Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Data warehouse e ETL para dados da saúde pública

Uma aplicação prática

Marcos Vinicius do Carmo Sousa

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisora: Profª. Drª. Kelly Rosa Braghetto

São Paulo
20 de janeiro de 2020

"Eu digo algo e geralmente ela acontece. Talvez não no calendário previsto, mas geralmente acontece."
Elon Musk

Agradecimentos

Aos meus pais, pelo amor, incentivo e apoio incondicional.
A minha orientadora, Profa. Dra. Kelly Rosa Braghetto, pela paciência e dedicação em me orientar durante o desenvolvimento desse trabalho.

Resumo

Marcos Vinicius do Carmo Sousa. **Data warehouse e ETL para dados da saúde pública: *Uma aplicação prática***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2019.

O Sistema Único de Saúde (SUS) usa vários Sistemas de Informações em Saúde (SIS) para manter dados de seus usuários e auxiliar diferentes áreas técnicas em todas as esferas públicas. Apesar de os dados armazenados nesses sistemas possuírem relacionamentos entre si no mundo real, esses relacionamentos não são explicitamente coletados e armazenados em suas bases de dados. Problemas como a fragmentação e a duplicação dos dados nos SIS-SUS limitam e dificultam sua análise. Um uso mais inteligente dos dados de saúde pública, tanto para planejar ações necessárias quanto para visualizar a atual situação da saúde nas cidades, depende da integração de dados de SIS-SUS. Tendo isso em vista, este trabalho propõe um repositório unificado de dados de saúde pública, na forma de *data warehouse*, e um processo automático de carga (ETL, de *extraction-transform-load*) para esse repositório. O esquema de dados multidimensional projetado para o *data warehouse* integra dados de três SIS-SUS da Secretaria Municipal de São Paulo, que são: SIM (Sistema de Informações sobre Mortalidade), SINASC (Sistema de Informações de Nascidos Vivos) e SIHSUS (Sistema de Informações Hospitalares do SUS).

Palavras-chave: integração de dados. saúde pública. modelagem multidimensional. data warehouse. ETL.

Abstract

Marcos Vinicius do Carmo Sousa. **Data warehouse e ETL para dados da saúde pública: *Uma aplicação prática***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2019.

The Brazilian Unified Health System uses various Health Information Systems (HIS) to maintain user data and assist different technical areas in all public spheres. Although the data stored in these systems has relationships among themselves in the real world, these relationships are not explicitly collected and stored in their databases. Problems such as data fragmentation and duplication in public HIS limit and hinder their analysis. A smarter use of public health data, both to plan necessary actions and to visualize the current health situation in cities, depends on the integration of HIS data. With this in mind, this work proposes a unified repository of public health data, in the form of a data warehouse, and an extraction-transform-load (ETL) process for this repository. The multidimensional data scheme designed for the data warehouse integrates data from three HIS of the São Paulo Municipal Secretariat, which register data from live births, deaths, and hospitalizations paid by the public health system.

Keywords: data integration. public health. multidimensional modeling. data warehouse. ETL.

Lista de Abreviaturas

ETL	Processo de extração, transformação e carga (*Extract, transform and load process*)
SIS	Sistemas de Informações em Saúde (*Health Information Systems*)
SIM	Sistema de Informações sobre Mortalidade (*Mortality Information System*)
SINASC	Sistema de Informações de Nascidos Vivos (*Live Birth Information System*)
SIHSUS	Sistema de Informações Hospitalares do SUS (*SUS Hospital Information System*)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de Figuras

2.1 Esquema estrela com tabelas dimensionais [Extraída de NAVATHE e ELMASRI, 2016] . . . 6
2.2 Esquema constelação com tabelas dimensionais. [Extraída de NAVATHE e ELMASRI, 2016] . . . 7

4.1 Modelo data warehouse completo . . . 18
4.2 Fato `Atendimento` e suas dimensões . . . 19
4.3 Fato `Pessoa` e suas dimensões . . . 20
4.4 Fato `Parto` e suas dimensões . . . 21
4.5 Fato gestação . . . 23
4.6 Fato `Óbito` e suas dimensões . . . 23
4.7 Fato `Internação` e suas dimensões . . . 25

5.1 Processo de carga completo [Extraída de KIMBALL e CASERTA, 2004] . . . 27
5.2 Tabela comparativa de ferramentas de ETL . . . 30
5.3 Página com o grafo do processo. Extraída de [Apache, 2019] . . . 31

Sumário

1 Introdução **1**
1.1 Contexto e motivação . . . 1
1.2 Problema . . . 2
1.3 Objetivos . . . 3
1.4 Contribuições . . . 3
1.5 Organização do texto . . . 3

2 Fundamentação **5**
2.1 Modelo de dados normalizado . . . 5
2.2 Modelagem multidimensional . . . 6
2.3 Data warehouse . . . 7
2.4 Etapas da construção de DW . . . 8
2.5 O processo ETL . . . 9
2.6 Construindo processo de ETL . . . 10

3 Sistemas de Informações de Saúde do SUS **13**
3.1 Sistema de Informações de Nascidos Vivos (SINASC) . . . 13
3.1.1 Dados do SINASC . . . 13
3.2 Sistema de Informações de Mortalidade (SIM) . . . 14
3.2.1 Dados do SIM . . . 14
3.3 Sistema de Informações Hospitalares(SIH) . . . 15
3.3.1 Dados do SIH . . . 15

4 Data warehouse modelado **17**
4.1 Fato Atendimento . . . 18
4.1.1 Dimensão Estabelecimento . . . 19
4.1.2 Dimensão CID . . . 19
4.1.3 Dimensão CID Secundário . . . 19
4.2 Fato Pessoa . . . 20

4.3 Fato Parto 20
4.3.1 Dimensão Ocupação 21
4.3.2 Dimensão Trabalho Parto Induzido 22
4.3.3 Dimensão Cesária Antes Trabalho Parto 22
4.3.4 Dimensão Nascimento Assistido 22
4.3.5 Dimensão Escolaridade 22
4.3.6 Dimensão Tipo Parto 22
4.3.7 Dimensão Tipo Apresentação RN 22
4.4 Fato Gestação 22
4.5 Fato Óbito 23
4.5.1 Dimensão Tipo Óbito 24
4.5.2 Dimensão Óbito na Gravidez 24
4.5.3 Dimensão Óbito no Puerpério 24
4.6 Fato Internação 24
4.6.1 Dimensão Procedimento 25
4.6.2 Dimensão Caráter Internação 25
4.6.3 Dimensão Especialidade 25

5 Processo de carga desenvolvido 27
5.1 Ferramentas para ETL 28
5.1.1 Kettle 28
5.1.2 Luigi 28
5.1.3 Airflow 29
5.1.4 Ferramenta escolhida 29
5.2 Arquitetura do processo ETL desenvolvido 30
5.3 Reprodução 39

6 Conclusão 41
6.1 Resultados 41
6.2 Próximos passos 42

Apêndices

A Comandos SQL data warehouse 43

Anexos

Referências **53**

Capítulo 1

Introdução

1.1 Contexto e motivação

Em cidades inteligentes, o principal objetivo é gerar eficiência urbana, acelerando o desenvolvimento e melhorando a qualidade de vida dos cidadãos. Um dos seus pilares é o uso inteligente dos dados de saúde, tanto para planejar ações necessárias quanto para visualizar a atual situação da saúde nas cidades.

O Brasil possui centenas de Sistemas de Informações em Saúde (SIS) desenvolvidos para auxiliar áreas técnicas em todas as esferas públicas (municipal, estadual e federal).[Saúde, 2004]

Devido à grande quantidade de sistemas e dados, os SIS do SUS enfrentam uma série de problemas, dentre os quais os principais são:[Fonseca, 2015]:

- Dados fragmentados
  Os dados dos usuários do sistema público de saúde estão fragmentados em diversas bases e sistemas que não possuem integração entre si.
- Dados duplicados
  A decentralização dos sistemas e a falta de integração entre eles gera a necessidade de se registrar um mesmo dado em diferentes bases, fazendo com que os dados sejam duplicados, aumentando o risco de inconsistências e desperdiçando espaço de armazenamento.
- Dados orientados a eventos
  Para ajudar na tomada de decisão nas áreas de epidemiologia ou de gestão pública, alguns sistemas foram criados com a finalidade principal de operacionalizar o pagamento das internações e demais procedimentos realizados nos estabelecimentos do SUS. Por esse motivo os registros nessas bases são orientados a eventos ocorridos e não nos pacientes.
- Obtenção de dados
  Uma grande parte dos SIS ainda obtém seus dados por meio de formulários em papel. Esses dados coletados em papel precisam ser transcritos em formato digital.

Essa operação aumenta a chance de introdução de erros e aumenta o tempo de disponibilização dos dados no sistema.

1.2 Problema

Neste trabalho, o foco são dados da Secretaria Municipal de Saúde de São Paulo (SMS-SP) gerenciados por meio do SIM (Sistema de Informações sobre Mortalidade), do SINASC (Sistema de Informações de Nascidos Vivos) e do SIHSUS (Sistema de Informações Hospitalares do SUS). Os três sistemas são mantidos pelo DATASUS – Departamento de Informática do SUS[1].

Apesar de os registros armazenados nas três bases da SMS-SP possuírem relacionamentos entre si no mundo real, esses relacionamentos não são explicitamente coletados e armazenados nas bases. Não existe uma integração entre os sistemas que as alimentam, o que torna a análise dos dados limitada e difícil.

Há diferentes estratégias que podem ser empregadas na integração desses dados, tais como *data linkage*, integração baseada em ontologias, sistemas mediadores e *data warehouses*. Entretanto, em contextos tais como os dos SIS da SMS-SP (com interfaces de acesso aos dados limitadas e alta latência na resposta às consultas), um *data warehouse* é uma solução que se destaca por ter um bom custo-benefício.

Um *data warehouse* (DW) é um banco de dados que armazena e mantém dados analíticos separados dos bancos de dados orientados a transações para fins de suporte à decisão. Os bancos de dados comuns, relacionais, armazenam dados por um período limitado de tempo antes que os dados percam sua utilidade imediata e sejam arquivados. Por outro lado, os data warehouses tendem a manter anos de dados para permitir a análise de dados históricos.[NAVATHE e ELMASRI, 2016].

Para manter um *data warehouse* atualizado, é necessário um processo automatizado que receba (ou extraia) os novos dados das fontes, trate-os e carregue-os no banco de dados unificado, mantendo a conformidade dos dados. Esse processo automatizado é chamado de ETL (*extract - transform - load*) ou processo de carga.

O projeto de um DW para integrar dados dos SIS da SMS-SP e do seu respectivo processo de ETL envolve os seguintes desafios:

- tratar no modelo do *data warehouse* o problema da falta de identificador único nos registros dos dados dos usuários do SUS;
- detectar e tratar os casos de duplicação de registros;
- lidar com a alta frequência da inserção de novos dados nos SIS, de modo a garantir que os dados estejam disponíveis no DW para analistas pouco tempo após serem incluídos nas fontes;
- lidar com grandes volumes de dados;
- garantir que não ocorra duplicação ou perda de dados durante o processo de carga.

[1] http://datasus.saude.gov.br/

1.3 Objetivos

Este trabalho tem como objetivo propor um esquema de dados de *data warehouse* (baseado no modelo de dados multidimensional) para integrar dados provenientes de SIS-SUS da SMS-SP e um processo de carga (ETL) para manter o *data warehouse* atualizado, ao mesmo tempo em que garante a conformidade e integridade dos dados unificados.

Para isso, diferentes ferramentas de software que podem ser empregadas em processos de ETL foram estudadas e avaliadas, considerando o caso de uso dos SIS-SUS. Um protótipo de solução combinando as ferramentas selecionadas foi implementado e testado usando dados fornecidos pela SMS-SP.

Apesar de terem sido motivados por demandas da SMS-SP, o esquema de DW e o processo ETL desenvolvidos neste trabalho podem ser usados na integração de dados de SIS-SUS de outras cidades ou estados. A solução proposta é genérica e de fácil implantação, uma vez que se baseia em software livre e não depende de uma infraestrutura computacional sofisticada.

1.4 Contribuições

As principais contribuições deste trabalho são as seguintes:

- Um esquema de dados de *data warehouse* (baseado em modelo multimensional) que integra dados de três SIS-SUS da SMS-SP: SIM, SINASC e SIHSUS;
- Uma solução de ETL para o *data warehouse* em questão, implementada com os softwares livres utilizando a ferramenta Airflow [Apache, 2019] e encapsulada em um contêiner *docker* [Docker, 2019a] (para facilitar sua reprodução).

O projeto foi realizado junto ao Instituto Nacional de Ciência e Tecnologia Internet do Futuro para Cidades Inteligentes (InterSCity), que tem uma colaboração com a SMS-SP. Outros alunos de graduação e pós-graduação que participam do InterSCity estiveram envolvidos na modelagem dos dados.

1.5 Organização do texto

O restante deste texto está organizado da seguinte maneira:

- **Capítulo 2**
  Nesse capítulo é descrita a teoria utilizada na construção da solução proposta no trabalho, que está principalmente relacionada a *data warehouses* e seu processo de carga.
- **Capítulo 3**
  Capítulo que descreve as três bases de dados do SIS-SUS consideradas no trabalho.
- **Capítulo 4**
  O Capítulo 4 apresenta o esquema de dados projetado para o DW e as ferramentas que auxiliaram no seu desenvolvimento.

- **Capítulo 5**
  Esse capítulo avalia ferramentas de software livre para o processo de carga e descreve a solução desenvolvida neste trabalho a partir delas.
- **Capítulo 6**
  O Capítulo 6 descreve os resultados do trabalho a discute a reprodutibilidade da solução proposta.

Capítulo 2

Fundamentação

O crescente poder de processamento e a sofisticação das ferramentas e técnicas analíticas resultaram no desenvolvimento dos *data warehouses*. Esses sistemas fornecem grande capacidade de armazenamento, funcionalidades analíticas e bom desempenho em consultas complexas, além dos recursos disponíveis em sistemas de bancos de dados orientados por transações.

W. H. Inmon caracterizou um *data warehouse* como uma coleção de dados que varia no tempo, integrada, não volátil e orientada ao assunto, em apoio à tomada de decisão. Os *data warehouses* fornecem acesso a dados para análises complexas, descoberta de conhecimento e tomada de decisão, suportando operações de alto desempenho sobre os dados de uma organização. *Data warehouses* têm a característica distintiva de serem destinados principalmente a aplicações de suporte à decisão, sendo otimizados para a recuperação de dados, não para processamento de transações de rotina. Em razão dessas características, o uso de data warehouses é uma abordagem interessante para a integração de dados de saúde pública. [NAVATHE e ELMASRI, 2016]

Um *data warehouse* é geralmente alimentado por meio de um processo de extração dos dados das fontes heterogêneas, tratamento dos mesmos (para limpeza, uniformização, etc.) e ingestão no repositório unificado. Esse processo é chamado de ETL – *Extract, Transform and Load*. [KIMBALL e CASERTA, 2004]

Nas próximas seções, o modelo usado na organização dos dados em data warehouses e o processo de ETL serão caracterizados de forma mais aprofundada.

2.1 Modelo de dados normalizado

O modelo relacional representa o banco de dados como uma coleção de relações. Informalmente, cada relação se assemelha a uma tabela de valores, cada linha da tabela representa uma coleção de valores de dados relacionados. Uma linha representa um fato que normalmente corresponde a uma entidade ou relacionamento do mundo real. [NAVATHE e ELMASRI, 2016].

A normalização de dados pode ser considerada um processo de análise dos esquemas

de relações do modelo relacional para alcançar as propriedades desejáveis de (1) minimizar redundância e (2) minimizar as anomalias de inserção, exclusão e atualização.[NAVATHE e ELMASRI, 2016]

Ainda que os dados em uma tabela normalizada sejam de uma forma muito pura e minimizem a redundância, podem dificultar muito analistas a navegarem e fazerem suas análises nas bases de dados.

2.2 Modelagem multidimensional

Modelos multidimensionais (também chamado de modelos dimensionais) aproveitam os relacionamentos inerentes aos dados para preencher dados em matrizes multidimensionais chamadas cubos de dados. (Estes podem ser chamados hipercubos se tiverem mais de três dimensões.) Para dados que se prestam à modelagem multidimensional, o desempenho da consulta em matrizes multidimensionais pode ser muito melhor do que no modelo de dados relacionais. Três exemplos de dimensões em um data warehouse corporativo são: os períodos fiscais, produtos e regiões da corporação. [NAVATHE e ELMASRI, 2016]

O modelo multidimensional envolve dois tipos de tabelas: tabelas de dimensões e tabelas de fatos:

- Uma tabela de dimensões consiste em tuplas de atributos da dimensão.
- Uma tabela de fatos pode ser considerada como tendo tuplas, uma por um fato registrado, que geralmente contém variáveis medidas ou observadas e referências para as tabelas de dimensões.

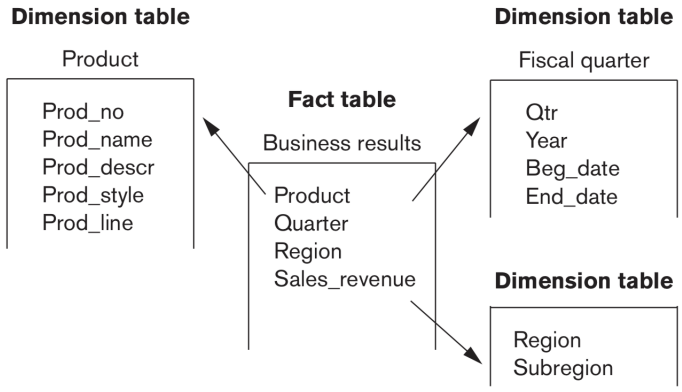

Figura 2.1: *Esquema estrela com tabelas dimensionais [Extraída de NAVATHE e ELMASRI, 2016]*

Dois esquemas multidimensionais comuns são o esquema em estrela e o esquema de floco de neve. O esquema em estrela consiste em uma tabela de fatos com uma única tabela para cada dimensão. O esquema de floco de neve é uma variação no esquema em estrela, na qual as tabelas dimensionais de um esquema em estrela são organizadas em uma hierarquia, normalizando-as. A figura 2.1 mostra um exemplo de uma tabela fato com 3 tabelas de dimensão que pode ser visto como um esquema multidimensional. [NAVATHE e ELMASRI, 2016]

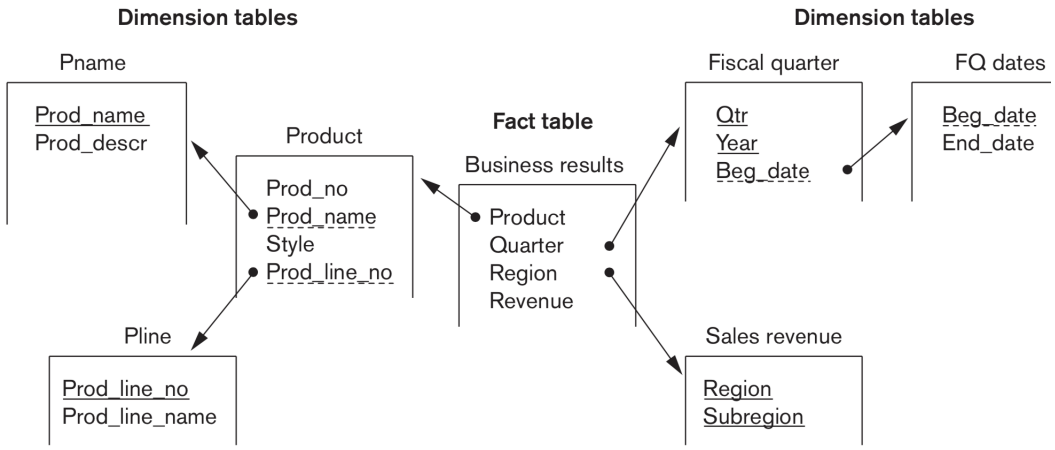

Figura 2.2: *Esquema constelação com tabelas dimensionais. [Extraída de NAVATHE e ELMASRI, 2016]*

Uma constelação de fatos é um conjunto de tabelas de fatos que compartilham algumas tabelas de dimensões. A figura 2.2 mostra uma constelação de fatos com duas tabelas de fatos, resultados e previsão de negócios. Eles compartilham a tabela de dimensões chamada produto. [NAVATHE e ELMASRI, 2016]

Para superar os problemas de performance em grandes consultas em um data warehouse, é amplamente utilizado um modelo dimensional para o banco de dados. O modelo dimensional traz uma maneira para melhorar as performances de buscas sem afetar a integridade dos dados, entretanto, como consequência de uma performance melhor vem com o custo de espaço extra. Um banco de dados dimensional geralmente requer muito mais espaço que o tradicional banco relacional, mas ainda assim é vantajoso visto que o custo de armazenamento vem decaindo ao longo dos anos.

Normalização é uma técnica que envolve duplicar dados em um ou mais tabelas para minimizar ou eliminar "SQL joins"muito demorados. Nesses casos deve-se adequar métodos para garantir que os dados duplicados estejam sempre consistentes em todas as tabelas para evitar problemas de integridade nos dados.

2.3 Data warehouse

Um dos ativos mais importantes de qualquer organização é a informação (os dados). Esse ativo é quase sempre usado para dois principais propósitos: manutenção de registros operacionais e tomada de decisão analítica. Em ambos os casos torna-se necessário organizar os dados de maneira eficiente e de fácil acesso.

Com isso, Ralph Kimball [KIMBALL e CASERTA, 2004] autor do livro The Data Warehouse Toolkit (Wiley, 1996) elenca princiapis características de um data warehouse completo, elas são:

O sistema data warehouse deve tornar as informações facilmente acessíveis. O conteúdo do sistema data warehouse deve ser compreensível, os dados devem ser intuitivos e óbvios para o usuário comercial, não apenas para o desenvolvedor. As estruturas

e os rótulos dos dados devem imitar os processos e o vocabulário dos usuários de negócios. As ferramentas e aplicativos de inteligência de negócios que acessam os dados devem ser simples e fáceis de usar. Eles também devem retornar os resultados da consulta ao usuário com tempos de espera mínimos. Podemos resumir esse requisito simplesmente dizendo simples e rápido.

O sistema data warehouse deve apresentar informações de forma consistente.
Os dados no sistema data warehouse devem ser críveis. Os dados devem ser cuidadosamente reunidos a partir de uma variedade de fontes, limpos, com garantia de qualidade e liberados somente quando adequados ao consumo do usuário. A consistência também implica que rótulos e definições comuns para o conteúdo do sistema data warehouse sejam usados em fontes de dados. Se duas medidas de desempenho tiverem o mesmo nome, elas deverão significar a mesma coisa. Por outro lado, se duas medidas não significam a mesma coisa, elas devem ser rotuladas de maneira diferente.

O sistema data warehouse deve apresentar informações indexadas de forma temporal.
Como o sistema data warehouse é usado com mais intensidade nas decisões operacionais, os dados brutos podem precisar ser convertidos em informações acionáveis em horas, minutos ou até segundos. Logo, todos os dados inseridos no data warehouse devem ser indexados por tempo.

O sistema data warehouse deve ser um local seguro que protege os dados.
Os ativos mais valiosos (dados) de uma organização são armazenados no data warehouse de dados. No mínimo, um data warehouse provavelmente contém informações sobre o que está sendo vendido, a quem e a que preço - detalhes que são potencialmente prejudiciais nas mãos de pessoas erradas. O sistema data warehouse deve controlar efetivamente o acesso às informações confidenciais da organização.

O sistema data warehouse deve servir como base autorizada e confiável para melhorar a tomada de decisões.
O data warehouse deve ter os dados corretos para dar suporte à tomada de decisão. Os resultados mais importantes de um sistema data warehouse são as decisões tomadas com base nas evidências analíticas apresentadas; essas decisões favorecem ao impacto no valor comercial atribuidos no sistema data warehouse.

2.4 Etapas da construção de DW

Ao construir um data warehouse, os desenvolvedores devem ter uma visão ampla do uso antecipado do data warehouse. Não há como antecipar todas as consultas ou análises possíveis durante a fase de design. No entanto, o design deve suportar especificamente consultas ad hoc; ou seja, acessar dados com qualquer combinação de valores para os atributos que seriam significativos nas tabelas de dimensões ou tabelas fatos. Por exemplo, uma empresa de marketing exigiria maneiras diferentes de organizar o data warehouse do que uma instituição de caridade sem fins lucrativos. Um esquema apropriado deve ser escolhido para refletir o uso antecipado. [KIMBALL e CASERTA, 2004]

A aquisição de dados para o data warehouse envolve as seguintes etapas:

1. Os dados devem ser extraídos de várias fontes heterogêneas; por exemplo, bancos de dados ou outras fontes de dados, como API's, planilhas de CSV, arquivos txt.

2. Os dados devem ser formatados para consistência dentro do data warehouse. Nomes, significados e domínios de dados de fontes não relacionadas devem ser reconciliados.

3. Os dados devem ser limpos para garantir a validade. A limpeza de dados é um processo complexo e envolvido que foi identificado como o maior componente exigente em mão-de-obra da construção do data warehouse. Para dados de entrada, a limpeza deve ocorrer antes que os dados sejam carregados no data warehouse. Como os dados de entrada devem ser examinados e formatados de maneira consistente, os construtores de data warehouse devem aproveitar esta oportunidade para verificar cada entrada quanto à validade e qualidade. É difícil automatizar o reconhecimento de dados errôneos e incompletos, e a limpeza que requer correção automática de erros pode ser ainda mais difícil.

4. Os dados devem ser ajustados ao modelo de dados do data warehouse. Os dados das várias fontes devem ser representados no modelo de dados do data warehouse. Os dados podem precisar ser convertidos de bancos de dados relacionais, orientados a objetos ou herdados (rede e / ou hierárquicos) em um modelo dimensional.

5. Os dados devem ser carregados no data warehouse. O grande volume de dados no data warehouse torna o carregamento dos dados uma tarefa significativa. São necessárias ferramentas de monitoramento de cargas, bem como métodos para recuperar cargas incompletas ou incorretas. Com o enorme volume de dados no data warehouse, a atualização incremental é geralmente a única abordagem viável.

Como foi dito, os bancos de dados devem encontrar um equilíbrio entre a eficiência no processamento de transações e o suporte a requisitos de consulta (solicitações ad hoc do usuário), mas um data warehouse geralmente é otimizado para acessar as necessidades de um tomador de decisão. O armazenamento de dados em um data warehouse reflete essa especialização e envolve os seguintes processos:

- Armazenar os dados de acordo com o modelo de dados do data warehouse
- Criar e manter estruturas de dados necessárias
- Criar e manter caminhos de acesso apropriados
- Fornecer dados variantes no tempo à medida que novos dados são adicionados
- Suportar atualização/incremento de dados do data warehouse

2.5 O processo ETL

O processo de ETL (Extract-Transform-Load), em português extrair, transformar e carregar é a base do data warehouse. Um sistema ETL projetado adequadamente extrai

dados dos sistemas de origem, aplica padrões de qualidade e consistência dos dados, conforma dados para que fontes separadas possam ser usadas juntas e, finalmente, entrega dados em um formato pronto para apresentação, para que os desenvolvedores de aplicativos possam criar aplicativos e usuários finais possam tomar decisões. Embora a construção do sistema ETL seja uma atividade de bastidores que não seja muito visível para os usuários finais, ela consome facilmente 70% dos recursos necessários para a implementação e manutenção de um data warehouse típico. [KIMBALL e CASERTA, 2004]

O processo ETL agrega valor significativo aos dados. É muito mais do que uma tentativa de obter dados dos sistemas de origem e entrar no data warehouse. Especificamente, o processo ETL:

- Remove erros e corrige dados ausentes
- Captura o fluxo de dados transacionais
- Ajusta os dados de várias fontes para serem usados juntos
- Estrutura os dados para serem utilizados pelas ferramentas do usuário final

2.6 Construindo processo de ETL

Construir um processo ETL (Extract-Transform-Load) é muito desafiador pois ao desenvolver o ETL deve-se atentar a diversos fatores: requisitos de negócios, os formatos e as deficiências dos dados de origem, os sistemas legados e as constantes mudanças e necessidades dos usuários finais [KIMBALL e CASERTA, 2004].

Na maioria das implementações, o ETL é constituído de 4 etapas fundamentais, e são elas:

- Extração. Os dados brutos (não tratados) vindo de sistemas são geralmente escritos diretamente no disco com nenhuma ou mínima estruturação antes que alguma transformação significativa aconteça. Em contrapartida, dados que vem de fontes estruturadas (como datasets em XML, CSV, ou TXT, e também bancos de dados relacionais) são geralmente gravados em arquivos simples ou tabelas relacionais em um banco comumente chamado de *stage* (área de transição ou área temporária), onde são convertidos para um único formato. Isso permite que a extração seja simples e mais rápida possível, permitindo também maior flexibilidade para reiniciar o processo de extração se houver alguma interrupção.
- Limpeza. Na maioria dos casos, o nível de qualidade dos dados aceitável para o sistemas de origem é diferente da qualidade exigida pelos *data warehouses*. O processamento de qualidade de dados pode envolver muitas etapas discretas, incluindo a verificação de valores válidos (por exemplo se CEP está presente e é na faixa de valores válidos), garantindo consistência entre os valores, removendo entradas duplicadas (por exemplo o mesmo cliente aparece duas vezes com atributos pouco diferentes), e verificar se regras e procedimentos de negócios complexos foram realizadas (por exemplo o cliente de uma empresa de cartão de crédito tem a extensão estendida associada ao valor de crédito)

- Conformidade. A conformação de dados é necessária sempre que duas ou mais origens de dados são mescladas no data warehouse.
- Entrega. O ponto principal do processo de ETL é fazer com que os dados estejam prontos para consulta. O último atrás é fisicamente estruturar os dados em um conjunto de esquemas simples e simétricos conhecidos como modelos dimensionais, ou equivalentemente, esquemas em estrela. Estes esquemas reduzem significativamente os tempos de consulta e simplifica o desenvolvimento de uma aplicação.

Capítulo 3

Sistemas de Informações de Saúde do SUS

Os Sistemas de Informações de Saúde (SIS) mantidos pelo SUS utilizam várias bases de dados. A maior parte desses sistemas registram eventos relacionados ao cuidado, tais como os nascimentos e óbitos. Existem também sistemas que têm com objetivo auxiliar a gestão dos gastos públicos, tais como os sistemas para notificação de internações hospitalares.

Este trabalho se dedica à construção de um data warehouse para a integração de dados provenientes dos sistemas SINASC, SIM e SIH, que serão brevemente descritos nas seções a seguir. Para a construção do *data warehouse*, serão usados apenas um subconjunto das centenas de atributos que compõem as bases de dados desses sistemas. Esse subconjunto de atributos de interesse para o trabalho foi definido pela Secretaria Municipal de Saúde de São Paulo (SMS-SP).

3.1 Sistema de Informações de Nascidos Vivos (SINASC)

O Sistema de Informações de Nascidos Vivos (SINASC) foi oficialmente implantado em 1990. Ao ocorrer um nascido vivo, a sua notificação no sistema é obrigatória, independente do mesmo ter acontecido em um hospital da rede SUS ou não.

3.1.1 Dados do SINASC

Os dados do SINASC fornecidos pelo SMS-SP são datados de 2016, disponibilizados em um arquivo de extensão .CSV, onde os campos com informações pessoais foram anonimizados. São 195138 registros com 98 colunas indicando diversas características do nascimento registrado.

As principais informações a serem consideradas na modelagem do *data warehouse* são:

- **declarações de nascido vivo (DN)**
  um identificador numérico que é impresso no formulário de preenchimento da DN. Este número é único e gerado pelo Ministério da Saúde. A partir dele é lavrada a Certidão de Nascimento emitida por Cartório de Registro Civil Brasil et al. [2011a]. A DN possui 52 variáveis decompostas em oito blocos contextuais, abrangendo dados estatísticos, sócio-demográficos e epidemiológicos.

- **informações do nascido vivo**
  apresenta informações do nascido vivo: data e hora de nascimento, sexo do nascido vivo, peso, raça, e informações do nascimento como: malformação, possível necessidade de retroalimentação, entre outros.

- **informações de localização**
  um conjunto de campos que informam a localização da ocorrência (nascido vivo), relativo ao estabelecimento, residência da mãe e cartório. Os principais campos são: CEP, município e bairro.

- **informações da mãe**
  características gerais da mãe, como: nome, raça, data de nascimento, escolaridade, estado civil, quantidade de filhos nascidos vivos, quantidade de filhos nascidos mortos, quantidade de abortos, entre outros.

- **informações do parto e da gestação**
  apresenta características detalhadas do parto, como: informações do trabalho de parto realizado, tipo de parto (cesária ou normal), apresentação do trabalho de parto, classificação Robson do nascido, informações da avaliação Apgar e características da gestação e de possíveis gestações anteriores, como: quantidade de gestações, o mês da gestação, semana da gestação, informações sobre a consulta pré-natal e outros.

- **informações do responsável**
  apresenta informações básicas do responsável, podendo ser o pai do nascido vivo ou pai/mãe da mãe do nascido vivo, como: nome, função, idade do responsável além de informações de localização.

3.2 Sistema de Informações de Mortalidade (SIM)

O Sistema de Informações de Mortalidade (SIM) é o sistema que registra, desde 1976, as declarações de óbito (DO) em todo o território nacional. Assim como ocorre com a DN, a DO é obrigatória para a confecção de um documento legal, a Certidão de Óbito. A DO não contém um campo para o identificador do Cadastro Nacional de Saúde(CNS), considerado o principal identificador para os usuários do SUS, contendo apenas um identificador único fornecido pelo Ministério da Saúde.

3.2.1 Dados do SIM

Os dados do SIM fornecidos pela SMS-SP são datados de 2016 e 2017, disponibilizado em um arquivo de extensão .CSV, nos quais os campos com informações pessoais foram

anonimizados. São 182200 registros com 53 colunas indicando diversas características do óbito registrado.

As principais informações a serem consideradas na modelagem são:

- **informações do óbito**
  informações gerais sobre o óbito: tipo de óbito, data e hora do óbito, local de ocorrência, tipo de estabelecimento de ocorrência
- **informações da pessoa levada ao óbito**
  informações gerais como: nome do indivíduo, data de nascimento, sexo, raça (cor), estado civil, escolaridade e ocupação do falecido, além também de informações de localização (residência).
- **informações da mãe da pessoa levada ao óbito**
  informações gerais como: nome, idade, escolaridade e ocupação da mãe
- **informações de localização**
  um conjunto de campos que informam a localização da ocorrência (óbito), onde os principais campos são: CEP, município e bairro.
- **informações de causa da morte**
  informações que descrevem a causa da morte, os campo principais são: o CID da causa básica (inicial observada), CID da causa da morte (determinada ou confirmada) o tempo aproximado da morte e descrição da explicação da morte.

3.3 Sistema de Informações Hospitalares(SIH)

O Sistema de Informações Hospitalares (SIH) opera desde 1990, com características herdadas do antigo dispositivo de registro de Autorização de Internação Hospitalar (AIH), de 1979, que vigorava antes da criação do SUS. O SIH trouxe muitas características com um viés financeiro, uma vez que tem por finalidade informar procedimentos e internações hospitalares que serão custeadas pelo SUS. Assim, sua base de dados possui informações administrativas, demográficas, geográficas, clínicas e principalmente, financeiras.[Saúde, 2004]

3.3.1 Dados do SIH

Os dados do SIH fornecidos pela SMS-SP são datados de 2015, 2016 e 2017, disponibilizados em um arquivo de extensão .CSV, nos quais os campos com informações pessoais foram anonimizados. São 2079345 registros com 115 colunas indicando diversas características de internações.

As principais informações a serem consideradas na modelagem são:

- **informações do paciente**
  informações básicas do paciente, tais como: nome, data de nascimento, sexo, raça (cor), nacionalidade, documento de registro, grau de instrução e identificador do Cadastro Nacional de Saúde(CNS), além de informações de localização do paciente (residência).

- **informações dos responsáveis (mãe e pai)**
  informações básicas, como: nome, documento dos responsáveis e localização.
- **informações de localização**
  um conjunto de campos que informam a localização da internação, onde os principais campos são: CEP, município e bairro.
- **informações de internação**
  informações relativas a internação, tais como: modalidade de internação, complexidade da internação, código diagnóstico primário da internação (CID [1]), código diagnóstico secundário da internação (CID), total de diárias na internação e na UTI, data de entrada e data de saída do paciente na internação, especialidade do leito, identificador da internação e tipo de financiamento (quem vai pagar: estado, união ou município).
- **informações de procedimentos**
  informações relativas aos procedimentos executados durante as internações, os principais atributos são: complexidade do procedimento, grupo do procedimento, código do procedimento realizado, quantidade de procedimentos realizados e código do procedimento solicitado, além de informações de custo do procedimento, como: valor, prestador (próprio ou terceiro) e rateio (geralmente esferas do poder público).
- **informações de gestação**
  informações específicas de internação relacionada a gestação: indicador de gestação de risco, meses de gestação na UTI neonatal, situação da gestação, identificação da gestante no pré-natal, entre outros.
- **informações de parto**
  informações específicas de internação relacionada ao parto: quantidade de nascidos vivos, quantidade de nascidos mortos, quantidade de nascidos que tiveram alta, quantidade de nascidos que foram transferidos, CID de indicação de laqueadura ou de vasectomia, indicação de método contraceptivos, entre outros.

[1] CID – Classificação Internacional da Doença, definida pela Organização Mundial da Saúde.

Capítulo 4

Data warehouse modelado

A construção de um *data warehouse* é composto pelas etapas de modelagem dos dados, decisão e criação de arquitetura de banco de dados para o modelo e implementação do processo de carga (ETL) nos bancos de dados do *data warehouse*. Este capítulo apresenta o esquema de dados modelado para o data warehouse proposto neste trabalho.

O esquema de *data warehouse* proposto neste trabalho é baseado no modelo de dados multidimensional que foi fundamentado no capítulo 2 e envolve dois tipos de tabelas: tabelas de dimensões e tabelas de fatos.

Para decidir os fatos do esquema multidimensional para a construção do data warehouse foi feito uma análise dos principais atributos dos dados disponibilizados pelos SMS-SP, que foram descritos no capítulo 3.

Essa análise levou em consideração os principais atributos de interesse do SMS-SP, que na maioria dos casos está relacionado com as consultas que eles pretendem realizar. As dimensões foram escolhidas de maneira a aumentar a performance das consultas. Ainda na análise, alguns atributos que pertenciam aos fatos foram transformados em dimensões pois o seu domínio é limitado e algumas vezes compartilhados com outros fatos.

Através dessa análise foi possível extrair seis fatos principais que determinam o esquema proposto, são eles: `Atendimento`, `Pessoa`, `Parto`, `Gestação`, `Óbito` e `Internação`. Na Figura 4.1 é possível observar todos os fatos e suas respectivas dimensões

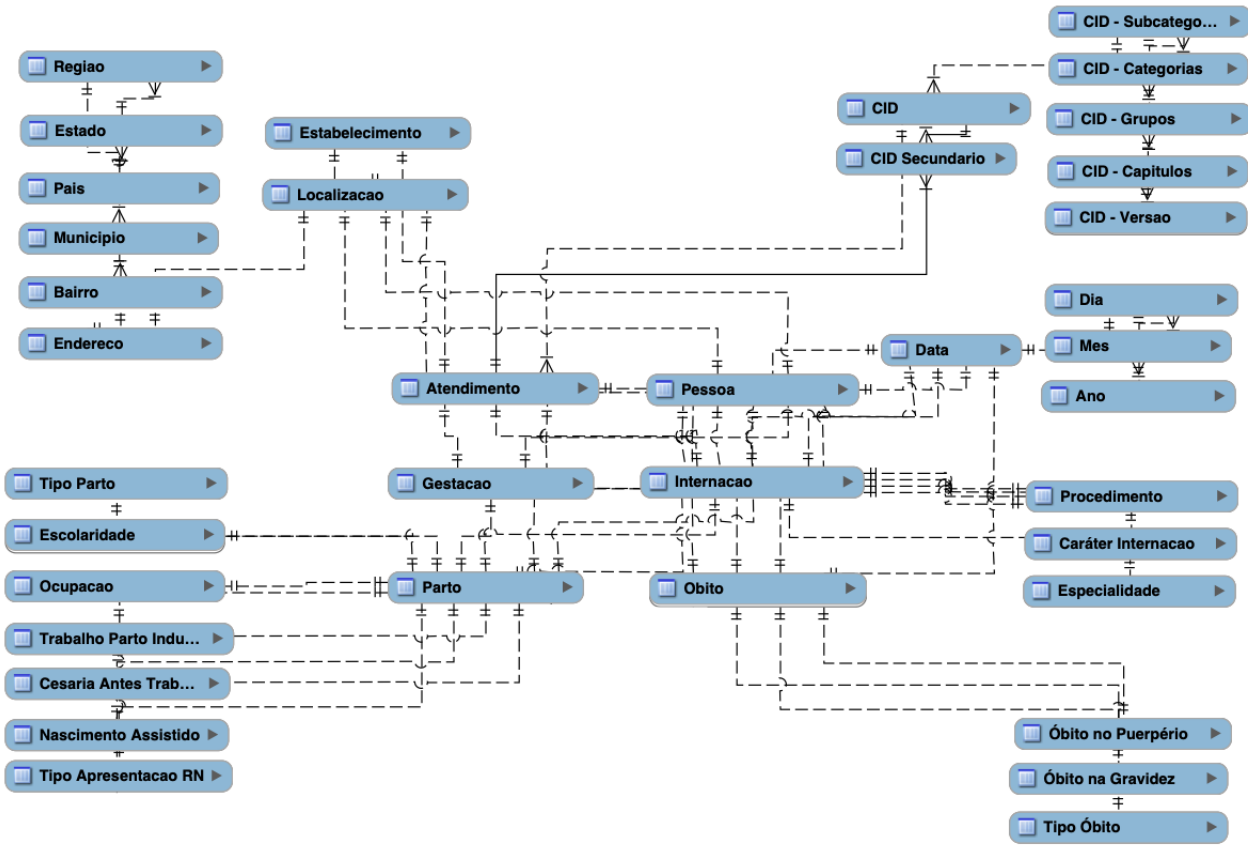

Figura 4.1: *Modelo data warehouse completo*

A seguir será descrito com mais detalhes cada um dos fatos propostos e suas respectivas dimensões. Os comandos SQL para criação de cada um dos fatos e dimensões pode ser visto no apêndice A.

4.1 Fato Atendimento

Os dados que compõem o fato `Atendimento` são principalmente provenientes do SIH-SUS (Sistema de Informações Hospitalares do SUS), que registra as internações dos pacientes do SUS. Mas alguns dados de `Atendimento` podem ser provenientes do SIM (Sistema de Informações sobre Mortalidade) caso o paciente venha a falecer durante um atendimento.

Esse fato é importante para a SMS-SP realizar consultas e resumos de atendimentos, com distinção entre estabelecimentos e CID.

Nesse fato, as principais dimensões tem características que informam a localidade e o motivo do atendimento, essas características são bem definidas nas dimensões: Estabelecimento (informações do estabelecimento) e CID (informações do diagnóstico inicial do paciente). Onde CID é a Classicação Internacional da Doença, definida pela Organização Mundial da Saúde.

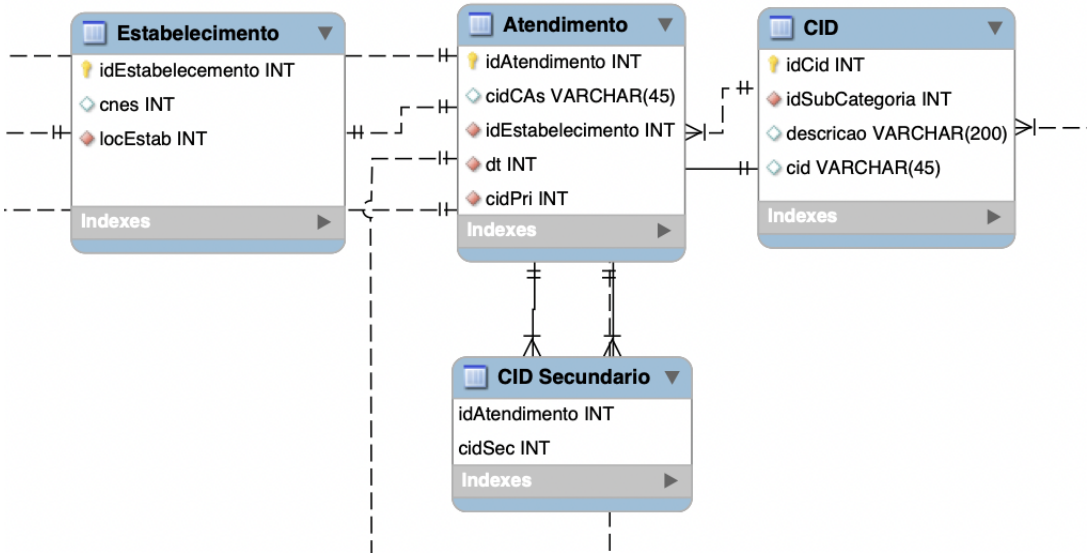

Figura 4.2: *Fato* `Atendimento` *e suas dimensões*

A Figura 4.2 exemplifica as relações entre as dimensões com o fato `Atendimento`. O fato tem duas chaves estrangeiras principais: a `cidPri`, que estabelece um relacionamento com a dimensão `CID`, e o `idEstabelecimento`, que faz a mesma coisa com a dimensão `Estabelecimento`. A última chave estrangeira em `Atendimento`, `dt`, expressa um relacionamento com a dimensão `Data`, que é considerada uma dimensão auxiliar e caracteriza informações de tempo em dia, mês e ano.

4.1.1 Dimensão Estabelecimento

Essa dimensão contém informações básicas do estabelecimento como: cnes (Cadastro Nacional de Estabelecimentos de Saúde) e o locEstab que informa a localização do estabelecimento. A única chave estrangeira nessa dimensão é a `locEstab`, que expressa um relacionamento com a dimensão `Localização`, que é considerada uma dimensão auxiliar e caracteriza informações de localização.

4.1.2 Dimensão CID

Essa dimensão armazena informações da Classificação Internacional de Doenças apartir de uma avaliação inicial. A única chave estrangeira dessa dimensão é a `idSubCategoria`, que expressa um relacionamento com a dimensão `CID`, que nesse caso é a `Sub Categoria do CID`.

4.1.3 Dimensão CID Secundário

Essa dimensão armazena as mesmas informações que a dimensão CID, porém na prática serve para indicar o CID secundário, que geralmente é preenchido após avaliação médica.

4.2 Fato Pessoa

Os dados que compõem o fato Pessoa são provenientes dos três sistemas fontes do *data warehouse*: SIH, SIM e SINASC.

A maior parte dos dados que compõem um registro no fato Pessoa são provenientes do sistema SINASC (Sistema de Informações de Nascidos Vivos), uma vez que em um registro desse sistema existem no mínimo duas pessoas envolvidas: o nascido vivo e a mãe, podendo em alguns casos haver informações do pai também.

Dados proveniente do sistema SIM geram até um registro no fato Pessoa, o responsável, enquanto que no sistema SIH podem gerar registros de até três pessoas de maneira indireta, caso o atendimento seja um parto.

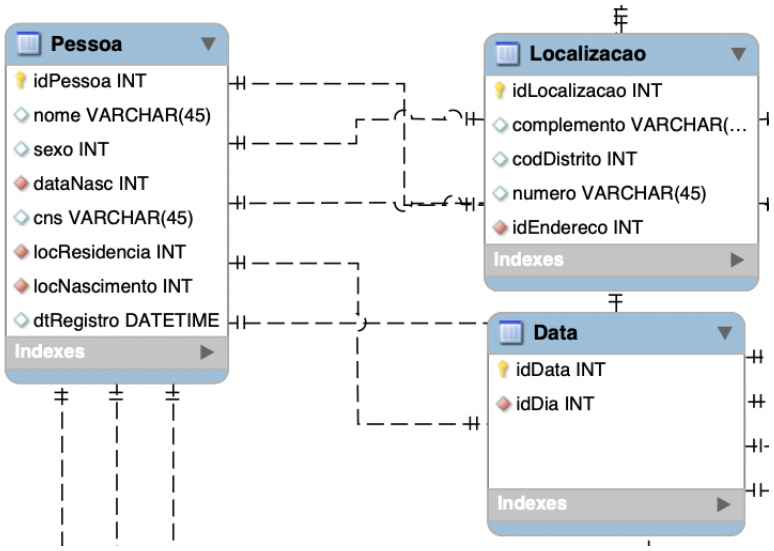

Figura 4.3: *Fato* Pessoa *e suas dimensões*

Conforme a figura 4.3, as principais dimensões associadas ao fato Pessoa são: Localização e Data que fazem um papel de dimensões auxiliares, pois não são exclusivas desse fato.

Onde a dimensão Localização tem informações essenciais para definir o local de nascimento e residencia no registro do fato Pessoa. Enquanto que dimensão Data tem informações da data de nascimento.

O fato Pessoa tem três chaves estrangeiras: dataNasc, que expressa um relacionamento com a dimensão Data, e locResidencia e locNascimento, que expressam relacionamentos com a dimensão Localização.

4.3 Fato Parto

Os dados que compõem o fato Parto são provenientes dos sistemas SIH e SINASC. Em um parto, é considerado que há um nascido vivo, por isso não há intersecção direta com os dados do sistema SIM. Nesse fato, as principais dimensões são: Ocupação, Trabalho

`Parto Induzido`, `Cesaria Antes Trabalho Parto`, `Nascimento Assistido`, `Escolaridade`, `Tipo Parto` e `Tipo Apresentacao RN`.

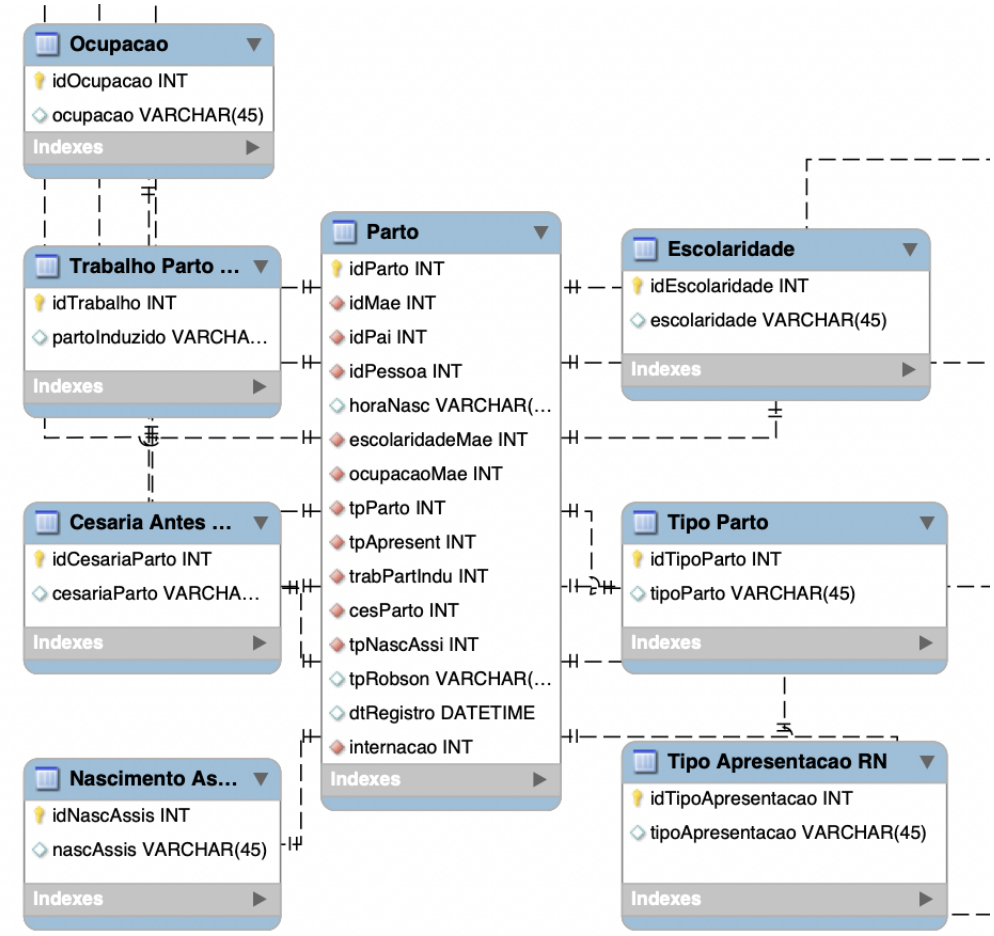

Figura 4.4: *Fato* `Parto` *e suas dimensões*

A Figura 4.4 exemplifica as relações entre as dimensões e o fato.
Esse fato é importante para a SMS-SP realizar consultas sobre o parto e resumos de suas características.

4.3.1 Dimensão Ocupação

A `Ocupação` é uma lista de atributos que indica a ocupação da mãe. O fato `Parto` se relaciona a ela por meio de seu atributo `ocupacaoMae`.

A `Ocupação` foi modelada como uma dimensão da tabela `Parto` porque a única informação sobre ocupação que se tem é a da mãe em um registro do sistema SINASC.

Se `Ocupação` fosse modelada como uma dimensão da tabela `Pessoa` (algo que parece mais natural), haveria diversos registros em `Pessoa` com valor nulo no atributo correspondente a essa dimensão.

4.3.2 Dimensão Trabalho Parto Induzido

A dimensão `Trabalho Parto Induzido` é uma tabela que contém um pequeno número de características que indicam se o parto foi induzido, essa dimensão é referenciada pela tabela fato `Parto`.

4.3.3 Dimensão Cesária Antes Trabalho Parto

A dimensão `Cesária Antes Trabalho Parto` é uma tabela que contém informações sobre o tipo de cesária que ocorreu, referenciada pela tabela fato `Parto`.

4.3.4 Dimensão Nascimento Assistido

A dimensão `Nascimento Assistido` é uma tabela que indica quem fez o nascimento: médico, enfermeira, obstetriz ou outros. Ela é referenciada pela tabela fato `Parto`.

4.3.5 Dimensão Escolaridade

A dimensão `Escolaridade`, assim como a `Ocupação`, naturalmente seria uma dimensão de `Pessoa`. Mas para evitar o excesso de dados nulos, optou-se por modelar `Escolaridade` como uma dimensão de `Parto`, já que essa informação só existe para mães nos registros do sistema SINASC.

4.3.6 Dimensão Tipo Parto

A dimensão `Tipo Parto`, é uma tabela que indica o tipo de parto ocorrido: vaginal, cesário ou ignorado. Ela é referenciada pela tabela fato `Parto`.

4.3.7 Dimensão Tipo Apresentação RN

A dimensão `Tipo Apresentacao RN`, apresenta informações de como se apresenta o nascido vivo no parto: cefálico, transversa, podálica. Ela é referenciada pela tabela fato `Parto`.

4.4 Fato Gestação

Os dados que compõem o fato `Gestação` são principalmente proveniente do SIH-SUS. Nesse fato não há dimensões.

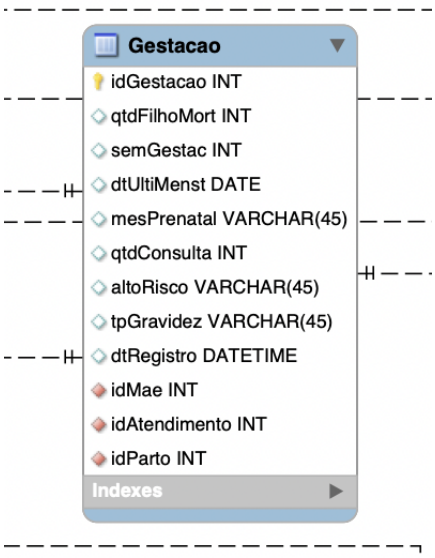

Figura 4.5: *Fato gestação*

Gestação é uma tabela de normalizada, por isso não depende de dimensões. Essa decisão de modelagem foi motivada pela falta de parâmetros para os seus atributos. Observe na Figura 4.5 que os atributos são, em geral, números inteiros ou cadeias de caracteres sem restrições. Entretanto, o fato Gestação referencia outros fatos como Atendimento, Parto e Pessoa.

4.5 Fato Óbito

Os dados que compõem o fato Óbito são provenientes do SIM (Sistema de Informações sobre Mortalidade). Nesse fato as principais dimensões são: Óbito no Puerpério, Óbito na Gravidez e Tipo Óbito, além das dimensões Data e Localização.

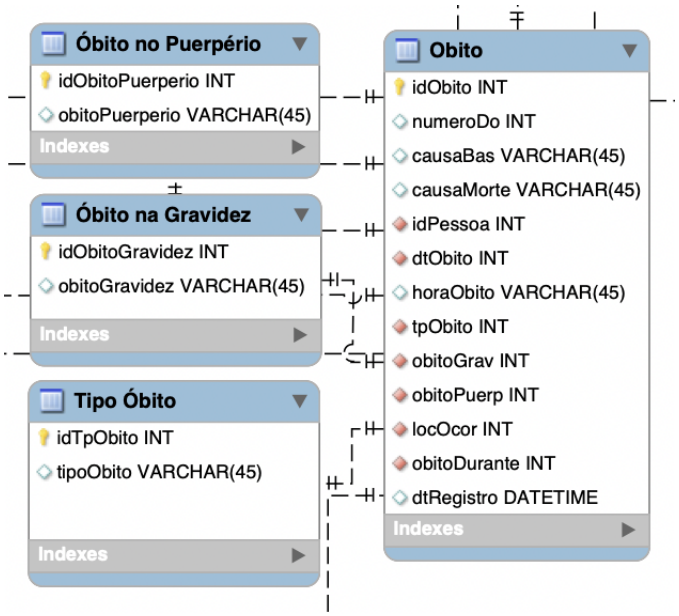

Figura 4.6: *Fato* Óbito *e suas dimensões*

As principais dimensões se relacionam pelas chaves estrangeiras: tpObito, obitoGrav e obitoPuerp. O fato Óbito, conforme na Figura 4.6, tem duas chaves estrangeiras que estabelecem relacionamentos com outros fatos, como o idPessoa e obitoDurante que, respectivamente, referenciam o fato Pessoa e o fato Atendimento. As chaves estrangeiras dtObito e locOcor são de dimensões auxiliares indicando, respectivamente, Data e Localização.

4.5.1 Dimensão Tipo Óbito

A dimensão Tipo Óbito é uma tabela que indica o tipo de óbito ocorrido: fetal ou não fetal. Ela é referenciada pela tabela fato Óbito.

4.5.2 Dimensão Óbito na Gravidez

A dimensão Óbito na Gravidez é uma tabela que indica se ocorreu ou não óbito na gravidez, referenciada pela tabela fato Óbito.

4.5.3 Dimensão Óbito no Puerpério

A dimensão Óbito no Puerpério, é uma tabela que indica se o óbito ocorreu até 42 dias após o parto, de 43 dias até 1 ano ou se não ocorreu óbito no puerpério. Ela é referenciada pela tabela fato Óbito.

4.6 Fato Internação

Os dados que compõem o fato Internação são proveniente do SIH-SUS (Sistema de Informações Hospitalares do SUS), que registra as internações dos pacientes do SUS. Nesse fato as principais dimensões são: Procedimento, Caráter Internação e Especialidade.

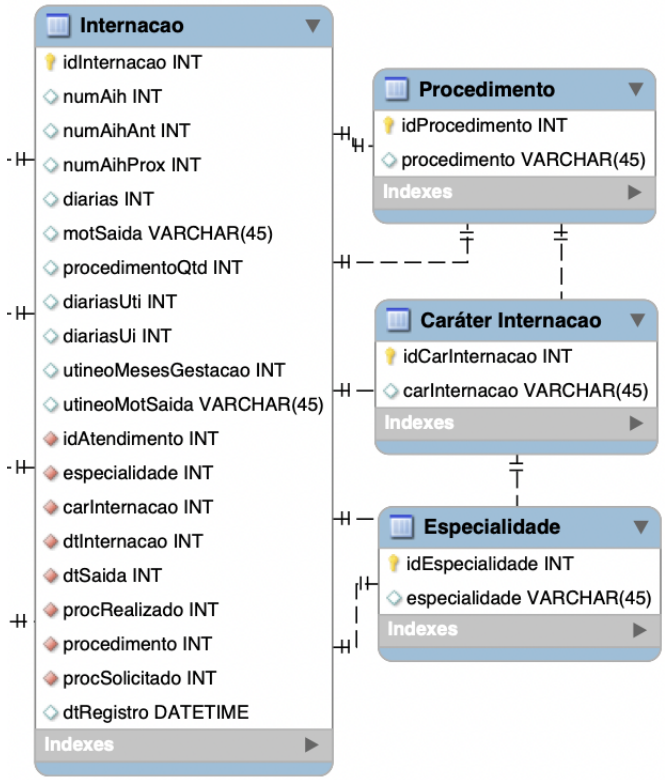

Figura 4.7: *Fato* `Internação` *e suas dimensões*

A Figura 4.7 exemplifica as relações entre as dimensões e o fato.

4.6.1 Dimensão Procedimento

A dimensão `Procedimento` é uma tabela que indica o procedimento realizado na internação, referenciada pela tabela fato `Internação`.

4.6.2 Dimensão Caráter Internação

A dimensão `Caráter Internação` é uma tabela que indica o a modalidade da internação, referenciada pela tabela fato `Internação`.

4.6.3 Dimensão Especialidade

A dimensão `Especialidade` é uma tabela que indica a especialidade da internação, referenciada pela tabela fato `Internação`.

Capítulo 5

Processo de carga desenvolvido

O processo de carga em um data warehouse envolve as etapas de modelagem dos dados (realizado no capítulo 4), decisão e criação de arquitetura de banco de dados para o modelo e implementação do processo de carga (ETL) nos bancos de dados do *data warehouse*.

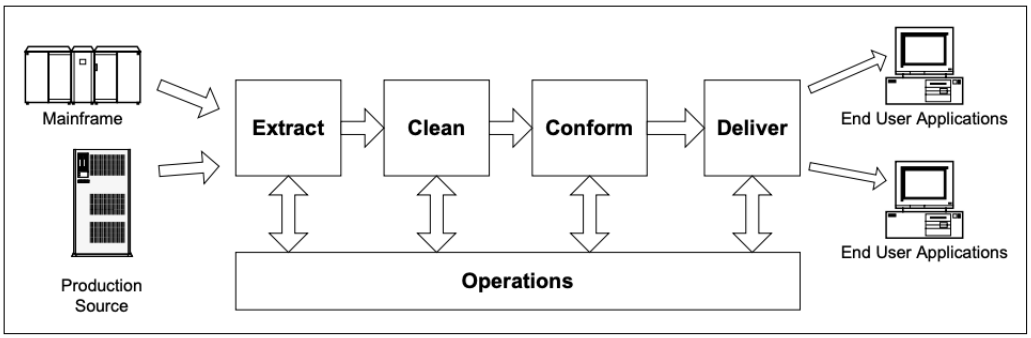

Figura 5.1: *Processo de carga completo [Extraída de KIMBALL e CASERTA, 2004]*

Na Figura 5.1 é possível observar as diferentes etapas de tratamento dos dados ao longo do processo de ETL, onde cada uma das etapas (extract, clean, conform e deliver) é um banco de dados diferente que é responsável por isolar o estado de tratamento do dado a ser carregado no data warehouse.

Neste trabalho, foi criado um protótipo de processo ETL (extract-transform-load) para a manutenção de um *data warehouse* para a integração de dados de SIS-SUS da SMS-SP. Um processo de ETL é responsável por extrair, transformar e carregar dados ao longo das etapas mostradas na Figura 5.1 até carregar um data warehouse.

Além da implementação dos passos do ETL propriamente dita, a criação do protótipo envolveu também a escolha de uma ferramenta de suporte ao processo. A seção 5.1 caracteriza as ferramentas que foram analisadas neste trabalho e justifica a escolha adotada no trabalho.

O protótipo foi testado usando um grande volume de dados reais (anonimizados) extraídos do SINASC, SIM e SIH, fornecidos pela SMS-SP ao projeto InterSCity.

5.1 Ferramentas para ETL

Existem inúmeras ferramentas que tem como principal objetivo auxiliar a construção de um processo de carga (ETL). As principais ferramentas se dividem em duas categorias: as proprietárias, como por exemplo a Kettle (Pentaho), e as de código aberto (software livre), como Airflow e Luigi.

5.1.1 Kettle

Kettle é uma ferramenta de ETL construído em Java, com opções em software livre com recursos limitados e versão empresarial completa. A ferramenta busca a facilidade de uso. Cada processo de ETL é criado com uma ferramenta gráfica onde é possível especificar todo o processo sem escrever uma linha de código. Por causa dessa característica, Kettle é chamado de orientado a metadados. Kettle ainda pode ser utilizado sozinho ou como parte da ferramenta Pentaho Suite, que aumenta o número de funcionalidades disponíveis para o usuário.[Pentaho, 2019]

Os principais benefícios do Kettle:

- Análise de dados de logs do processo ETL
- Criar sua própria *dashboard* para ETL
- Suporte dedicado
- Suporte pré-definido de conectores de diversas fontes

Os pontos negativos do Kettle:

- Maioria dos recursos estão disponíveis apenas na versão empresarial
- Suporte é somente na versão empresarial
- Não é muito utilizado pela comunidade *open source*
- Documentação de difícil acesso

5.1.2 Luigi

Luigi é uma ferramenta de código aberto para criar e gerenciar fluxos de dados, criado e mantido pelo Spotify inspirada no GNU Make.
Escrito em Python, com o objetivo de auxiliar a construção de fluxos de dados complexos. Suporta dependências entre tarefas, contém um agendador central com uma biblioteca extensiva para construir fluxo de dados para diversos banco de dados.[Spotify, 2019]

Os principais benefícios do Luigi são:

- Definição de fluxos de dados utilizando scripts em Python
- Código modular, tornando mais fácil de manter e atualizar fluxos
- Integração com linha de comando

Os pontos negativos do Luigi são:

- Não tem suporte nativo para processos distribuídos
- Interface gráfica não é intuitiva, tornando mais difícil de navegar.
- Escalabilidade é muito limitada

5.1.3 Airflow

Airflow é uma ferramenta agendador de fluxo de dados que permite definir tarefas e dependências em código Python, executar e agendar essas tarefas e distribuir em vários nós de execução. Suporta também agendamento por calendário, podendo agendar tarefas para executar a cada hora, a cada dia e etc. Tem um excelente *dashboard* web, que facilita a visualização das tarefas, do histórico de execução e das configurações de conexão.
No Airflow, o fluxo de dados é definido como uma coleção de tarefas com dependência direcional, basicamente um grafo acíclico direcionado (DAG), no qual cada nó é uma tarefa e as arestas definem dependências entre as tarefas.[Apache, 2019]

Os principais benefícios do Airflow são:

- Tarefas são definidas em Python
- Utilização de DAGs como padrão de trabalho
- Possui agendamento de tarefas mais completo
- Interface web muito completa
- Altamente escalável
- Ferramenta muito utilizada pela comunidade *open source*

Os pontos negativos do Airflow são:

- Airflow não tem suporte para fluxo de dados contínuo (data streaming)

5.1.4 Ferramenta escolhida

Analisando essas possibilidades pode-se observar na Figura 5.2 que o Airflow é a ferramenta que mais chama atenção, principalmente pela flexibilidade e suporte da comunidade. Além de ser atualizado frequentemente e mantido por uma grande organização.
A utilização de Python para desenvolver ETL em código é extremamente útil, trazendo versatilidade, versionamento e clareza no desenvolvimento de todo o *pipeline* do ETL.

Ferramenta	Tipo licensa	Empresa/Organização	Escrito em	Vantagens	Desvantagens
Kettle	código proprietário	Pentaho	Java	Criar sua própria dashboard para ETL	Maioria dos recursos são da versão empresarial
				Análise de dados de logs do processo ETL	Suporte é somente na versão empresarial
				Suporte dedicado	Não é muito utilizado pela comunidade open source
				Suporte pré-de nido de conectores de diversas fontes	Documentação de difícil acesso
Luigi	código aberto	Spotify	Python	Definição de fluxos de dados utilizando Python scripts	Não tem suporte nativo para processos distribuídos
				Código modular, tornando mais fácil de manter e atualizar fluxos	Interface gráfica não é intuitiva, tornando mais difícil de navegar.
				Integração com linha de comando	Escalabilidade é muito limitada
Airflow	código aberto	Airbnb	Python	Definição de fluxos de dados utilizando Python scripts	Airflow não tem suporte para data streaming
				Utilização de grafos acíclicos (DAG) como padrão de	
				Possui agendamento de tarefas mais completo	
				Interface web muito completa	
				Altamente escalável	
				Ferramenta é muito utilizado pela comunidade open source	

Figura 5.2: *Tabela comparativa de ferramentas de ETL*

5.2 Arquitetura do processo ETL desenvolvido

Para isolar os diferentes estágios de tratamento dos dados, foram criados quatro bancos de dados com diferentes esquemas:

- Banco de dados de extração – nesse banco o esquema de dados é o mais simples possível, justamente para armazenar os dados proveniente das fontes;
- Banco de dados de limpeza – nesse banco são armazenados os dados que passaram por um processo de limpeza dos dados, que geralmente envolve remover espaços, caracteres especiais, padronizar campos e outros;
- Banco de dados de conformidade – aqui busca-se armazenar dados não duplicados ou agregados de diferentes fontes que tem significado antes da etapa final;
- Banco de dados de destino (final) – o esquema de dados desse banco é o multidimensional descrito no capítulo anterior, onde os dados estão prontos para consulta pelo usuário final.

O processo de ETL construído tem um total de 10 tarefas. que podem ser vista na Figura 5.3, que extraem, limpam e entregam os dados. Essas tarefas são definidas em um arquivo de leitura Python que é carregado automaticamente ao ligar o Airflow. Elas são executadas para realizarem suas funções nos bancos de dados mencionados na seção 5.2.

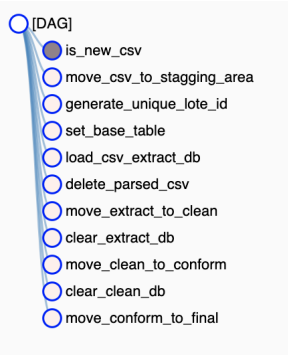

Figura 5.3: *Página com o grafo do processo. Extraída de [Apache, 2019]*

As tarefas definidas são:

```
is_new_csv, move_csv_to_stagging_area, generate_unique_lote_id,
load_csv_extract_db, delete_parsed_csv, move_extract_to_clean,
clear_extract_db, move_clean_to_conform, clear_clean_db, move_conform_to_final.
```

Verificar se existe um novo csv

Essa tarefa é simplesmente uma implementação de um sensor do Airflow para verificar a cada 30 segundos se algum arquivo CSV novo chegou para ser processado. Foi escolhido uma pasta local para ser escutada, pois é mais fácil para testar, sendo assim ao chegar um CSV na pasta, essa tarefa é concluída com sucesso e então o Airflow passa para a próxima tarefa da sua agenda.

```
is_new_csv = FileSensor(
    task_id="is_new_csv",
    poke_interval=30,
    filepath=receive_path
)
```

Esse trecho do código em Python invoca uma classe FileSensor do Airflow que, pelo intervalo definido em 'poke–interval' verifica no local definido pela variável "filepath"se há algum arquivo CSV novo.

Mover CSV para uma área temporária

Área temporária ou "stagging"é um local para o qual o arquivo CSV vai ser movido para que ele fique bloqueado de outras tarefas no Airflow processá-lo simultaneamente. Após sua movimentação o Airflow passa para a próxima tarefa da sua agenda.

```
def move_csv_to_stagging_area(**kwargs):
    import shutil
    import time
    import os
    filepath = kwargs['filepath']
    csv_list = glob.glob(filepath + '*.csv')
```

```
    ti = kwargs['ti']
    if csv_list == []:
        return None
    currently = csv_list.pop()
    file_name = currently.split("/").pop()
    destination = "/usr/local/airflow/stagging/" + file_name
    shutil.move(currently, destination)
    file_size = os.path.getsize(destination)
    if os.path.exists("/usr/local/airflow/from/.DS_Store"):
        os.remove("/usr/local/airflow/from/.DS_Store")
    ti.xcom_push(key='currently_processing', value=destination)
    ti.xcom_push(key='source', value=currently)
    ti.xcom_push(key='file_size', value=file_size)
    return destination
```

Esse método simplesmente move um arquivo para uma pasta definida na variável "destination"e atualiza as variáveis globais "currently−processing", "source"e "file−size"do Airflow com o médoto "ti.xcom push"para que essas informações estejam disponíveis em uma eventual consulta de origem dos dados, já que elas armazenam informações do lote de dados de origem, tais como: o fato destino, o tamanho do arquivo que o registro veio e o arquivo de origem.

Gerar um identificador único para o lote

A cada processo de carga é identificado unicamente como lote, e esse id é armazenado nos fato do esquema final. Nesse momento é gerado o id, e assim o Airflow guarda esse valor para ser utilizado em qualquer tarefa subsequente.

```
def generate_unique_lote_id(**kwargs):
    import random
    ti = kwargs['ti']
    lote_id = random.getrandbits(32)
    ti.xcom_push(key='lote_id', value=lote_id)
    return lote_id
```

Carregar o CSV no banco de extração

Essa tarefa carrega o CSV da área temporária no banco de dados de extração.

```
def load_csv_extract_db(**kwargs):
    ti = kwargs['ti']
  file    = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='currently_processing')
   lote_id  = ti.xcom_pull(task_ids='generate_unique_lote_id', key='lote_id')
   source   = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='source')
  file_size = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='file_size')
    table     = ti.xcom_pull(task_ids='set_base_table', key='table')
```

```
dest_conn = PostgresHook(postgres_conn_id='etl_stage_extract').get_conn()
dest_cursor = dest_conn.cursor()

header = tuple(pd.read_csv(file, sep=';', nrows=1).columns)
header = header + ('loteid',)
strfy_header = '(' + ','.join(str(s) for s in header) + ')'
count = 0
set_lote_id_begin(dest_cursor, lote_id, source, file_size)
for df_row in pd.read_csv(file, sep=';', chunksize=1):
    row = [tuple(x) for x in df_row.values]
    row = row[0]
    row = row + (lote_id, )
    strfy_row = '(' + ','.join('\'' + str(s) + '\'' for s in row) + ')'

query = """INSERT INTO {} {} VALUES {};""".format(table, strfy_header, strfy_row)
    dest_cursor.execute(query)
    count += 1
    if count >= 100:
        break
set_lote_id_end(dest_cursor, lote_id, count, source, file_size)
dest_conn.commit()
dest_cursor.close()
dest_conn.close()
return True
```

A linha lida no CSV com auxílio da biblioteca pandas é armazenada na variavel "row"e inserida no banco com o cursor dest (cursor de destino) do Airflow.

Remover o CSV da área temporária

Essa tarefa remove o CSV da área temporária.

```
def delete_parsed_csv(**kwargs):
    import os
    ti = kwargs['ti']
filepath = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='currently_processing')
    os.remove(filepath)
    filepath = None
    ti.xcom_push(key='currently_processing', value=filepath)
    return True
```

Mover e tratar dados do banco de extração para o banco de limpeza

Essa tarefa realiza o tratamento dos dados a medida que eles são lidos do banco de dados de extração. Após o tratamento o dado é carregado no novo banco que tem somente

dados limpos.

```
def move_extract_to_clean(**kwargs):
    ti = kwargs['ti']
  file    = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='currently_processing')
   lote_id  = ti.xcom_pull(task_ids='generate_unique_lote_id', key='lote_id')
   source   = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='source')
  file_size = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='file_size')
    table     = ti.xcom_pull(task_ids='set_base_table', key='table')
   query = """SELECT * FROM {} WHERE loteId = '{}';""".format(table, lote_id)

    src_conn = PostgresHook(postgres_conn_id='etl_stage_extract').get_conn()
    dest_conn = PostgresHook(postgres_conn_id='etl_stage_clean').get_conn()

    src_cursor = src_conn.cursor()
    dest_cursor = dest_conn.cursor()

    src_cursor.execute(query)
    count = 0
    set_lote_id_begin(dest_cursor, lote_id, source, file_size)
    while True:
        records = src_cursor.fetchmany(size=1)
        if not records:
            break
        clean_records = cleaning(records)


        query_insert = """INSERT INTO {} VALUES %s""".format(table)
        execute_values(dest_cursor, query_insert, clean_records)
        dest_conn.commit()
        count += 1
    set_lote_id_end(dest_cursor, lote_id, count, source, file_size)
    dest_conn.commit()
    src_cursor.close()
    dest_cursor.close()
    src_conn.close()
    dest_conn.close()
    return True

def cleaning(records):
    t_record = list(records[0])
    for id, item in enumerate(t_record):
        if item == 'nan':
            t_record[id] = 'NULL'
        else:
            t_record[id] = " ".join(item.split())
```

```
    records = []
    records.append(tuple(t_record))
    return records
```

Nesse trecho de código tem dois cursores, o src cursor que faz leitura de um registro do banco de origem e o dest cursor que escreve o registro no banco de destino.

Entre esses cursores o método cleaning é chamado, esse método faz uma limpeza simples que remove espaços em brancos, ou registros que tem 'nan' ou 'NULL'. Esse método pode ser estendido com novas regras dependendo do registro, tornando-o bem flexível.

Os métodos auxiliares

`set_lote_id_begin e set_lote_id_end`

servem para registrar os horários respectivamente de início e fim de carga, além de registrar o lote id, tamanho do arquivo sendo processado e etapa do data warehouse atual.

Limpar banco de extração

Como os dados já estão tratados e no banco de limpeza, essa tarefa visa limpar o banco de extração para os próximos dados, assim evitando conflito e isolando o processo.

```
    def clear_extract_db(**kwargs):
        ti = kwargs['ti']
    file    = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='currently_processing')
    lote_id  = ti.xcom_pull(task_ids='generate_unique_lote_id', key='lote_id')
    source   = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='source')
    file_size = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='file_size')
        table     = ti.xcom_pull(task_ids='set_base_table', key='table')
        query = """DELETE FROM {};""".format(table)
    dest_conn = PostgresHook(postgres_conn_id='etl_stage_extract').get_conn()
        dest_cursor = dest_conn.cursor()
        dest_cursor.execute(query)
        dest_conn.commit()
        dest_conn.close()
        return True
```

Esse método simplesmente deleta todos os registros do banco de extração.

Mover e agregar dados do banco de limpeza para o banco de conformidade

Nesta tarefa os dados do banco de limpeza são movidos para o banco de conformidade.

```
    def move_clean_to_conform(**kwargs):
        ti = kwargs['ti']
      file    = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='currently_processing')
```

```
    lote_id  = ti.xcom_pull(task_ids='generate_unique_lote_id', key='lote_id')
    source   = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='source')
    file_size = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='file_size')
    table     = ti.xcom_pull(task_ids='set_base_table', key='table')
    query = """SELECT * FROM {} WHERE loteId = '{}';""".format(table, lote_id)

    src_conn = PostgresHook(postgres_conn_id='etl_stage_clean').get_conn()
    dest_conn = PostgresHook(postgres_conn_id='etl_stage_conform').get_conn()

    src_cursor = src_conn.cursor()
    dest_cursor = dest_conn.cursor()

    src_cursor.execute(query)
    count = 0
    set_lote_id_begin(dest_cursor, lote_id, source, file_size)
    while True:
        records = src_cursor.fetchmany(size=1)
        if not records:
            break
        query_insert = """INSERT INTO {} VALUES %s""".format(table)
        execute_values(dest_cursor, query_insert, records)
        dest_conn.commit()
        count += 1
    set_lote_id_end(dest_cursor, lote_id, count, source, file_size)
    dest_conn.commit()
    src_cursor.close()
    dest_cursor.close()
    src_conn.close()
    dest_conn.close()
    return True
```

Neste método utiliza-se de dois cursores o src que lê do banco de dados de origem e o cursor dest que grava o registro no banco de dados de destino.

Os métodos auxiliares

`set_lote_id_begin e set_lote_id_end`

servem para registrar os horários respectivamente de início e fim de carga, além de registrar o lote id, tamanho do arquivo sendo processado e etapa do data warehouse atual.

Limpar banco de limpeza

Como os dados já estão tratados e no banco de conformidade, essa tarefa visa limpar o banco de conformidade para os próximos dados, assim evitando conflito e isolando o processo.

```
def clear_clean_db(**kwargs):
```

```
    ti = kwargs['ti']
file    = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='currently_processing')
 lote_id  = ti.xcom_pull(task_ids='generate_unique_lote_id', key='lote_id')
source   = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='source')
file_size = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='file_size')
    table     = ti.xcom_pull(task_ids='set_base_table', key='table')
    query = """DELETE FROM {};""".format(table)
    dest_conn = PostgresHook(postgres_conn_id='etl_stage_clean').get_conn()
    dest_cursor = dest_conn.cursor()
    dest_cursor.execute(query)
    dest_conn.commit()
    dest_conn.close()
    return True
```

Esse método simplesmente deleta todos os registros do banco de limpeza.

Mover dados do banco de conformidade para o banco final

Nesta tarefa os dados do banco de conformidade são movidos para o banco final de acordo com o esquema multidimensional.

```
    def move_conform_to_final(**kwargs):
        ti = kwargs['ti']
    file    = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='currently_processing')
    lote_id  = ti.xcom_pull(task_ids='generate_unique_lote_id', key='lote_id')
    source   = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='source')
    file_size = ti.xcom_pull(task_ids='move_csv_to_stagging_area', key='file_size')
        table     = ti.xcom_pull(task_ids='set_base_table', key='table')
     query = """SELECT * FROM {} WHERE loteId = '{}';""".format(table, lote_id)

     src_conn = PostgresHook(postgres_conn_id='etl_stage_conform').get_conn()
      dest_conn = PostgresHook(postgres_conn_id='etl_stage_final').get_conn()

        src_cursor = src_conn.cursor()
        dest_cursor = dest_conn.cursor()

    query_header = """ select column_name from information_schema.columns where table_name = '{}'
        src_cursor.execute(query_header)
        headers_result = src_cursor.fetchmany(size=1000)
        headers = [str(s[0]).lower() for s in headers_result]

        src_cursor.execute(query)
        count = 0
        set_lote_id_begin(dest_cursor, lote_id, source, file_size)
        while True:
            records = src_cursor.fetchmany(size=1)
            if not records:
```

```
                break
            insert_final_table(dest_cursor, table, records, headers)
            count += 1
        set_lote_id_end(dest_cursor, lote_id, count, source, file_size)
        dest_conn.commit()
        src_cursor.close()
        dest_cursor.close()
        src_conn.close()
        dest_conn.close()
        return True
```

Esse método faz a mesma coisa que os anteriores, porém o método intermediário chamado entre os cursores de leitura do banco de dados fonte e do banco de dados de destino é o insert – final – table, que realiza diversas ações para concluir a modelagem dos dados no esquema multidimensional.

Um pequeno exemplo disso, ocorre no trecho abaixo que recebe 1 registro proveniente do sistema SINASC e realiza a população no esquema multidimensional chamando outros métodos.

```
def insert_final_table(dest_cursor, table, records, headers):
    if table == 'sinasc':
        id_mae = add_pessoa_mae(dest_cursor, table, records[0], headers)
        id_bebe = add_pessoa_bebe(dest_cursor, table, records[0], headers)
    id_parto = add_parto(dest_cursor, table, records[0], headers, id_mae, id_bebe)
    id_gestacao = add_gestacao(dest_cursor, table, records[0], headers, id_mae, id_parto)
```

O método de adicionar pessoa com os dados do SINASC que caracteriza uma mãe é realizado da seguinte maneira: os dados são extraídos das colunas originais do registro do sinasc, tratados, se necessário adicionados em outras tabelas do modelo multidimensional e depois adicionado na tabela Pessoa do modelo dimensional.

```
    def add_pessoa_mae(dest_cursor, table, record, headers):
        idPessoa = get_next_id_from_table('Pessoa', dest_cursor)
        nome = get_values_from_headers(headers, record, ['NO_MAE_HASH'])[0]
        sexo = 'F'
    dataNasc = add_data(get_values_from_headers(headers, record, ['DT_NASCIMENTO_MAE'])[0]
        cns = get_values_from_headers(headers, record, ['CO_SEQ_DN'])[0]
        locResidencia = add_location('', dest_cursor)
        locNascimento = add_location('', dest_cursor)
        loteid = get_values_from_headers(headers, record, ['loteid'])[0]

    value = [(idPessoa, nome, sexo, dataNasc, cns, locResidencia, locNascimento, loteid)]
    query_insert = """INSERT INTO Pessoa (idPessoa, nome, sexo, dataNasc, cns, locResidencia
        execute_values(dest_cursor, query_insert, value)
```

Realizando esse processo para cada um dos dados no esquema multidimensional e organizando-os corretamente, os dados ficam disponíveis no data warehouse para consulta, junto com as informações do lote que vieram e de quando passaram por cada etapa do ETL até chegar no data warehouse.

5.3 Reprodução

Para reproduzir a prova de conceito do data warehouse com o processo de carga, é necessário um computador rodando sistema operacional Unix (Linux ou MacOS) com docker [Docker, 2019a] e docker-compose [Docker, 2019b] instalados.
Após isso, clonar o repositório do processo de ETL:

```
git clone https://github.com/mvsousa/mac499.git
```

Entrar na pasta do repositório e rode o comando:

```
docker-compose up -d
```

Após a finalização das construções de imagens docker, basta acessar no navegador a url http://localhost:8080 para acessar a dashboard do Airflow.
Na página inicial do Airflow, basta ligar uma "chave"para ETL.PROCESS e então o processo de ETL desenvolvido estará rodando.
Devido a restrições, os dados utilizados para teste não podem ser disponibilizados.

Capítulo 6

Conclusão

Devido à grande quantidade de sistemas e dados, os atuais Sistemas de Informações em Saúde (SIS) do SUS enfrentam uma série de problemas, tais como: dados fragmentados, dados duplicados e dados descentralizados. A falta de integração entre os SIS do SUS torna a análise dos dados limitada e difícil, sendo assim a construção de um *data warehouse* é uma solução que se destaca por ter um bom custo-benefício.

Para manter o *data warehouse* atualizado, é necessário um processo automatizado que receba (ou extraia) os novos dados das fontes, trate-os e carregue-os no banco de dados unificado, mantendo a conformidade dos dados, sendo este um processo de ETL (extract-transform-load) ou processo de carga.

6.1 Resultados

Neste trabalho foi construído um esquema e um banco de dados multidimensional rodando em um servidor PostgreSQL.

Foi criado também um processo de carga (ETL) utilizando a ferramenta Airflow em conjunto com Python, esse processo age de forma automática, detectando a chegada de novos dados das fontes do SIS-SUS e assim realizando a atualização do data warehouse.

Com um processo de carga bem robusto, foi possível realizar testes com quase 2.5 milhões de registros provenientes dos sistemas SIM, SIH e SINASC em arquivos CSV. O principal objetivo a foi atingido, que era carregar o data warehouse com os dados de saúde pública e mantê-los nesse repositório unificado de dados indexados de maneira temporal.

Como prova de conceito, foi criado um pequeno teste para reprodução em conteineres docker, que pode ser executada em qualquer máquina unix com o docker e o docker-compose instalados.

6.2 Próximos passos

Implantar e configurar o *data warehouse* e o processo de carga na SMS-SP para as fontes de dados disponíveis.

Ampliar a cobertura do data warehouse nos dados de saúde pública, os próximos passos são mapear no processo de carga outras relações de outros sistemas do SUS.

Adaptar e generalizar o *data warehouse* e o processo de carga para que possa ser utilizado por outras prefeituras.

Adaptar o processo de carga para receber dados de novas fontes.

Apêndice A

Comandos SQL data warehouse

Abaixo seguem os comandos que criam as tabelas do esquema de banco de dados para o data warehouse.

O comando em SQL que cria a tabela fato `Atendimento` é:

```
CREATE TABLE IF NOT EXISTS Atendimento (
  idAtendimento INT NOT NULL,
  cidCAs VARCHAR(45) NULL,
  idEstabelecimento INT NULL,
  dt INT NULL,
  cidPri INT NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idAtendimento),
  CONSTRAINT fk_Atendimento_Estabelecimento1
    FOREIGN KEY (idEstabelecimento)
    REFERENCES Estabelecimento (idEstabelecemento)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Atendimento_Data1
    FOREIGN KEY (dt)
    REFERENCES Data (idData)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Atendimento_CID1
    FOREIGN KEY (cidPri)
    REFERENCES CID (idCid)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
CREATE INDEX fk_Atendimento_Estabelecimento1_idx ON Atendimento (idEstabelecimento ASC);
CREATE INDEX fk_Atendimento_Data1_idx ON Atendimento (dt ASC);
```

```
CREATE INDEX fk_Atendimento_CID1_idx ON Atendimento (cidPri ASC);
```

O comando em SQL que gera a tabela dimensão Estabelecimento é:

```
CREATE TABLE IF NOT EXISTS Estabelecimento (
  idEstabelecemento INT NOT NULL,
  cnes INT NULL,
  locEstab INT NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idEstabelecemento),
  CONSTRAINT fk_Estabelecimento_Localizacao1
    FOREIGN KEY (locEstab)
    REFERENCES Localizacao (idLocalizacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
CREATE INDEX fk_Estabelecimento_Localizacao1_idx ON Estabelecimento (locEstab ASC);
```

O comando em SQL que gera a tabela dimensão CID é:

```
CREATE TABLE IF NOT EXISTS CID (
  idCid INT NOT NULL,
  idSubCategoria INT NULL,
  descricao VARCHAR(200) NULL,
  cid VARCHAR(45) NULL,
  PRIMARY KEY (idCid),
  CONSTRAINT fk_CID_CID_Subcategorias1
    FOREIGN KEY (idSubCategoria)
    REFERENCES CID_Subcategorias (idSubcategorias)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
CREATE INDEX fk_CID_CID_Subcategorias1_idx ON CID (idSubCategoria ASC)  ;
```

O comando em SQL que gera a tabela fato Pessoa é:

```
CREATE TABLE IF NOT EXISTS Pessoa (
  idPessoa INT NOT NULL,
  nome VARCHAR(200) NULL,
  sexo VARCHAR(1) NULL,
  dataNasc INT NULL,
  cns VARCHAR(200) NULL,
  locResidencia INT NULL,
```

```
  locNascimento INT NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idPessoa),
  CONSTRAINT fk_Pessoa_Data1
    FOREIGN KEY (dataNasc)
    REFERENCES Data (idData)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Pessoa_Localizacao1
    FOREIGN KEY (locResidencia)
    REFERENCES Localizacao (idLocalizacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Pessoa_Localizacao2
    FOREIGN KEY (locNascimento)
    REFERENCES Localizacao (idLocalizacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
CREATE INDEX fk_Pessoa_Data1_idx ON Pessoa (dataNasc ASC);
```

O comando em SQL que gera a tabela fato `Parto` é:

```
CREATE TABLE IF NOT EXISTS Parto (
  idParto INT NOT NULL,
  idMae INT NULL,
  idPai INT NULL,
  idPessoa INT NULL,
  horaNasc VARCHAR(45) NULL,
  escolaridadeMae INT NULL,
  ocupacaoMae INT NULL,
  tpParto INT NULL,
  tpApresent INT NULL,
  trabPartIndu INT NULL,
  cesParto INT NULL,
  tpNascAssi INT NULL,
  tpRobson VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  internacao INT NULL,
  PRIMARY KEY (idParto),
  CONSTRAINT fk_Nascimento_Pessoa1
    FOREIGN KEY (idMae)
    REFERENCES Pessoa (idPessoa)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Nascimento_Pessoa2
    FOREIGN KEY (idPai)
    REFERENCES Pessoa (idPessoa)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Nascimento_Pessoa3
    FOREIGN KEY (idPessoa)
    REFERENCES Pessoa (idPessoa)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Nascimento_Escolaridade1
    FOREIGN KEY (escolaridadeMae)
    REFERENCES Escolaridade (idEscolaridade)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Parto_Ocupacao1
    FOREIGN KEY (ocupacaoMae)
    REFERENCES Ocupacao (idOcupacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Parto_Tipo_Parto1
    FOREIGN KEY (tpParto)
    REFERENCES Tipo_Parto (idTipoParto)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Parto_Tipo_Apresentacao_RN1
    FOREIGN KEY (tpApresent)
    REFERENCES Tipo_Apresentacao_RN (idTipoApresentacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Parto_Trabalho_Parto_Induzido1
    FOREIGN KEY (trabPartIndu)
    REFERENCES Trabalho_Parto_Induzido (idTrabalho)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Parto_Cesaria_Antes_Trabalho_Parto1
    FOREIGN KEY (cesParto)
    REFERENCES Cesaria_Antes_Trabalho_Parto (idCesariaParto)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Parto_Nascimento_Assistido1
    FOREIGN KEY (tpNascAssi)
    REFERENCES Nascimento_Assistido (idNascAssis)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Parto_Internacao1
    FOREIGN KEY (internacao)
    REFERENCES Internacao (idInternacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
  );
CREATE INDEX fk_Nascimento_Pessoa1_idx ON Parto (idMae ASC);
CREATE INDEX fk_Nascimento_Pessoa2_idx ON Parto (idPai ASC);
CREATE INDEX fk_Nascimento_Pessoa3_idx ON Parto (idPessoa ASC);
CREATE INDEX fk_Nascimento_Escolaridade1_idx ON Parto (escolaridadeMae ASC);
CREATE INDEX fk_Parto_Ocupacao1_idx ON Parto (ocupacaoMae ASC);
CREATE INDEX fk_Parto_Tipo_Parto1_idx ON Parto (tpParto ASC);
CREATE INDEX fk_Parto_Tipo_Apresentacao_RN1_idx ON Parto (tpApresent ASC);
CREATE INDEX fk_Parto_Trabalho_Parto_Induzido1_idx ON Parto (trabPartIndu ASC);
CREATE INDEX fk_Parto_Cesaria_Antes_Trabalho_Parto1_idx ON Parto (cesParto ASC);
CREATE INDEX fk_Parto_Nascimento_Assistido1_idx ON Parto (tpNascAssi ASC);
CREATE INDEX fk_Parto_Internacao1_idx ON Parto (internacao ASC);
```

O comando em SQL que gera a tabela dimensão Ocupação é:

```
CREATE TABLE IF NOT EXISTS Ocupacao (
  idOcupacao INT NOT NULL,
  ocupacao VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idOcupacao)
);
```

O comando em SQL que gera a tabela dimensão Trabalho Parto Induzido é:

```
CREATE TABLE IF NOT EXISTS Trabalho_Parto_Induzido (
  idTrabalho INT NOT NULL,
  partoInduzido VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
```

```
  loteId VARCHAR(40),
  PRIMARY KEY (idTrabalho)
);
```

O comando em SQL que gera a tabela dimensão `Cesária Antes Trabalho Parto` é:

```
CREATE TABLE IF NOT EXISTS Cesaria_Antes_Trabalho_Parto (
  idCesariaParto INT NOT NULL,
  cesariaParto VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idCesariaParto)
);
```

O comando em SQL que gera a tabela dimensão `Nascimento Assistido` é:

```
CREATE TABLE IF NOT EXISTS Nascimento_Assistido (
  idNascAssis INT NOT NULL,
  nascAssis VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idNascAssis)
);
```

O comando em SQL que gera a tabela dimensão `Escolaridade` é:

```
CREATE TABLE IF NOT EXISTS Escolaridade (
  idEscolaridade INT NOT NULL,
  escolaridade VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idEscolaridade)
);
```

O comando em SQL que gera a tabela dimensão `Tipo Parto` é:

```
CREATE TABLE IF NOT EXISTS Tipo_Parto (
  idTipoParto INT NOT NULL,
  tipoParto VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idTipoParto)
);
```

O comando em SQL que gera a tabela dimensão `Tipo Apresentacao RN` é:

```
CREATE TABLE IF NOT EXISTS Tipo_Apresentacao_RN (
  idTipoApresentacao INT NOT NULL,
  tipoApresentacao VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
```

```
  PRIMARY KEY (idTipoApresentacao)
);
```

O comando em SQL que gera a tabela fato Gestação é:

```
CREATE TABLE IF NOT EXISTS Gestacao (
idGestacao INT NOT NULL,
qtdFilhoMort INT NULL,
semGestac INT NULL,
dtUltiMenst INT NULL,
mesPrenatal VARCHAR(45) NULL,
qtdConsulta INT NULL,
altoRisco VARCHAR(45) NULL,
tpGravidez VARCHAR(45) NULL,
dtRegistro TIMESTAMP(0) NULL,
loteId VARCHAR(40),
idMae INT NULL,
idAtendimento INT NULL,
idParto INT NULL,
PRIMARY KEY (idGestacao),
CONSTRAINT fk_Gestacao_Pessoa1
  FOREIGN KEY (idMae)
  REFERENCES Pessoa (idPessoa)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT fk_Gestacao_Atendimento1
  FOREIGN KEY (idAtendimento)
  REFERENCES Atendimento (idAtendimento)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT fk_Gestacao_Parto1
  FOREIGN KEY (idParto)
  REFERENCES Parto (idParto)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
);
CREATE INDEX fk_Gestacao_Pessoa1_idx ON Gestacao (idMae ASC);
CREATE INDEX fk_Gestacao_Atendimento1_idx ON Gestacao (idAtendimento ASC);
CREATE INDEX fk_Gestacao_Parto1_idx ON Gestacao (idParto ASC);
```

O comando em SQL que gera a tabela fato Óbito é:

```
CREATE TABLE IF NOT EXISTS Obito (
idObito INT NOT NULL,
numeroDo INT NULL,
causaBas VARCHAR(45) NULL,
causaMorte VARCHAR(45) NULL,
idPessoa INT NOT NULL,
```

```
dtObito INT NOT NULL,
horaObito VARCHAR(45) NULL,
tpObito INT NOT NULL,
obitoGrav INT NOT NULL,
obitoPuerp INT NOT NULL,
locOcor INT NOT NULL,
obitoDurante INT NOT NULL,
dtRegistro TIMESTAMP(0) NULL,
loteId VARCHAR(40),
PRIMARY KEY (idObito),
CONSTRAINT fk_Obito_Pessoa1
  FOREIGN KEY (idPessoa)
  REFERENCES Pessoa (idPessoa)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT fk_Obito_Data1
  FOREIGN KEY (dtObito)
  REFERENCES Data (idData)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT fk_Obito_Tipo_Obito1
  FOREIGN KEY (tpObito)
  REFERENCES Tipo_Obito (idTpObito)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT fk_Obito_Obito_na_Gravidez1
  FOREIGN KEY (obitoGrav)
  REFERENCES Obito_na_Gravidez (idObitoGravidez)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT fk_Obito_Obito_no_Puerperio1
  FOREIGN KEY (obitoPuerp)
  REFERENCES Obito_no_Puerperio (idObitoPuerperio)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT fk_Obito_Localizacao1
  FOREIGN KEY (locOcor)
  REFERENCES Localizacao (idLocalizacao)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT fk_Obito_Atendimento1
  FOREIGN KEY (obitoDurante)
  REFERENCES Atendimento (idAtendimento)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
);
```

```
CREATE INDEX fk_Obito_Pessoa1_idx ON Obito (idPessoa ASC);
CREATE INDEX fk_Obito_Data1_idx ON Obito (dtObito ASC);
CREATE INDEX fk_Obito_Tipo_Obito1_idx ON Obito (tpObito ASC);
CREATE INDEX fk_Obito_Obito_na_Gravidez1_idx ON Obito (obitoGrav ASC);
CREATE INDEX fk_Obito_Obito_no_Puerperio1_idx ON Obito (obitoPuerp ASC);
CREATE INDEX fk_Obito_Localizacao1_idx ON Obito (locOcor ASC);
CREATE INDEX fk_Obito_Atendimento1_idx ON Obito (obitoDurante ASC);
```

O comando em SQL que gera a tabela dimensão `Tipo Óbito` é:

```
CREATE TABLE IF NOT EXISTS Tipo_Obito (
  idTpObito INT NOT NULL,
  tipoObito VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idTpObito)
);
```

O comando em SQL que gera a tabela dimensão `Óbito na Gravidez` é:

```
CREATE TABLE IF NOT EXISTS Obito_na_Gravidez (
  idObitoGravidez INT NOT NULL,
  obitoGravidez VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idObitoGravidez)
);
```

O comando em SQL que gera a tabela dimensão `Óbito no Puerpério` é:

```
CREATE TABLE IF NOT EXISTS Obito_no_Puerperio (
  idObitoPuerperio INT NOT NULL,
  obitoPuerperio VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idObitoPuerperio)
);
```

O comando em SQL que gera a tabela fato `Internação` é:

```
CREATE TABLE IF NOT EXISTS Internacao (
  idInternacao INT NOT NULL,
  numAih INT NULL,
  numAihAnt INT NULL,
  numAihProx INT NULL,
  diarias INT NULL,
  motSaida VARCHAR(45) NULL,
  procedimentoQtd INT NULL,
  diariasUti INT NULL,
  diariasUi INT NULL,
```

```
  utineoMesesGestacao INT NULL,
  utineoMotSaida VARCHAR(45) NULL,
  idAtendimento INT NOT NULL,
  especialidade INT NOT NULL,
  carInternacao INT NOT NULL,
  dtInternacao INT NOT NULL,
  dtSaida INT NOT NULL,
  procRealizado INT NOT NULL,
  procedimento INT NOT NULL,
  procSolicitado INT NOT NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idInternacao),
  CONSTRAINT fk_Internacao_Atendimento1
    FOREIGN KEY (idAtendimento)
    REFERENCES Atendimento (idAtendimento)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Internacao_Especialidade1
    FOREIGN KEY (especialidade)
    REFERENCES Especialidade (idEspecialidade)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Internacao_Carater_Internacao1
    FOREIGN KEY (carInternacao)
    REFERENCES Carater_Internacao (idCarInternacao)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Internacao_Data1
    FOREIGN KEY (dtInternacao)
    REFERENCES Data (idData)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Internacao_Data2
    FOREIGN KEY (dtSaida)
    REFERENCES Data (idData)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Internacao_Procedimento1
    FOREIGN KEY (procRealizado)
    REFERENCES Procedimento (idProcedimento)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Internacao_Procedimento2
    FOREIGN KEY (procedimento)
    REFERENCES Procedimento (idProcedimento)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_Internacao_Procedimento3
    FOREIGN KEY (procSolicitado)
    REFERENCES Procedimento (idProcedimento)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
CREATE INDEX fk_Internacao_Atendimento1_idx ON Internacao (idAtendimento ASC);
CREATE INDEX fk_Internacao_Especialidade1_idx ON Internacao (especialidade ASC);
CREATE INDEX fk_Internacao_Carater_Internacao1_idx ON Internacao (carInternacao ASC);
CREATE INDEX fk_Internacao_Data1_idx ON Internacao (dtInternacao ASC);
CREATE INDEX fk_Internacao_Data2_idx ON Internacao (dtSaida ASC);
CREATE INDEX fk_Internacao_Procedimento1_idx ON Internacao (procRealizado ASC);
CREATE INDEX fk_Internacao_Procedimento2_idx ON Internacao (procedimento ASC);
CREATE INDEX fk_Internacao_Procedimento3_idx ON Internacao (procSolicitado ASC);
```

O comando em SQL que gera a tabela dimensão Procedimento é:

```
CREATE TABLE IF NOT EXISTS Procedimento (
  idProcedimento INT NOT NULL,
  procedimento VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idProcedimento)
);
```

O comando em SQL que gera a tabela dimensão Caráter Internação é:

```
CREATE TABLE IF NOT EXISTS Carater_Internacao (
  idCarInternacao INT NOT NULL,
  carInternacao VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idCarInternacao)
);
```

O comando em SQL que gera a tabela dimensão Especialidade é:

```
CREATE TABLE IF NOT EXISTS Especialidade (
  idEspecialidade INT NOT NULL,
  especialidade VARCHAR(45) NULL,
  dtRegistro TIMESTAMP(0) NULL,
  loteId VARCHAR(40),
  PRIMARY KEY (idEspecialidade)
);
```

Referências

[Apache 2019] Apache. *Airflow*. URL: https://airflow.apache.org (acesso em 02/12/2019) (citado nas pgs. 3, 29, 31).

[Docker 2019a] Inc Docker. *Docker*. URL: https://www.docker.com (acesso em 02/12/2019) (citado nas pgs. 3, 39).

[Docker 2019b] Inc Docker. *Docker Compose*. URL: https://docs.docker.com/compose/install (acesso em 02/12/2019) (citado na pg. 39).

[Fonseca 2015] F.C.S.d. Fonseca. "Sistema de informação da atenção à saúde: fragmentação à interoperabilidade. em sistemas de informação da atenção à saúde: contextos históricos, avanços e perspectivas no sus". Em: (2015), pg 9–21 (citado na pg. 1).

[KIMBALL e CASERTA 2004] Ralph. KIMBALL e Joe CASERTA. "The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data". Em: 1 ed (2004), pg 3–25 (citado nas pgs. 5, 7, 8, 10, 27).

[NAVATHE e ELMASRI 2016] Shamkant B. NAVATHE e Ramez ELMASRI. "Fundamentals of database systems". Em: *Overview of Data Warehousing and OLAP* 6 ed (2016), pg 1067–1069 (citado nas pgs. 2, 5–7).

[Pentaho 2019] Inc Pentaho. *Kettle*. URL: https://community.hitachivantara.com/s/article/data-integration-kettle (acesso em 02/12/2019) (citado na pg. 28).

[Saúde 2004] Ministério da Saúde. *Política Nacional de Informação e Informática em Saúde Proposta Versão 2.0 (Inclui deliberações da 12a. Conferência Nacional de Saúde)*. 2004. URL: http://bvsms.saude.gov.br/bvs/publicacoes/PoliticaInformacaoSaude29_03_2004.pdf (acesso em 02/12/2019) (citado nas pgs. 1, 15).

[Spotify 2019] Inc Spotify. *Luigi*. URL: https://github.com/spotify/luigi (acesso em 02/12/2019) (citado na pg. 28).