições/características específicas desse tipo de aplicação. Este tipo de requisito pode ser funcional ou não funcional (SOMMERVILLE, 2007).

Diante do exposto, é possível listar e detalhar todos os requisitos do sistema desenvolvido neste projeto. Com o propósito de organizar melhor os requisitos, estes foram classificados de acordo com sua característica: Plataforma de teste, Relatório de dissertação e Arquitetura RTDW. A lista de requisitos funcionais, com seu detalhamento e a lista de atividades de cada requisito é a seguinte:

- **Plataforma de testes e aplicação de automatização de testes**

    1. Gerar dados de origem: essa funcionalidade serve para criação de novos dados de origem, alimentando a base de dados DW. Envolve também a documentação em tutorial;
        a. Criar mecanismo de geração de dados de origem.
    2. Gerir índices e chaves: serviço que realizará as tarefas de gestão (criar ou remover) dos índices e chaves primárias e estrangeiras presentes nas tabelas de fato do modelo estrela do DW e da área de estágio. Inclui ainda a adição desta funcionalidade a aplicação de automatização de testes;
        a. Criar método para criação das chaves e índices na área de estágio (STA);
        b. Criar método para criação das chaves e índices nas estrelas;
        c. Criar método para destruir as chaves e índices na STA;
        d. Criar método para destruir as chaves e índices nas estrelas.

3. Gerir área de estágio: mecanismo responsável por gerir a área de estágio que auxilia o processo de ETL. Este possuirá métodos para limpar as tabelas de fato da área de estágio e realizar o carregamento dos dados. Inclui ainda a adição desta funcionalidade à aplicação de automatização de testes;
    a. Criar método para limpeza da STA;
    b. Criar método para preparação e envio dos dados para carregamento na STA.
4. Gerir as estrelas: semelhante ao item anterior, esta funcionalidade realizará a gestão das tabelas de fato da base de dados DW. Contém a adição desta funcionalidade à aplicação de automatização de testes;
    a. Criar método para preparação e envio dos dados para carregamento nas estrelas.
5. Agregar informações: este serviço é responsável por realizar o agrupamento das informações em conformidade com as visões definidas no modelo da base de dados DW e a atualização das tabelas de agregação. Abarca ainda a adição desta funcionalidade à aplicação de automatização de testes;
    a. Criar métodos para realização da agregação dos dados para todas as tabelas de agregação.
6. Fazer ETL: requisito que realiza a tarefa do processo de ETL. Este função utiliza um conjunto de outras funcionalidades para desempenhar as atividades necessárias para seu funcionamento. Este procedimento deve funcionar tanto para o DW tradicional quanto para o RTDW. Abrange também a adição desta funcionalidade à aplicação de automatização de testes;
    a. Criar mecanismo de leitura do ficheiro com dados de origem;
    b. Criação do ISB (*Input Stream Buffer*);
    c. Desenvolver as transformações do processo ETL;
    d. Criar o mecanismo para carregamento dos dados (área de estágio e estrelas);
    e. Criação do método principal que realizará o processo de ETL;
    f. Criação do método principal que realizará o processo de ETL-RTDW;

7. Fazer *Query Workload*: funcionalidade que irá simular a execução de consultas efetuadas por utilizadores. Esta função deve permitir a simulação de várias consultas em simultâneo, conjunto de consultas pré-definidas e recuperação do tempo de execução de cada consulta realizada. Inclui ainda a adição desta funcionalidade à aplicação de automatização de testes;
    a. Definir as consultas para o *query workload*;
    b. Criar mecanismo de consultas aleatórias;
    c. Criar o mecanismo de junção de informações (*Merger*);
8. Criar ambiente de teste: funcionalidade que é responsável pela criação de todo o ambiente para execução dos testes. Inclui também a adição desta funcionalidade à aplicação de automatização de testes;
    a. Criar o mecanismo de instalação do ambiente de testes;
    b. Desenvolver mecanismo para criação dos dados iniciais que serão carregados no DW.

- **Relatório de Dissertação e Disseminação**

9. Criar documentação: elaborar documentação de um determinado assunto que esteja sendo abordado;
    a. Criar documentação do estado da arte;
    b. Criar documentação relacionada à plataforma de testes para DW tradicionais;
    c. Desenvolver o relatório intermédio;
    d. Elaborar apresentação intermédia do projeto;
    e. Criar a documentação relacionada à plataforma de testes para RTDW;
    f. Criar documentação relacionada aos testes concretizados;
    g. Criar artigo sobre os testes realizados na arquitetura tradicional;
    h. Criar artigo sobre a arquitetura RTDW e testes;
    i. Criar artigo sobre o benchmark definido;
    j. Criar *survey* sobre RTDW;
    k. Aperfeiçoamento do relatório para a versão final;
    l. Aperfeiçoamento da apresentação para a versão final;

10. Pesquisar material de apoio: requisito para pesquisa de material de apoio para servir de base teórica para elaboração de documentação;
    a. Levantamento de material para o estado da arte;
    b. Levantamento de material para metodologia de desenvolvimento e gestão de riscos.
11. Revisar documentação: requisito para realização de revisão do(s) documento(s) desenvolvido(s) com a finalidade de identificar erros ou elementos mal escritos;
    a. Revisar documentação referente ao estado da arte;
    b. Revisar documentação relacionada a plataforma de testes DW;
    c. Revisar relatório intermédio;
    d. Revisar apresentação intermédia;
    e. Revisar documentação relacionada a plataforma de testes RTDW;
    f. Revisar relatório final;
    g. Revisar apresentação final.
12. Realizar testes: requisito para realização dos testes e obtenção dos resultados para futura avaliação;
    a. Testes sob a arquitetura de DW tradicional num contexto de tempo real;
    b. Testes sob a arquitetura RTDW.

Agora serão listados alguns requisitos não funcionais encontrados neste projeto:

1. Performance no *Merger*: o tempo que o *Merger* leva para retornar o resultado final não pode ser muito diferente do tempo que uma consulta leva para retornar o resultado quando executada no DW, no máximo 10% superior;
2. A plataforma de testes deve ser facilmente instalada para testes futuros;
3. A plataforma deve possibilitar testes em diversos motores de base de dados;
4. A solução deverá ser desenvolvida em Java;

Sprints do projeto

A duração de cada *Sprint* deste projeto depende do tamanho e da complexidade do *Backlog* do mesmo. Nesta seção todos os *Sprints* do projeto serão detalhados, apresentando sua duração, *Backlog* e o incremento desenvolvido.

A primeira *Sprint* do projeto foi responsável pelo levantamento inicial de material para desenvolvimento do estado da arte do relatório final e a criação dos *scripts* para o modelo das bases de dados. A Tabela 14 mostra o *Backlog* desta *Sprint* com todas as tarefas, duração total da *Sprint*, duração de cada tarefa e o incremento produzido. É importante frisar que, o Scrum sugere que as *Sprints* tenham uma duração máxima de quatro semanas, porém, este projeto não segue, rigorosamente, esta metodologia.

Sprint 1 : Planeamento Inicial - 38 dias		Requisito	Inicio	Fim
1	Levantamento de Material para Estado da Arte	10	01/10/2012	14/10/2012
2	Documentação do Estado da Arte	9	15/10/2012	25/10/2012
3	Criar scripts para criação das estrelas das bases de dados	8	26/10/2012	27/10/2012
4	Criar scripts para criação das tabelas auxiliares	8	28/10/2012	29/10/2012
5	Criar scripts para destruir tabelas auxiliares	8	30/10/2012	31/10/2012
6	Revisão da documentação	11	01/11/2012	07/11/2012
Incremento: Documento com o estado da arte do projeto, lista de requisitos e definição da arquitetura RTDW.				

Tabela 14: **Detalhes da *Sprint* 1.**

A partir da segunda *Sprint*, visualiza-se tarefas relacionadas com o desenvolvimento da plataforma de testes para arquitetura tradicional de DW. Inicialmente, foi realizada uma reunião com o *Time Scrum*, com a finalidade de refinar os requisitos do incremento a ser desenvolvido. Posteriormente, deu-se inicio ao desenvolvimento das atividades restantes listadas no *Backlog* dessa *Sprint*. A Tabela 15 mostra os detalhes da *Sprint* 2.

Sprint 2 : Plataforma de testes DW Parte 1 - 30 dias		Requisito	Inicio	Fim
1	Geração dos dados de origem	1	08/11/2012	11/11/2012
2	Leitura de ficheiro de dados de origem	6	12/11/2012	14/11/2012
3	Buffer de entrada	6	14/11/2012	15/11/2012
4	Transformações do ETL	6	16/11/2012	20/11/2012
5	Criar scripts para criação das chaves primárias e estrangeiras	2	21/11/2012	24/11/2012
6	Definir consultas para query workload	7	25/11/2012	27/11/2012
7	Criar mecanismo de consultas aleatórias	7	28/11/2012	08/12/2012
Incremento: Extração e transformação dos dados.				

Tabela 15: **Detalhes da *Sprint* 2.**

Devido ao grande número de requisitos, houve a necessidade de dividir o desenvolvimento da plataforma de testes. A *Sprint* seguinte (terceira) abrange as tarefas relacionadas ao desenvolvimento restante da plataforma de testes. A Tabela 16 ilustra a configuração desta *Sprint*.

Sprint 3 : Plataforma de testes DW Parte 2 - 21 dias		Requisito	Inicio	Fim
1	Criar método para criação das chaves na área de estágio (STA)	2	09/12/2012	10/12/2012
2	Criar método para criação das chaves na estrela	2	09/12/2012	10/12/2012
3	Criar método para destruir chaves na STA	2	11/12/2012	12/12/2012
4	Criar método para destruir chaves nas estrelas	2	11/12/2012	12/12/2012
5	Criar método para limpeza da STA	3	13/12/2012	14/12/2012
6	Método para preparar e enviar dados para carregar na STA	3	13/12/2013	14/12/2013
7	Método para preparar e enviar dados para carregar nas estrelas	4	13/12/2014	14/12/2014
8	Loader (carregamento dos dados)	6	15/12/2012	18/12/2012
9	Agregadores	5	19/12/2012	03/01/2013
Incremento: Gestor de índices, Gestor de área de estágio, Gestor das estrelas, Loader e Agregadores.				

Tabela 16: Detalhes da *Sprint* 3.

A *Sprint* 4 é responsável pela finalização do desenvolvimento do processo de ETL para a plataforma tradicional do DW e pela documentação da plataforma desenvolvida. Nesta etapa, também foi realizado o desenvolvimento do relatório intermédio e da apresentação deste projeto. A Tabela 17 demonstra as atividades, em pormenores, que foram abordadas nesta *Sprint*.

Sprint 4 : Relatório Intermédio - 30 dias		Requisito	Inicio	Fim
1	Criação do método para o processo ETL	6	04/01/2013	19/01/2013
2	Documentação da plataforma de testes DW	9	20/01/2013	22/01/2013
3	Desenvolvimento do relatório intermédio	9	23/01/2013	26/01/2013
4	Revisão da documentação	11	27/01/2013	27/01/2013
5	Elaboração da Apresentação intermédia	9	28/01/2013	01/02/2013
6	Revisão da apresentação	11	02/02/2013	03/02/2013
Incremento: Processo ETL, documento da plataforma de testes DW e relatório intermédio.				

Tabela 17: Detalhes da *Sprint* 4.

O inicio do desenvolvimento da plataforma de testes para o RTDW foi evidenciado na *Sprint* 5. Esta iteração possui as tarefas referentes aos mecanismos utilizados na plataforma RTDW, documentação e o desenvolvimento da instalação automática das plataformas de testes desenvolvidas neste projeto. A Tabela 18 mostra detalhes da *Sprint* 5.

Sprint 5 : Plataforma de testes RTDW - 30 dias		Requisito	Inicio	Fim
1	Merger	7	04/02/2013	06/02/2013
2	Master Node	7	07/02/2013	09/02/2013
3	Processo ETL-RTDW	6	10/02/2013	15/02/2013
4	Documentação da plataforma de testes RTDW	9	16/02/2013	18/02/2013
5	Revisão da documentação	11	19/02/2013	20/02/2013
6	Criar mecanismo para instalação do ambiente de testes	8	21/02/2013	03/03/2013
7	Criar mecanismo para geração de dados iniciais na DW	8	04/03/2013	07/03/2013
Incremento: Processo ETL-RTDW e documentação da plataforma de testes RTDW.				

Tabela 18: Detalhes da *Sprint* 5.

A próxima *Sprint* possui o planejamento dos testes para a fase de avaliação das arquiteturas. Nesta iteração foram listadas e foram atribuídos os prazos em todas as tarefas relacionadas aos testes tanto da plataforma tradicional de DW quanto na arquitetura RTDW. A Tabela 19 mostra a listagem da *Sprint* 6.

Sprint 6 : Testes e Avaliações - 55 dias		Requisito	Inicio	Fim
1	Testes em arquitetura tradicional	12	08/03/2013	23/03/2013
2	Testes em arquitetura RTDW	12	24/03/2013	13/04/2013
3	Avaliação dos teste	9	14/04/2013	20/04/2013
4	Documentação dos testes	9	21/04/2013	02/05/2013
Incremento: Todos os resultados dos testes e capítulo da tese com a análise destes resultados.				

Tabela 19: Detalhes da *Sprint* 5.

Por fim, a *Sprint* 7 possui atividades relacionadas à finalização do trabalho. Essencialmente atividades de documentação e revisão são encontradas nesta *Sprint* final. A Tabela 20 mostra as tarefas listadas nesta *Sprint*.

Sprint 7 : Entrega Final - 69 dias		Requisito	Inicio	Fim
1	Desenvolvimento de artigo: "Near Real-time in traditional Data Warehouse"	9	29/04/2013	15/05/2013
2	Desenvolvimento de artigo: "Real-time Data Warehouse Architecture"	9	16/05/2013	31/05/2013
3	Desenvolvimento de artigo: "Real-time Data Warehouse Survey"	9	01/06/2013	15/06/2013
4	Desenvolvimento de artigo: "A Real-time Data Warehouse Benchmark"	9	16/06/2013	26/06/2013
5	Aperfeiçoamento do relatório	9	27/06/2013	30/06/2013
6	Revisão do relatório	11	01/07/2013	02/07/2013
7	Aperfeiçoamento da apresentação	9	03/07/2013	07/07/2013
Incremento: Relatório, apresentação final do projeto e artigos submetidos em conferências ou revistas.				

Tabela 20: Detalhes da *Sprint* 5.

Gestão de Riscos direcionada ao projeto

Após a constante gestão dos riscos do projeto foi produzido um artefato cujo seu conteúdo é a lista dos riscos encontrados. As ocorrências desta lista se encontram classificadas, priorizadas, possuem a probabilidade de ocorrência (Baixa, Média ou Alta), o nível de impacto que pode realizar no projeto e a medida de prevenção ou correção caso esta ocorrência aconteça. A Tabela 21 mostra todas as ocorrências encontradas neste processo de gestão de riscos.

Ocorrência	Descrição	Prioridade	Classificação	Probabilidade	Impacto
0	Arquitetura não seja eficaz no carregamento	0	IMPORTANTE	MÉDIA	ALTO
Ação: Criar mecanismos alternativos que auxiliem no carregamento de dados por agendamento.					
Ocorrência	**Descrição**	**Prioridade**	**Classificação**	**Probabilidade**	**Impacto**
1	Estratégia merge pouco eficiente	1	IMPORTANTE	MÉDIA	MÉDIO
Ação: Criar um mecanismo de cache que auxilie nesta tarefa e adicione mais performance à funcionalidade.					
Ocorrência	**Descrição**	**Prioridade**	**Classificação**	**Probabilidade**	**Impacto**
2	Ultrapassar prazo de desenvolvimento	2	MÉDIO	BAIXA	BAIXO
Ação: Eliminar requisitos menos importantes.					
Ocorrência	**Descrição**	**Prioridade**	**Classificação**	**Probabilidade**	**Impacto**
3	Ultrapassar prazo de teste	3	MÉDIO	MÉDIA	BAIXO
Ação: Diminuir a quantidade de testes, realizando cálculos por estimativa.					
Ocorrência	**Descrição**	**Prioridade**	**Classificação**	**Probabilidade**	**Impacto**
4	Problema na aquisição do novo servidor	4	MÉDIO	MÉDIO	MÉDIO
Ação: Utiliação de uma das máquimas existentes no laboratório.					

Tabela 21: Lista de riscos encontrados no projeto.

Apêndice B

Detalhes específicos do Benchmark SSB-RT

Este apêndice possui detalhes específicos do SSB-RT, visto no capítulo 4, relacionados à estrutura das tabelas, *query workload*, entre outros.

Estrutura das tabelas

Coluna	Tipo	Restrição
O_ORDERKEY	Numérico	PRIMARY KEY
O_CUSTKEY	Numérico	FOREIGN KEY
O_ORDERSTATUS	Alfanumérico	NOT NULL
O_TOTALPRICE	Numérico	NOT NULL
O_AVGREVENUE	Numérico	NOT NULL
O_AVGTAX	Numérico	NOT NULL
O_AVGDISCOUNT	Numérico	NOT NULL
O_TOTALSUPPLYCOST	Numérico	NOT NULL
O_TOTALITEM	Numérico	NOT NULL
O_TOTALQUANTITY	Numérico	NOT NULL
O_ORDERDATE	Numérico	FOREIGN KEY
O_ORDERTIME	Numérico	FOREIGN KEY
O_ORDERPRIORITY	Alfanumérico	NOT NULL
O_CLERK	Alfanumérico	NOT NULL
O_SHIPPRIORITY	Numérico	NOT NULL

Tabela 22: Colunas da tabela de fato Orders.

Coluna	Tipo	Restrição
L_ORDERKEY	Numérico	PRIMARY KEY
L_LINENUMBER	Numérico	PRIMARY KEY
L_CUSTKEY	Numérico	FOREIGN KEY
L_PARTKEY	Numérico	FOREIGN KEY
L_SUPPKEY	Numérico	FOREIGN KEY
L_ORDERDATE	Numérico	FOREIGN KEY
L_ORDERTIME	Numérico	FOREIGN KEY
L_ORDERPRIORITY	Alfanumérico	
L_SHIPPRIORITY	Numérico	
L_QUANTITY	Numérico	NOT NULL
L_EXTENDEDPRICE	Numérico	NOT NULL
L_ORDTOTALPRICE	Numérico	NOT NULL
L_SUPPLYCOST	Numérico	NOT NULL
L_DISCOUNT	Numérico	NOT NULL
L_REVENUE	Numérico	NOT NULL
L_TAX	Numérico	NOT NULL
L_COMMITDATE	Numérico	FOREIGN KEY
L_SHIPMODE	Alfanumérico	

Tabela 23: Colunas da tabela de fato Lineorder.

Coluna	Tipo	Restrição
D_DATEKEY	Numérico	PRIMARY KEY
D_DATEID	Alfanumérico	NOT NULL
D_DATE	Data	NOT NULL
D_MONTHSEQ	Numérico	NOT NULL
D_WEEKSEQ	Numérico	NOT NULL
D_QUARTERSEQ	Numérico	NOT NULL
D_YEAR	Numérico	NOT NULL
D_DOW	Numérico	NOT NULL
D_DOM	Numérico	NOT NULL
D_DAYNAME	Alfanumérico	NOT NULL
D_QUARTERNAME	Alfanumérico	NOT NULL
D_HOLIDAY	Alfanumérico	NOT NULL
D_WEEKEND	Alfanumérico	NOT NULL
D_FOLLOWINGHOLIDAY	Alfanumérico	NOT NULL

Tabela 24: Colunas da tabela de dimensão Date_dim.

Coluna	Tipo	Restrição
T_TIMEKEY	Numérico	PRIMARY KEY
T_TIME	Alfanumérico	NOT NULL
T_HOUR	Numérico	NOT NULL
T_MINUTE	Numérico	NOT NULL
T_SECOND	Numérico	NOT NULL
T_AM_PM	Alfanumérico	NOT NULL

Tabela 25: Colunas da tabela de dimensão Time_dim.

Coluna	Tipo	Restrição
C_CUSTKEY	Numérico	PRIMARY KEY
C_NAME	Alfanumérico	NOT NULL
C_ADDRESS	Alfanumérico	NOT NULL
C_NATION	Alfanumérico	NOT NULL
C_REGION	Alfanumérico	NOT NULL
C_PHONE	Alfanumérico	NOT NULL
C_MKTSEGMENT	Alfanumérico	NOT NULL

Tabela 26: Colunas da tabela de dimensão Customer.

Coluna	Tipo	Restrição
S_SUPPKEY	Numérico	PRIMARY KEY
S_NAME	Alfanumérico	NOT NULL
S_ADDRESS	Alfanumérico	NOT NULL
S_NATION	Alfanumérico	NOT NULL
S_REGION	Alfanumérico	NOT NULL
S_PHONE	Alfanumérico	NOT NULL

Tabela 27: Colunas da tabela de dimensão Supplier.

Coluna	Tipo	Restrição
P_PARTKEY	Numérico	PRIMARY KEY
P_NAME	Alfanumérico	NOT NULL
P_MFRG	Alfanumérico	NOT NULL
P_BRAND	Alfanumérico	NOT NULL
P_TYPE	Alfanumérico	NOT NULL
P_SIZE	Numérico	NOT NULL
P_CONTAINER	Alfanumérico	NOT NULL
P_RETAILPRICE	Numérico	NOT NULL

Tabela 28: Colunas da tabela de dimensão Part.

Coluna	Tipo	Restrição
O_ORDERKEY	Numérico	
O_CUSTKEY	Numérico	
O_ORDERSTATUS	Alfanumérico	
O_TOTALPRICE	Numérico	
O_AVGREVENUE	Numérico	
O_AVGTAX	Numérico	
O_AVGDISCOUNT	Numérico	
O_TOTALSUPPLYCOST	Numérico	
O_TOTALITEM	Numérico	
O_TOTALQUANTITY	Numérico	
O_ORDERDATE	Numérico	
O_ORDERTIME	Numérico	
O_ORDERPRIORITY	Alfanumérico	
O_CLERK	Alfanumérico	
O_SHIPPRIORITY	Numérico	

Tabela 29: Colunas da tabela auxiliar Orders_STA.

Coluna	Tipo	Restrição
L_ORDERKEY	Numérico	
L_LINENUMBER	Numérico	
L_CUSTKEY	Numérico	
L_PARTKEY	Numérico	
L_SUPPKEY	Numérico	
L_ORDERDATE	Numérico	
L_ORDERTIME	Numérico	
L_ORDERPRIORITY	Alfanumérico	
L_SHIPPRIORITY	Numérico	
L_QUANTITY	Numérico	
L_EXTENDEDPRICE	Numérico	
L_ORDTOTALPRICE	Numérico	
L_SUPPLYCOST	Numérico	
L_DISCOUNT	Numérico	
L_REVENUE	Numérico	
L_TAX	Numérico	
L_COMMITDATE	Numérico	
L_SHIPMODE	Alfanumérico	

Tabela 30: Colunas da tabela auxiliar Lineorder_STA.

Dados do ficheiro de log (LogFile)

Campo	Descrição do campo	Tipo
Cabeçalho	Cabeçalho indicativo do tipo da linha, neste caso "ORDER".	Alfabético
ORDERKEY	Número identificador da venda.	Numérico
CUSTKEY	Número que identifica o cliente.	Numérico
ORDERSTATUS	Estado da encomenda.	Alfanumérico
TOTALPRICE	Valor total da encomenda.	Numérico
ORDERDATE	Data da venda.	Data (YYYY-MM-DD)
ORDER-PRIORITY	Prioridade da encomenda.	Alfanumérico
CLERK	Funcionário que efetuou a encomenda.	Alfanumérico
SHIP-PRIORITY	Prioridade na entrega da encomenda.	Numérico
COMMENT	Comentários sobre a venda.	Alfanumérico

Tabela 31: Campos dos registros Orders do ficheiro LogFile.

Campo	Descrição do campo	Tipo
Cabeçalho	Cabeçalho indicativo do tipo da linha, neste caso "LINEITEM".	Alfabético
ORDERKEY	Número identificador da venda.	Numérico
PARTKEY	Número identificador do produto da venda.	Numérico
SUPPKEY	Número identificador do fornecedor do produto.	Numérico
LINENUMBER	Número sequencial do item.	Numérico
QUANTITY	Quantidade vendida do produto.	Numérico
EXTENDEDPRICE	Valor do produto.	Numérico
DISCOUNT	Desconto aplicado no produto.	Numérico
TAX	Taxas aplicadas no produto.	Numérico
RETURNFLAG		Alfanumérico
LINESTATUS	Status do item da venda.	Alfanumérico
SHIPDATE	Data de entrega do item.	Data (YYYY-MM-DD)
COMMITDATE		Data (YYYY-MM-DD)
RECEIPTDATE	Data de recebimento do item.	Data (YYYY-MM-DD)
SHIPINSTRUCT	Instruções relacionadas com a entrega do item.	Alfanumérico
SHIPMODE	Tipo da entrega.	Alfanumérico
COMMENT	Comentário sobre o item.	Alfanumérico

Tabela 32: Campos dos registros Lineitem do ficheiro LogFile.

Query Workload

Quadro B.1. Consulta Q1

```
select sum(l_extendedprice*l_discount) as revenue
  from <<TABLESPACE>>lineorder<<DBLINK>>,
       <<TABLESPACE>>date_dim<<DBLINK>>
  where
  l_orderdate = d_datekey and
  d_year = <<Y1>> and
  l_discount between 1
  and 7 and
  l_quantity < 25;
```

Quadro B.2. Consulta Q2

```
select sum(l_extendedprice*l_discount) as revenue
 from <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>date_dim<<DBLINK>>
 where
 l_orderdate = d_datekey and
 d_year = <<Y1>> and
 d_monthseq = 01  and
 l_discount between 3 and 9 and
 l_quantity between 26 and 35;
```

Quadro B.3. Consulta Q3

```
select sum(l_extendedprice*l_discount) as revenue
 from <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>date_dim<<DBLINK>>
 where
 l_orderdate = d_datekey and
 d_weekseq = 6 and
 d_year = <<Y1>> and
 l_discount between 2 and 8 and
 l_quantity between 26 and 35;
```

Quadro B.4. Consulta Q4

```
select sum(l_revenue) as l_revenue, d_year, p_brand
 from <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>date_dim<<DBLINK>>,
        <<TABLESPACE>>part<<DBLINK>>, <<TABLESPACE>>supplier<<DBLINK>>
 where
 l_orderdate = d_datekey and
 l_partkey = p_partkey and
 l_suppkey = s_suppkey and
 p_category = '<<CAT>>' and
 s_region = '<<REG>>'
 group by d_year, p_brand
 order by d_year, p_brand;
```

Quadro B.5. Consulta Q5

```
select sum(l_revenue) as l_revenue, d_year, p_brand
 from <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>date_dim<<DBLINK>>,
        <<TABLESPACE>>part<<DBLINK>>, <<TABLESPACE>>supplier<<DBLINK>>
 where
 l_orderdate = d_datekey and
 l_partkey = p_partkey and
 l_suppkey = s_suppkey and
 p_brand between 'MFGR#2221'
 and 'MFGR#2228' and
 s_region = '<<REG>>'
 group by d_year, p_brand
 order by d_year, p_brand;
```

Quadro B.6. Consulta Q6

```
select sum(l_revenue) as l_revenue, d_year, p_brand
 from <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>date_dim<<DBLINK>>,
        <<TABLESPACE>>part<<DBLINK>>, <<TABLESPACE>>supplier<<DBLINK>>
 where
 l_orderdate = d_datekey and
 l_partkey = p_partkey and
 l_suppkey = s_suppkey and
 p_brand = 'MFGR#2221' and
 s_region = '<<REG>>'
 group by d_year, p_brand
 order by d_year, p_brand;
```

Quadro B.7. Consulta Q7

```
select c_nation, s_nation, d_year, sum(l_revenue) as revenue
 from <<TABLESPACE>>customer<<DBLINK>>,
        <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>supplier<<DBLINK>>, <<TABLESPACE>>date_dim<<DBLINK>>
 where
 l_custkey = c_custkey and
 l_suppkey = s_suppkey and
 l_orderdate = d_datekey and
 c_region = '<<REG>>' and
 s_region = '<<REG>>' and
 d_year >= <<Y1>> and
 d_year <= <<Y2>>
 group by c_nation, s_nation, d_year
 order by d_year asc, revenue desc;
```

Quadro B.8. Consulta Q8

```
select c_city, s_city, d_year, sum(l_revenue) as revenue
 from <<TABLESPACE>>customer<<DBLINK>>,
        <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>supplier<<DBLINK>>, <<TABLESPACE>>date_dim<<DBLINK>>
 where
 l_custkey = c_custkey and
 l_suppkey = s_suppkey and
 l_orderdate = d_datekey and
 c_nation = '<<REG>>' and
 s_nation = '<<REG>>' and
 d_year >= <<Y1>> and
 d_year <= <<Y2>>
 group by c_city, s_city, d_year
 order by d_year asc, revenue desc;
```

Quadro B.9. Consulta Q9

```
select c_city, s_city, d_year, sum(l_revenue) as revenue
 from <<TABLESPACE>>customer<<DBLINK>>,
        <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>supplier<<DBLINK>>, <<TABLESPACE>>date_dim<<DBLINK>>
 where l_custkey = c_custkey and
 l_suppkey = s_suppkey and
 l_orderdate = d_datekey and
 (c_city='<<C1>>' or c_city='<<C2>>') and
 (s_city='<<C1>>' or s_city='<<C2>>') and
 d_year >= <<Y1>> and
 d_year <= <<Y2>>
 group by c_city, s_city, d_year
 order by d_year asc, revenue desc;
```

Quadro B.10. Consulta Q10

```
select  c_city, s_city, d_year, sum(l_revenue) as revenue
 from <<TABLESPACE>>customer<<DBLINK>>,
        <<TABLESPACE>>lineorder<<DBLINK>>,
        <<TABLESPACE>>supplier<<DBLINK>>, <<TABLESPACE>>date_dim<<DBLINK>>
 where
 l_custkey = c_custkey and
 l_suppkey = s_suppkey and
 l_orderdate = d_datekey and
 (c_city='<<C1>>' or c_city='<<C2>>') and
 (s_city='<<C1>>' or s_city='<<C2>>') and
 d_year = <<Y1>> and
 d_monthseq = 12
 group by c_city, s_city, d_year
 order by d_year asc, revenue desc;
```

Quadro B.11. Consulta Q11

```
select d_year, c_nation, sum(l_revenue - l_supplycost) as profit
 from <<TABLESPACE>>date_dim<<DBLINK>>,
        <<TABLESPACE>>customer<<DBLINK>>,
        <<TABLESPACE>>supplier<<DBLINK>>, <<TABLESPACE>>part<<DBLINK>>,
        <<TABLESPACE>>lineorder<<DBLINK>>
 where
 l_custkey = c_custkey and
 l_suppkey = s_suppkey and
 l_partkey = p_partkey and
 l_orderdate = d_datekey and
 c_region = '<<REG>>' and
 s_region = '<<REG>>' and
 (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
 group by d_year, c_nation
 order by d_year, c_nation;
```

Quadro B.12. Consulta Q12

```
select d_year, s_nation, p_category, sum(l_revenue - l_supplycost) as profit
from <<TABLESPACE>>date_dim<<DBLINK>>,
     <<TABLESPACE>>customer<<DBLINK>>,
     <<TABLESPACE>>supplier<<DBLINK>>, <<TABLESPACE>>part<<DBLINK>>,
     <<TABLESPACE>>lineorder<<DBLINK>>
where
l_custkey = c_custkey and
l_suppkey = s_suppkey and
l_partkey = p_partkey and
l_orderdate = d_datekey and
c_region = '<<REG>>' and
s_region = '<<REG>>' and
(d_year = <<Y1>> or d_year = <<Y2>>) and
(p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, s_nation, p_category
order by d_year, s_nation, p_category;
```

Quadro B.13. Consulta Q13

```
select d_year, s_city, p_brand, sum(l_revenue - l_supplycost) as profit
from <<TABLESPACE>>date_dim<<DBLINK>>,
     <<TABLESPACE>>customer<<DBLINK>>,
     <<TABLESPACE>>supplier<<DBLINK>>, <<TABLESPACE>>part<<DBLINK>>,
     <<TABLESPACE>>lineorder<<DBLINK>>
where
l_custkey = c_custkey and
l_suppkey = s_suppkey and
l_partkey = p_partkey and
l_orderdate = d_datekey and
c_region = '<<REG>>' and
s_nation = '<<NAT>>' and
(d_year = <<Y1>> or d_year = <<Y2>>) and
p_category = '<<CAT>>'
group by d_year, s_city,  p_brand
order by d_year, s_city,  p_brand;
```

Referências

BRUCKNER, R.M., LIST, B., SCHIEFER, J. 2002. Striving towards near real-time data integration for data warehouses. In: Proceedings of DaWaK. (2002) 317-326.

CONDIE, T., CONWAY, N., ALVARO, P., HELLERSTEIN, J.M., GERTH, J., TALBOT, J., ELMELEEGY, K., SEARS, R. 2010. Online aggregation and continuous query support in mapreduce. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 1115–1118. ACM (2010).

COOPER, D., GREY, S., RAYMOND, G., and WALKER, P. 2005. Project Risk Management Guidelines, Editora: John Wiley and Sons, Ltd, 2005, ISBN: 0-470-02281-7.

DBLINK, 2013. Create Database Link, http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_5005.htm.

ETI, 2013. http://www.eti.com/product-details.

GARTNER, 2012. Market Share: All Software Markets, Worldwide, 2011. http://www.gartner.com/DisplayDocument?ref=clientFriendlyUrl&id=1969315#document_history.

GOLDENGATE Software, Inc. 2009. Going Real-Time for Data Warehousing and Operational BI, White Paper.

GUERRA, J., ANDREWS, D. 2011. "Creating a Real Time Data Warehouse". White Paper.

INFORMATICA, 2013. Informatica PowerExchange. http://www.informatica.com/us/products/enterprise-data-integration/powerexchange/.

JÖRG, T., and DESSLOCH, S. 2010. Near Real-time data warehousing using state-of-the-art ETL tools. Enabling Real-Time Business Intelligence, Lecture Notes in Business Information Processing, Volume 41. ISBN 978-3-642-14558-2. Springer-Verlag Heidelberg (2010).

KIMBALL, R., CASERTA, J. 2004. The data warehouse ETL toolkit: Practical techniques for extracting, cleaning, conforming, and delivering data. John Wiley & Sons (2004).

NGUYEN, M., TJOAV, A. 2003. Zero-Latency Data Warehousing for Heterogeneous Data Sources and Continuous Data Streams, Fifth International Conference on Information and Web-based Applications and Services, Austrian Computer Society (OCG) (2003).

O'NEIL, P., O'NEIL, E., and CHEN, X. 2009. Star Schema Benchmark. http://www.cs.umb.edu/~poneil/StarSchemaB.PDF.

ORACLE, 2013. Oracle Golden Gate. http://www.oracle.com/technetwork/middleware/goldengate/overview/index.html.

ORACLE, 2012. Best Practices for Real-time Data Warehousing, White Paper.

POLYZOTIS, N., SKIADOPOULOS, S., VASSILIADIS, P., SIMITSIS, A., FRANTZELL, N. 2007. Supporting Streaming Updates in an Active Data Warehouse. In ICDE, pp. 476–485, 2007.

RAM, P., DO, L. 2000. Extracting delta for incremental data Warehouse maintenance. In ICDE ’00: Proceedings of the 16th International Conference on Data Engineering, page 220, Washington, DC, USA, 2000.

SANTOS, R., BERNARDINO, J. 2008. Real-Time Data Warehouse Loading Methodology, International Database Engineering and Applications Symposium (IDEAS), 2008.

SCHALLEHN, E., SATTLER, K., SAAKE, G. 2001. Advanced Grouping and Aggregation for Data Integration. CIKM’01, November 2001, Atlanta, Georgia, USA, 2001.

SCHWABER, K., and SUTHERLAND, J. 2011. Scrum Guide. http://www.scrum.org/Scrum-Guides.

SHI, J., BAO, Y., LENG, F., and YU, G. 2008. Study on log-based change data capture and handling mechanism in real-time data warehouse. International Conference on Computer Science and Software Engineering. Washington, DC, USA, 2008, 478-481.

SOMMERVILLE, I. 2007. *Engenharia de Software*, 8a. Edição, Editora: Addison-Wesley, ISBN: 9788588639287.

THOMSEN, C.S., PEDERSEN, T.B., LEHNER, W. 2008. RiTE: Providing on-demand data for right-time data warehousing. In: Proceedings of ICDE, Washington, DC, USA, IEEE Computer Society (2008) 456-465.

TPC-H, 2012. TPC-H. http://tpc.org/tpch/default.asp.

VASSILIADIS, P., and SIMITSIS, A. 2008. Near Real Time ETL. In Springer journal Annals of Information Systems, Vol. 3, Special issue on New Trends in Data Warehousing and Data Analysis, ISBN 978-0-387-87430-2, Springer.

VERTICA, 2013, Real-Time Loading & Querying. http://www.vertica.com/the-analytics-platform/real-time-loading-querying/.

WAAS, F., WREMBEL, R., FREUDENREICH, T., THIELE, M., KONCILIA, C., FURTADO, P. 2013. On-Demand ELT Architecture for Right-Time BI: Extending the Vision. In: International Journal of Data Warehousing and Mining (IJDWM), volume 9 number 2 (2013).

WAAS, F., WREMBEL, R., FREUDENREICH, T., THIELE, M., KONCILIA, C., FURTADO, P. 2012. On-Demand ETL Architecture for Right-Time BI. In: Proceedings of the 6th International Workshop on Business Intelligence for the Real-Time Enterprise (BIRTE), Istanbul (2012).

WIKI, 2013. Scrum. http://pt.wikipedia.org/wiki/Scrum.

WYATT, L., CAUFIELD, B., POL, D. 2009. Principles for an ETL Benchmark. In: R. Nambiar and M. Poess: TPCTC 2009. LNCS 5895. Springer-Verlag, Berlin Heidelberg (2009) 183-198.

YU, T. 2006. A Materialized View-based Approach to Integrating ETL Process and Data Warehouse Applications. International Conference on Information and Knowledge Engineering, Las Vegas, Nevada, USA, Junho 2006, 257-263.

ZHU, Y., AN, L., LIU, S. 2008. Data Updating and Query in Real-time Data Warehouse System, International Conference on Computer Science and Software Engineering, Wuhan, Hubei, Dezembro 2008, 1295-1297.

ZUTERS, J. 2011. Near Real-Time Data Warehousing with Multi-stage Trickle and Flip. Perspectives in Business Informatics Research, volume 90 of Lecture Notes in Business Information Processing, 73–82.