UNIVERSIDADE DO MINHO

Trabalho Prático: Episódios de Urgência

Mestrado Integrado em Engenharia Informática
Aplicações Informáticas na Biomedicina
(4º ano, 1º semestre, 2020/2021)
Relatório de Desenvolvimento
Grupo 2

Ana Almeida (a83916)

André Figueiredo (a84807)

Bruno Veloso (a78352)

Miguel Solino (a86435)

Rafael Lourenço (a86266)

Braga, 24 de fevereiro de 2021

Resumo

Este relatório foi elaborado no âmbito da Unidade Curricular de Aplicações Informáticas na Biomedicina, para a qual desenvolvemos um sistema de `Data Warehousing` e um sistema de `Business Intelligence` para o suporte à decisão clínica.

Neste relatório são apresentados e detalhados os vários componentes desenvolvidos para estes sistemas, desde o pensamento por detrás da modelação lógica (e consequente modelação dimensional), à forma de povoamento e indicadores gerados, entre outros componentes.

Primeiramente, realizámos uma breve contextualização e declaração dos objetivos deste trabalho, passando, também, por explicar em que é que este consiste.

Seguidamente, expomos um conjunto de decisões que considerámos importantes no processo de criação do modelo lógico. Na nossa proposta o modelo lógico foi construído com o esquema *fact constellation*, isto é, modelo dimensional de factos em constelação, recorrendo ao *software* `MySQL`. O processo anteriormente descrito traduz-se na modelação do sistema de `Data Warehousing`.

Posteriormente, apresentámos a forma como o povoamento do `Data Warehousing` foi efetuado, isto é, recorrendo a *scripts* escritos em `Python` e à criação de *jobs* no `Talend`. Esta última ferramenta combina todas as fases do processo `ETL` (*Extract*, *Transform*, *Load*), ou seja, extração, transformação e carregamento de dados. Ainda nesta fase, apresentámos as estruturas criadas para a atualização dos dados de forma incremental e/ou diferencial, isto é, as rotinas do `MySQL` - *procedures*, *functions* e *triggers*.

De seguida, fundamentámos e analisámos os indicadores (criados com recurso ao *software* `Tableau Desktop`) relativos aos dados em estudo, baseados nas tabelas do modelo lógico, mais precisamente, nos parâmetros e possíveis relações que cada uma possui. O processo anteriormente descrito compõe o sistema de `Business Intelligence`. Desenvolvemos, ainda, um modelo de *interface*, com recurso a *mockups*, para uma possível aplicação web/móvel.

Por fim, efetuámos uma reflexão sobre o desenvolvimento deste projeto.

Área de Aplicação: Desenvolvimento de um sistema para o suporte à decisão clínica, na área da Biomedicina.
Palavras-Chave: `Data Warehousing`, `Business Intelligence`, `ETL`.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Enquadramento e Contexto

A Biomedicina é uma área focada no estudo de doenças com o objetivo de compreender as causas, efeitos, fatores ambientais e epidemiológicos para aprimorar (ou até evitar) diagnósticos e tratamentos. Ora, tratamentos mais eficazes têm como consequência uma saúde melhor cuidada, sendo este um tema de extrema importância, principalmente nos tempos que vivemos atualmente. O bem-estar dos cidadãos e das sociedades está diretamente ligado a esta área, pelo que a prestação de cuidados básicos de saúde é fundamental. Sendo que, no nosso país, a população está cada vez mais envelhecida, o bom funcionamento dos sistemas de serviço de urgência é crucial para uma ação eficaz no tratamento e manutenção da saúde da população. Todavia, a gestão e planeamento dos múltiplos atos realizados neste serviço é complexa, pois é necessário coordenar vários pedidos, como procedimentos e prescrições de medicação, com vista a encontrar um potencial diagnóstico.
Assim e para manter uma boa gestão, a análise de dados dos serviços de urgência de vários hospitais torna-se crucial, de forma a encontrar padrões relevantes e perceber o que se pode ou não melhorar e como alocar os variados recursos disponíveis.

1.2 Problema e Objetivo

Este relatório é, então, o resultado do trabalho prático proposto e elaborado para esta unidade curricular e que consiste na análise de um *dataset* que contém dados reais de vários episódios de urgência de num determinado hospital nacional.
Assim, o problema apresentado consiste em desenvolver um sistema de `Data Warehousing` e de `Business Intelligence` (aplicando, para tal, os vários conhecimentos que esta UC nos forneceu), com o principal objetivo de analisar dados relativos a episódios de urgência, auxiliando no suporte à decisão clínica.
De um modo mais geral e com base nos aspetos referidos, acreditamos que esta análise trará um conceito útil e praticável, facilitando, assim, a gestão efetuada nos serviços de urgência de vários hospitais, bem como do próprio Serviço Nacional de Saúde.

1.3 Estrutura do Relatório

O presente relatório é composto por 7 capítulos.
No primeiro capítulo, *Introdução*, é feita uma descrição da natureza do problema tratado, assim como o contexto em que se encontra inserido e os objetivos pretendidos.

No segundo capítulo, *Contextualização do dataset*, é efetuada a descrição dos vários ficheiros *.csv* que compõem o *dataset* fornecido e como se relaciona a sua informação.
No terceiro capítulo, *Modelação*, é descrita a forma como o modelo lógico (e consequente modelo físico) foi idealizado/construído, através de uma análise detalhada tabela a tabela.
No quarto capítulo, *Povoamento e Gestão do Data Warehouse*, é feita uma descrição das várias ferramentas utilizadas para o povoamento do `Data Warehouse`, analisando as diferenças que existem entre elas e que nos levaram a optar por uma em detrimento da outra em certos ficheiros do *dataset*.
No quinto capítulo, *Business Intelligence - Indicadores Clínicos*, são ilustrados os vários *dashboards* desenvolvidos, sendo fornecida uma breve descrição sobre o porquê da sua criação e uma análise detalhada de cada um dos gráficos que os compõem.
No sexto capítulo, *Possível Aplicação e sua Interface*, é fornecida uma visão, em forma de *mockups*, de uma possível aplicação e respetiva *interface* que possa ser usada, no futuro, num serviço de urgência como o em estudo.
No último capítulo, *Conclusões e Perspetivas Futuras*, é feita uma retrospetiva de todo o trabalho concebido e uma análise futura de possível continuidade deste tipo de análise.

Capítulo 2

Contextualização do *dataset*

Seguindo a linha de raciocínio apresentada na secção 1.1, o *dataset* fornecido permite-nos analisar o serviço de urgência de um dado hospital, durante o ano de 2018.
Assim sendo, os dados encontram-se divididos em cinco ficheiros *.csv*, estando cada um destes relacionado com diferentes categorias:

- `urgency_episodes_new.csv` - Trata-se do ficheiro principal que contém mais informação, como os *ids* dos vários episódios de urgência, sendo que cada um destes é acompanhado pela informação sobre qual o profissional que foi responsável pelo episódio, a data e hora da admissão, triagem, diagnóstico e alta hospitalar. Além disto, contém, ainda, informação sobre o paciente (distrito, género e data de nascimento), bem como dados relevantes para o seu tratamento/cuidados, como a causa da deslocação ao serviço de urgência, cor da pulseira e valor da escala de dor que o paciente sente;

- `urgency_prescriptions.csv` - Trata-se do ficheiro que contém informação relativa às prescrições efetuadas para os vários episódios de urgência (identificados pelo seu *id*) como, por exemplo, o código da prescrição, o profissional que a receitou, a data de prescrição, o medicamento e quantidade receitadas, a posologia, entre outros;

- `urgency_procedures.csv` - Trata-se do ficheiro que contém informação relativa aos procedimentos e que descreve a sua utilidade. Para cada episódio de urgência (identificado pelo seu *id*) é indicado o profissional responsável por prescrever dado procedimento e a data da sua prescrição. Além disto, contém também a data e hora de início e cancelamento (se aplicável) do procedimento, uma nota para contextualizar a decisão, os profissionais responsáveis e a descrição do procedimento.

- `urgency_exams.csv` - Trata-se do ficheiro que contém informação relativa sobre os exames realizados para os vários episódios de urgência, como, por exemplo, o número do exame e a sua descrição;

- `icd9_hierarchy.csv` - Trata-se do único ficheiro que não se relaciona diretamente com os episódios de urgência, pois apenas contém informação sobre a hierarquia `ICD9` (*International Classification of Diseases* versão `9`), isto é, para cada diagnóstico, tem a informação sobre os vários níveis (no máximo cinco) que a caracterizam.

Em suma, é necessário cruzar a informação dos vários ficheiros para conseguir enquadrar os múltiplos acontecimentos que, por sua vez, irão permitir criar um sistema que melhor utiliza a informação disponibilizada.

Capítulo 3

Modelação

3.1 Modelo Lógico

Após uma visualização detalhada de cada *.csv* fornecido e consequente análise abordada na secção anterior, procedemos à elaboração do modelo lógico que melhor se adequa a toda a informação que estes contêm. O modelo criado segue um esquema em constelação (*Fact Constellation Schema*), com 21 tabelas, sendo 2 de factos e 19 de dimensão.

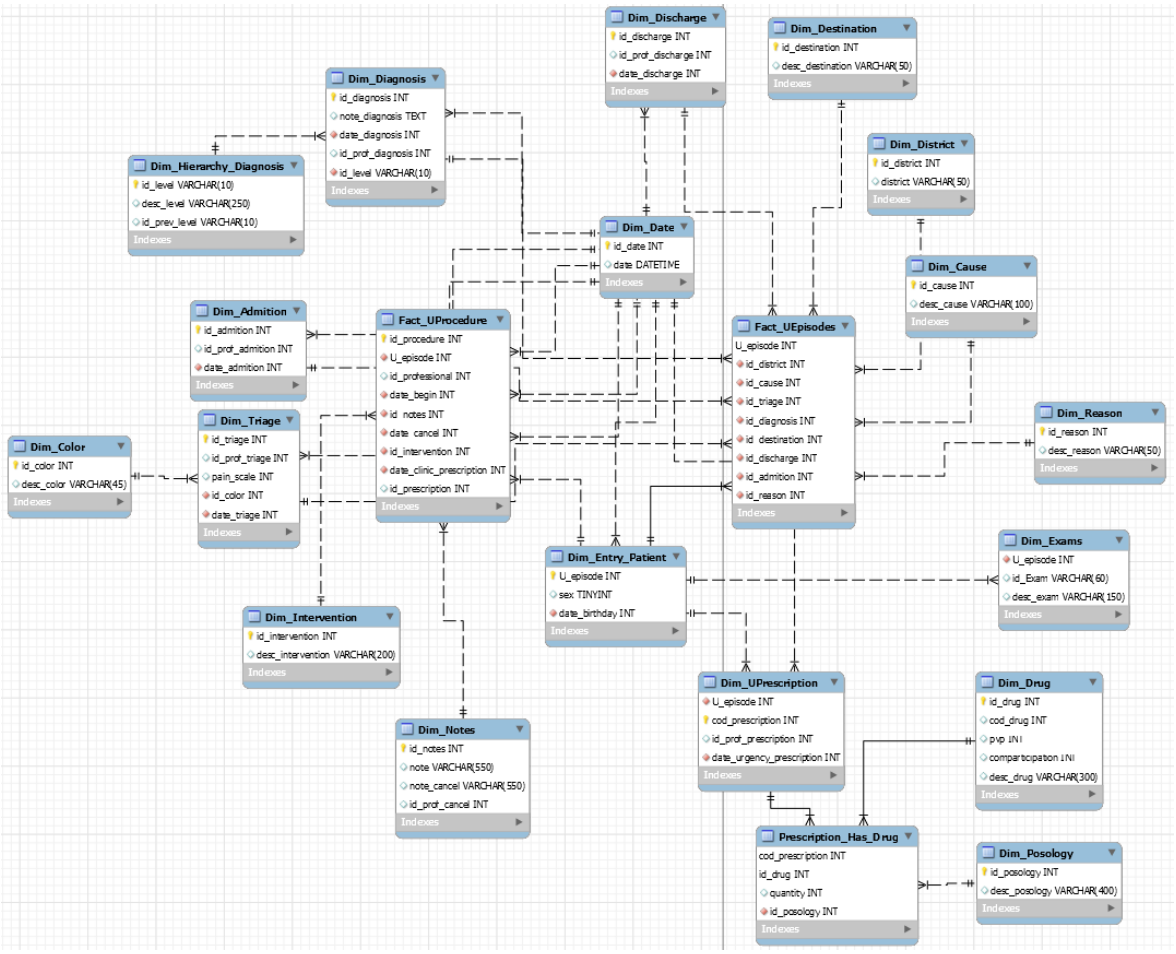


Figura 3.1: Modelo lógico cujo modelo dimensional é esquema em constelação

De seguida encontra-se uma explicação para cada tabela, indicando os valores que esta guarda, bem como as chaves (primária ou estrangeira(s)) que tenha.

Comecemos pela tabela de dimensão `Dim_Date`, que guarda todas as datas existentes dos ficheiros *.csv*, que possui um inteiro, *id_date*, como chave primária e o atributo *date*, que representa a data em si, no formato `yyyy-mm-dd hh:mm:ss`.

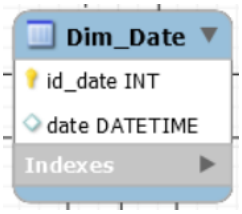


Figura 3.2: Tabela `Dim_Date`

Seguidamente, temos a tabela `Dim_Entry_Patient` que representa um paciente. Esta tem associado o atributo *U_episode*, que é a sua chave primária, o atributo *sex*, que indica o seu género (`0` para feminino e `1` para masculino) e uma chave estrangeira *date_birthday* (da tabela `Dim_Date`), que indica a sua data de nascimento.

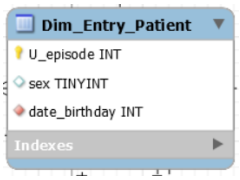


Figura 3.3: Tabela `Dim_Entry_Patient`

Desenvolvemos a tabela de dimensão `Dim_Color`, que remete para a cor usada na triagem do serviço, para diferenciar a gravidade da situação, sendo que cada episódio tem uma cor correspondente. Esta tem, então, o atributo *id_color*, que é chave primária e o atributo *desc_color*, que é a descrição da cor, podendo esta ser "Laranja", "Amarelo" ou "Verde".

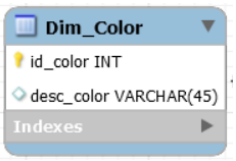


Figura 3.4: Tabela `Dim_Color`

A tabela `Dim_District` representa o distrito onde vive um paciente. Esta tem o *id* incremental, *id_district*, como chave primária e, ainda, o atributo *district*, que indica o nome do distrito em questão.

Figura 3.5: Tabela `Dim_District`

Para guardar informação relativamente à causa do episódio (isto é, o que levou o paciente ao hospital) temos a tabela `Dim_Cause`. Esta é composta pelo atributo *id_cause*, que é a sua chave primária e pelo atributo *desc_cause*, que representa a descrição da causa do episódio.

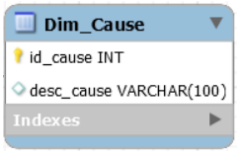


Figura 3.6: Tabela `Dim_Cause`

Para a informação sobre admissões no serviço, foi criada a tabela de dimensão `Dim_Admition`. Esta contém três atributos - a chave primária auto incremental *id_admition*, o atributo *id_prof_admition*, que indica o profissional que fez a admissão do paciente no serviço e uma chave estrangeira *date_admition* (da tabela `Dim_Date`), para saber a data de admissão.

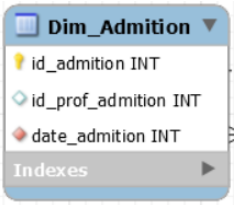


Figura 3.7: Tabela `Dim_Admition`

Já para a informação sobre o destino de um paciente (ARS/Centro de Saúde, Ce Neurologia, entre outras), foi desenvolvida a tabela de dimensão `Dim_Destination`, que contém o atributo inteiro *id_destination*, sendo a sua chave primária e uma descrição sobre o destino, representada pelo atributo *desc_destination*.

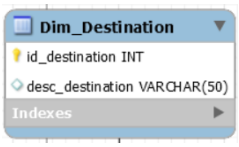


Figura 3.8: Tabela `Dim_Destination`

Relativamente à informação sobre a classificação de doenças (`ICD9`), foi criada a tabela `Dim_Hierarchy_Diagnosis` para guardar os valores pretendidos. Esta contém uma chave primária, *id_level*, que representa o identificador do nível atual, uma descrição do nível em questão fornecida por *desc_level* e o código para o nível anterior ao atual, *id_prev_level*.

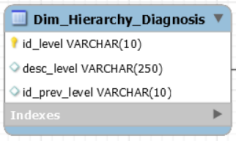


Figura 3.9: Tabela `Dim_Hierarchy_Diagnosis`

Todo o processo de diagnóstico é guardado na tabela `Dim_Diagnosis`. Esta tem um *id* auto incremental como chave primária, *id_diagnosis*, o atributo *note_diagnosis*, que representa toda a nota deixada pelo profissional para o diagnóstico do paciente, uma chave estrangeira

date_diagnosis, para ser possível saber a data em que foi feito o diagnóstico (estando esta na tabela `Dim_Date`), o atributo *id_prof_diagnosis*, que representa o profissional que fez o diagnóstico e, por fim, o atributo que é chave estrangeira e que representa o código de diagnóstico (*id_level* da tabela `Dim_Hierarchy_Diagnosis`) dado por *id_level*.

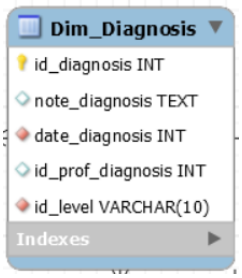


Figura 3.10: Tabela `Dim_Diagnosis`

Para lidar com a informação relativa às altas dadas aos pacientes foi criada a tabela de dimensão `Dim_Discharge` que tem como chave primária um *id* auto incremental, *id_discharge*, como atributo *id_prof_discharge*, que guarda informação relativa ao profissional que deu alta ao paciente e como chave estrangeira *date_discharge* (da tabela `Dim_Date`), para saber a data em o paciente teve alta.

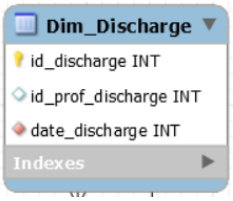


Figura 3.11: Tabela `Dim_Discharge`

Para tratar e guardar a informação sobre a razão para a alta do serviço de urgência foi desenvolvida a tabela de dimensão `Dim_Reason`, tendo como chave primária *id_reason* e como atributo uma descrição da razão para a alta da urgência, *desc_reason*.

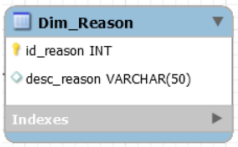


Figura 3.12: Tabela `Dim_Reason`

Toda a informação relacionada com exames, proveniente do *.csv* `urgency_exams`, encontra-se na tabela de dimensão `Dim_Exams`. Esta tem atributos como a chave estrangeira *U_episode* (da tabela `Dim_Entry_Patient`), que remete para o identificador do episódio, o atributo *id_Exam*, que identifica o exame e o atributo *desc_exam*, que representa descrição do exame.

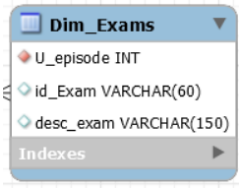


Figura 3.13: Tabela `Dim_Exams`

A informação relativa às prescrições efetuadas pelos profissionais e aos medicamentos presentes nas mesmas está contida em duas tabelas - a `Dim_UPrescription` e a `Dim_Drug`, respetivamente. Visto que uma prescrição pode ter vários medicamentos (*drugs*) e um medicamento pode constar de várias prescrições, isto leva a que exista uma ligação `N:M` entre a tabela de prescrições e a de medicamentos, sendo, então, necessário criar a tabela desta ligação, denominada por `Prescription_Has_Drug`.
No caso da tabela `Dim_UPrescription`, esta tem um atributo *U_episode*, que é chave estrangeira da tabela `Dim_Entry_Patient`, o atributo *cod_prescription*, que é a chave primária da tabela, o *id_prof_prescription*, que remete ao profissional que fez a prescrição ao paciente e uma chave estrangeira da tabela `Dim_Date`, *date_urgency_prescription*, que representa a data em que a prescrição foi efetuada. Já a tabela `Dim_Drug` tem como atributos uma chave primária auto incremental, *id_dug*, um código de medicamento representado por *cod_drug*, o seu preço de venda ao público e comparticipação dados por *pvp* e *comparticipation*, respetivamente e, ainda, o atributo *desc_drug*, que guarda toda a descrição do medicamento.
Quanto à tabela `Prescription_Has_Drug`, que advém da relação `N:M` entre `Dim_UPrescription` e `Dim_Drug`, esta apresenta uma chave primária composta por *id_drug* e *cod_prescription*, sendo ambas chaves estrangeiras (das tabelas referidas), um atributo *quantity*, que indica a quantidade do medicamento e a chave estrangeira da tabela de dimensão `Dim_Posology`, *id_posology*. Esta nova tabela referida contém o atributo *id_posology*, como chave primária auto incremental e o atributo *desc_posology*, que descreve a posologia do medicamento, ou seja, a forma correta de o tomar.

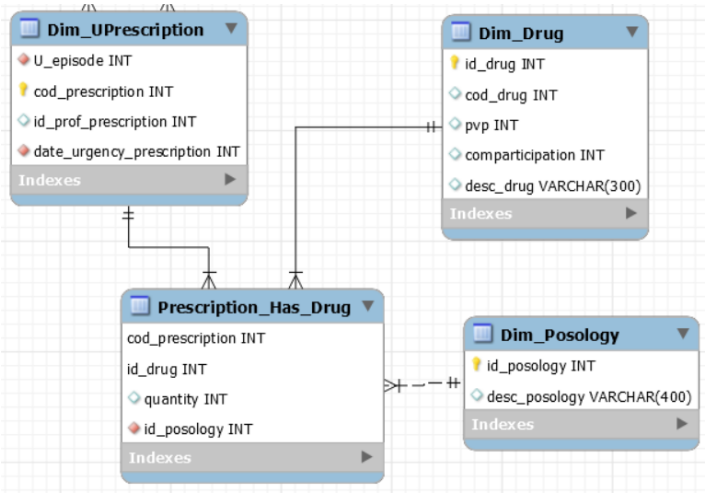


Figura 3.14: Tabelas `Dim_UPrescription`, `Dim_Drug`, `Prescription_Has_Drug` e `Dim_Posology`

Uma intervenção será representada pela tabela `Dim_Intervention`, sendo a sua chave primária o inteiro *id_intervention* e o atributo *desc_intervention*, a descrição da intervenção.

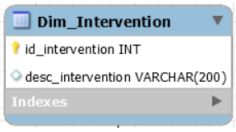


Figura 3.15: Tabela `Dim_Intervention`

De forma a guardar todas as notas de um procedimento, bem como o profissional que as fez, foi criada a tabela `Dim_Notes`. Esta tem uma chave primária auto incremental dada por *id_notes*, o atributo *note*, que descreve a nota do procedimento, o atributo *note_cancel*, que, caso algum procedimento seja cancelado, terá essa informação e um atributo que indica o profissional que cancelou o procedimento, *id_prof_cancel* (caso não tenha sido cancelado, este último atributo apresentará o valor `NULL` até ao seu tratamento, que será explicado na secção 4.4).

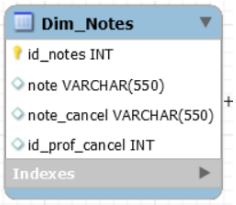


Figura 3.16: Tabela `Dim_Notes`

A triagem feita no serviço é guardada pela tabela `Dim_Triage`. Esta predispõe de vários atributos como uma chave primária auto incremental *id_triage*, um atributo *id_prof_triage*, que representa o profissional que realizou a triagem, o atributo *pain_scale*, que representa a dor, numa escala de `1` a `10`, que o doente sente, o atributo *id_color* (chave estrangeira da tabela `Dim_Color`), para saber qual a cor atribuída ao paciente e, ainda, uma outra chave estrangeira, mas da tabela `Dim_Date`, *date_triage*, que serve para saber a data em que a triagem foi realizada.

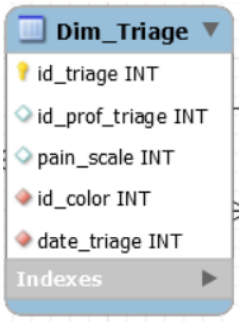


Figura 3.17: Tabela `Dim_Triage`

Uma das tabelas de factos criada é `Fact_UEpisodes`. A sua chave primária, *U_episode*, é um inteiro que representa o número do episódio, sendo também chave estrangeira (da tabela `Dim_Entry_Patient`). Esta é, ainda, constituída por um conjunto de chaves estrangeiras - *id_district* (da tabela `Dim_District`), *id_cause* (da tabela `Dim_Cause`), *id_triage* (da tabela `Dim_Triage`), *id_diagnosis* (da tabela `Dim_Diagnosis`), *id_destination* (da tabela `Dim_Destination`), *id_admition* (da tabela `Dim_Admition`) e *id_reason* (da tabela `Dim_Reason`).

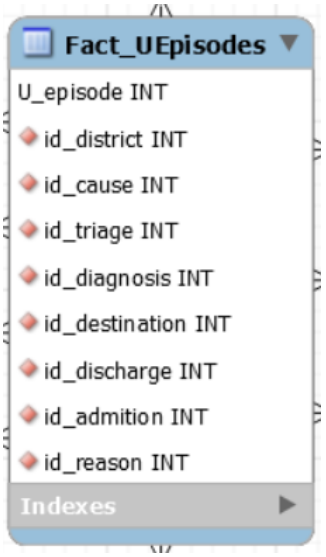


Figura 3.18: Tabela de factos `Fact_UEpisodes`

Por fim, temos a segunda tabela de factos, `Fact_UProcedure`. Esta tem como chave primária um *id* auto incremental, *id_procedure*, e é constituída por algumas chaves estrangeiras - *U_episode* (da tabela `Dim_Entry_Patient`), *date_begin* (da tabela `Dim_Date`), que representa a data de início de um procedimento, *id_notes* (da tabela `Dim_Notes`), *date_cancel* (da tabela `Dim_Date`), que representa a data de cancelamento do procedimento (se não foi cancelado fica a `NULL`), *id_intervention* (da tabela `Dim_Intervention`) e *date_clinic_prescription* (da `Dim_Date`), que representa a data da prescrição clínica. Existem, ainda, dois outros atributos - *id_professional*, que indica o profissional que fez o procedimento e *id_prescription*, que representa o identificador da prescrição clínica.

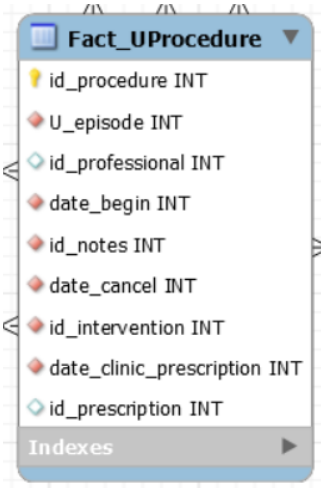


Figura 3.19: Tabela de factos `Fact_UProcedure`

3.2 Modelo Físico

Através do uso do mecanismo *forward engineering*, aplicado sobre o modelo lógico da secção prévia, obtivemos o modelo físico. A estrutura para a criação das tabelas da base de dados *urgency* encontra-se no apêndice A.

Capítulo 4

Povoamento e Gestão do *Data Warehouse*

Para o povoamento do `Data Warehouse` foram utilizados dois métodos - a linguagem de programação `Python`, em conjunto com `Regex` (ou expressões regulares) e a ferramenta `Talend`. Para a inserção de informação nova no `Data Warehouse` utilizámos as rotinas que o `MySQL` dispõe.

4.1 ETL e *Data Warehouse*

O processo `ETL` compreende várias fases sendo estas a extração de informação a partir de diferentes fontes de dados (neste caso, o *dataset* fornecido), a transformação e tratamento desses dados, garantindo a sua consistência (por exemplo, o caso das datas, como será explicado na secção seguinte e o caso das células com valor `NULL` ou vazio, como será abordado na secção 4.4) e o carregamento dos dados para o `Data Warehouse` no modelo físico (isto é, perceber o modelo de negócio e aplicá-lo na construção do modelo lógico (e, consequentemente, do modelo físico)). Assim, um `Data Warehouse` caracteriza-se pela integração e consistência dos dados que contém e serve para guardar grandes coleções de dados (*data*) sobre o negócio onde este está implementado. O seu principal objetivo é o auxílio a organizações na tomada de decisões, com recurso ao seu forte poder de análise de grandes volumes de dados. Para tal análise ser o mais fiável possível, estes dados devem ser tratados, de forma a que o *end user* consiga interpretá-los com a maior exatidão possível. Todavia, estes dados não podem ser tratados sem regras, isto é, é necessário refletir sobre o que é que estes representam e conhecer o negócio onde estão implementados.

4.2 Python

Recorremos à criação de *scripts* em `Python` para o tratamento de datas num formato *standard*, bem como para o povoamento e tratamento dos vários níveis hierárquicos de *International Classification of Diseases* versão `9` (`ICD9`).

Para isso, foram importadas todas as datas dos campos dos vários ficheiros *.csv*. Para além disso, foi necessário formatar todas as datas e horas para um formato comum (`yyyy-mm-dd hh:mm:ss`), retirando as datas repetidas. De seguida, era necessário processar o ano referente à data, isto é, uma data no formato "`01/01/18`" deve ser convertida para "`2018-01-01`", contudo a data "`01/01/19`" deve ser convertida para "`1919-01-01`", devido ao facto de exis-

tirem pacientes com datas de nascimento no ano de `xx19` e com a data de admissão no ano `xx18`, pelo que, a primeira não poderia ser posterior à segunda. Por fim, era necessário tratar das horas, pois algumas destas não continham o campo dos segundos, sendo necessário adicioná-lo ("`:00`").

Para o povoamento da tabela `ICD9`, decidimos colocar todos os níveis numa só tabela, isto é, para cada linha/entrada nesta tabela, existe um *id* do nível, a sua descrição e o identificador do nível anterior. Assim sendo, o mecanismo para transformar o *dataset* de `ICD9` para o formato referido consiste em procurar pelo identificador mais à direita, criando o par (identificador, descrição). De seguida, deve ser associado a este par o identificador mais perto à esquerda e assim sucessivamente para os restantes identificadores de níveis prévios. Quando for encontrado um identificador de nível 1, este é associado ao valor `NULL` e avança-se para a linha seguinte. À imagem do que acontece nas datas, são também retirados os níveis repetidos. Para além disto, constatámos que alguns identificadores eram referenciados no *.csv* `urgency_episodes_new`, contudo não se encontravam presentes no *.csv* com a hierarquia `ICD9`. Por esse motivo, optámos por adicionar estes identificadores e utilizar a descrição contida no primeiro *.csv* referido.

4.3 Talend

Para os restantes dados utilizámos a ferramenta `Talend`, todavia, para fazer uso desta, foi necessário modificar o formato dos ficheiros *.csv*, tendo adicionado delimitadores mais simples, garantindo, ainda, que não existiam delimitadores dentro do valor de um campo. Outro procedimento necessário, antes de começar a importar o *dataset*, foi a formatação das datas para o formato estabelecido na secção anterior.
Para estas duas operações, recorremos a expressões regulares, de modo a capturar os excertos "alvo" e substituí-los pela(s) transformação(ões) correspondente(s).

De seguida, importámos os vários *.csv* do *dataset* e começámos a criar os mecanismos/ condições de povoamento.

4.3.1 Procedimento para o povoamento

O povoamento de todas as tabelas que se encontram no modelo lógico (secção 3.1) foi realizado com o auxílio do *software* `Talend`, excluindo as tabelas `Dim_Date` e `Dim_Hierarchy_Diagnosis` que foram povoadas a partir do `MySQL`, com *scripts* em `Python`.

É de ressalvar que, todas as tabelas em análise possuem entradas únicas, isto é, se existir uma entrada/conjunto de entradas que sejam iguais ao conjunto único, esta(e) não vai ser colocada(o) nas linhas do povoamento.
Assim e para o povoamento das tabelas `Dim_Reason`, `Dim_Destination`, `Dim_Cause`, `Dim_District` e `Dim_Color` foi utilizado o *.csv* `urgency_episodes_new`.
Para a tabela `Dim_Reason`, o conjunto único é composto por *id_reason* e *desc_reason*, o que faz com que apenas existam `9` entradas na tabela após percorrer as `65617` entradas que o *.csv* possui. Já para a tabela `Dim_Destination`, o conjunto único é constituído por *id_destination* e *desc_destination*, fazendo com que sejam introduzidas `99` entradas diferentes. No que diz respeito à tabela `Dim_Cause` (que representa a *external cause* do *.csv*), o seu conjunto único é composto por *id_cause* e *desc_cause*, sendo povoadas `15` entradas distintas. No caso da tabela `Dim_District`, a única coluna que tem de ser única é o *district*, uma vez que o *id_district* é

um valor incremental, obtendo, assim, 26 entradas na tabela. Por fim, a tabela `Dim_Color` tem o seu o conjunto único constituído por *id_color* e *desc_color*, povoando um total de 3 entradas diferentes.

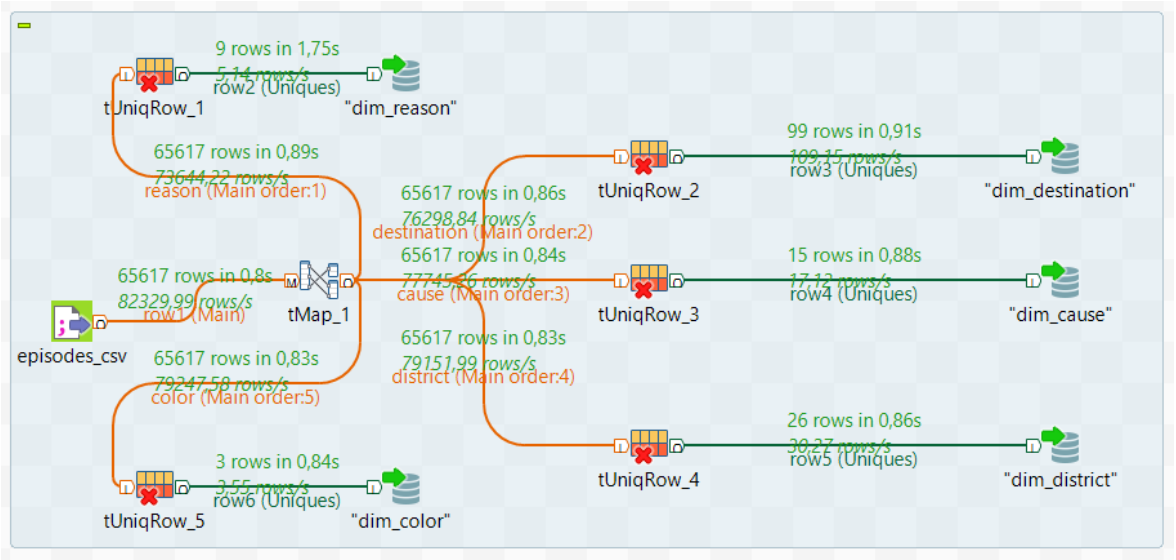


Figura 4.1: Povoamento das tabelas `Dim_Reason`, `Dim_Destination`, `Dim_Cause`, `Dim_District` e `Dim_Color`

De seguida, para o povoamento das tabelas de dimensão `Dim_Drug` e `Dim_Posology` foi utilizado o *.csv* `urgency_prescriptions`. Para a primeira, o conjunto único é formado por *cod_drug*, *pvp*, *comparticipation* e *desc_drug*, sendo que existem 4085 entradas únicas. Já no caso da segunda, o único atributo que necessita de ser único é o *desc_posology*, tendo, então, um total de 26174 entradas.

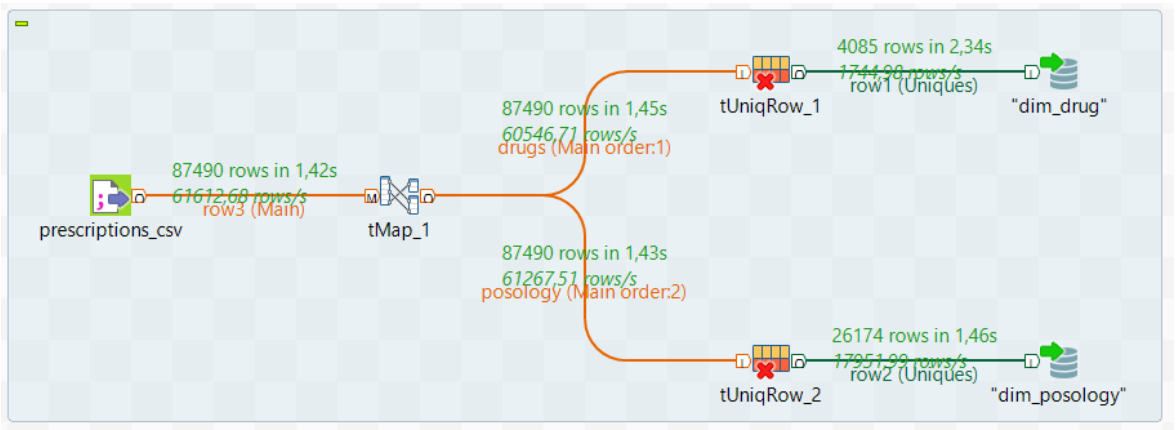


Figura 4.2: Povoamento das tabelas `Dim_Drug` e `Dim_Posology`

As tabelas de dimensão `Dim_Notes` e `Dim_Intervention` foram povoadas com o recurso ao *.csv* `urgency_procedures`. No caso da primeira, por existirem algumas notas repetidas, foi decidido que o conjunto único seria constituído pelas colunas *note*, *note_cancel* e *id_prof_cancel*, obtendo 3594 entradas únicas. Já para a segunda, o seu conjunto único é composto por *id_intervention* e *desc_intervention*, perfazendo um total de 103 entradas.

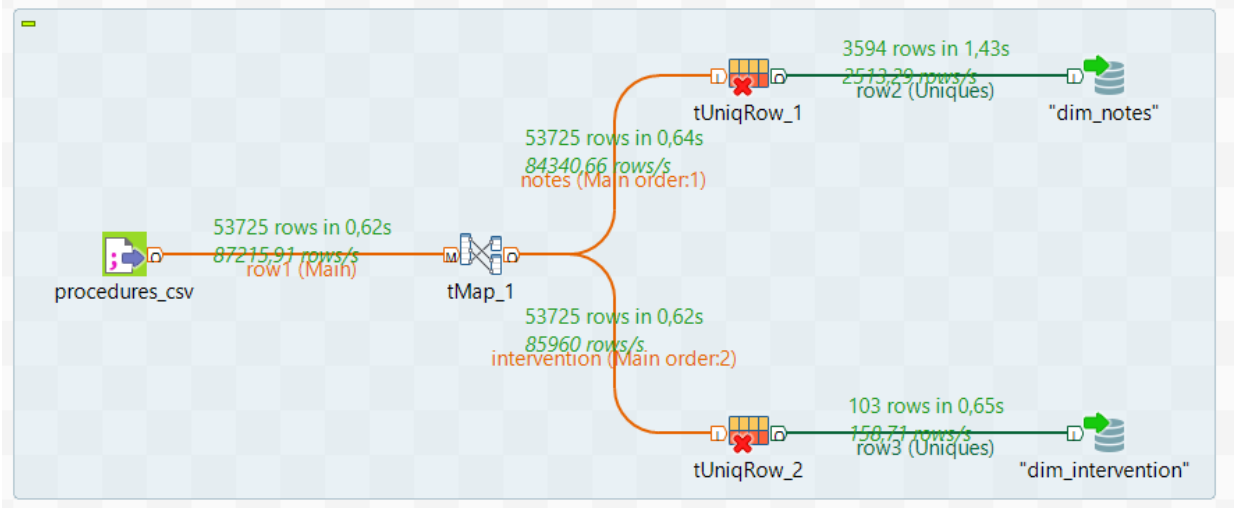


Figura 4.3: Povoamento das tabelas `Dim_Notes` e `Dim_Intervention`

A etapa seguinte consistiu em povoar as tabelas `Dim_Admition`, `Dim_Discharge`, `Dim_Triage` e `Dim_Diagnosis`, recorrendo ao *.csv* `urgency_episodes_new`. Uma vez que cada uma destas tabelas tem uma data que precisa de ser povoada e é necessário saber qual o *id* para tal data, foi necessário realizar quatro *lookups* à tabela `Dim_Date`, para encontrar os *ids* das datas correspondentes. Estas foram convertidas para o formato *string*, de forma a que a probabilidade das comparações entre elas falharem seja nula. Assim, a tabela `Dim_Admition` tem o seu conjunto único composto por *id_prof_admition* e *date_admition*. Já para a `Dim_Discharge`, o conjunto único é constituído por *id_prof_discharge* e *date_discharge*. No que diz respeito às entradas na tabela `Dim_Triage`, o seu conjunto único é formado por *id_prof_triage*, *pain_scale*, *id_color* e *date_triage*. Por fim, a tabela `Dim_Diagnosis` tem definido como único o conjunto composto por *note_diagnosis*, *date_diagnosis*, *id_prof_diagnosis* e *id_level*.

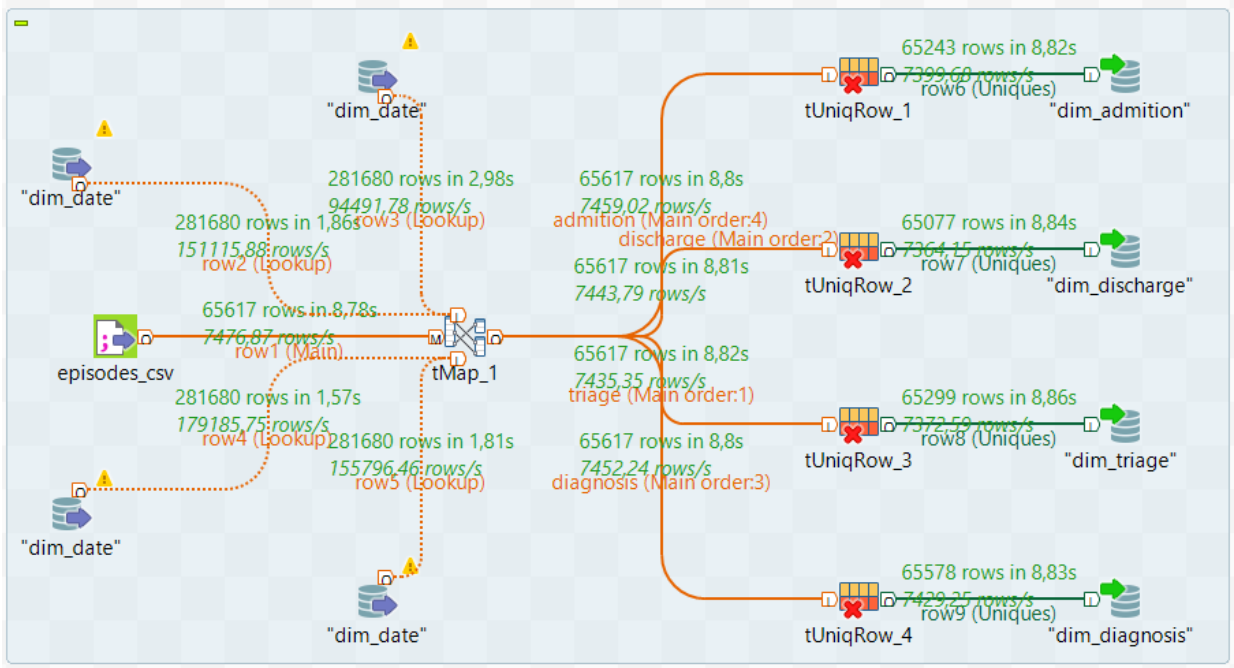


Figura 4.4: Povoamento das tabelas `Dim_Admition`, `Dim_Discharge`, `Dim_Triage` e `Dim_Diagnosis`

Para povoar a tabela de dimensão `Dim_Entry_Patient` é necessário efetuar um *lookup* à tabela `Dim_Date` para saber qual a chave que corresponde à data de nascimento do paciente[1]. O valor do género do paciente que se encontra no *.csv* usa um "`F`" para representar o género feminino e um "`M`" para representar o género masculino, no entanto, optámos por converter o valor "`F`" para `0` (*false*) e o valor "`M`" para `1` (*true*), utilizando, para isso, a fórmula `row1.SEX == 'F' ? false:true` em que *row1* representa uma entrada do *.csv* `urgency_episodes_new`. Uma vez que não existem episódios repetidos e que a chave primária desta tabela é o número do episódio, não é necessário referir que as entradas têm de ser únicas.

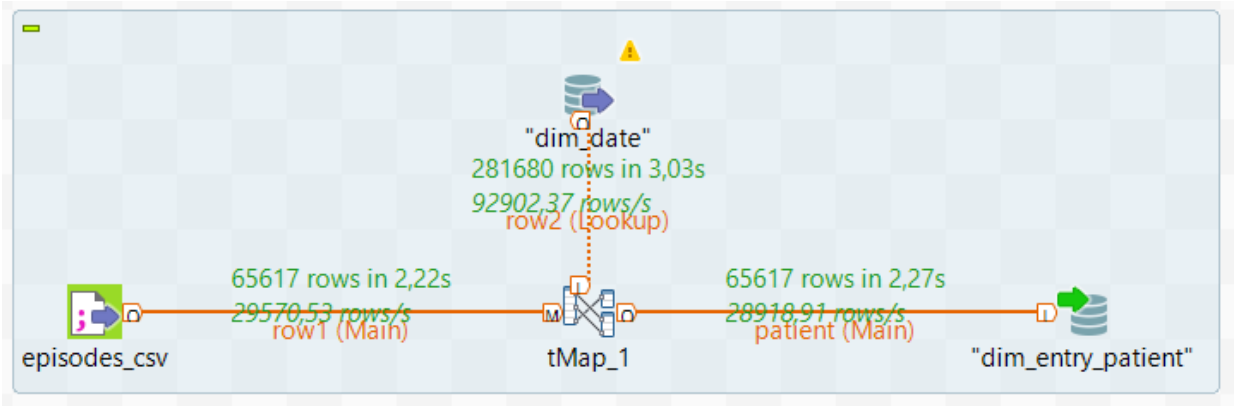


Figura 4.5: Povoamento da tabela `Dim_Entry_Patient`

O povoamento da tabela de dimensão `Dim_Exams` foi efetuado utilizando o *.csv* `urgency_exams`. Neste caso, o conjunto único é composto por *U_episode*, *id_Exam* e *desc_exam*. É possível observar, na figura seguinte, que nunca existem entradas iguais, no entanto, no futuro, não podemos afirmar com a máxima certeza que o *.csv* não contenha entradas repetidas, por esse motivo, optámos por manter o `tUniqRow`.

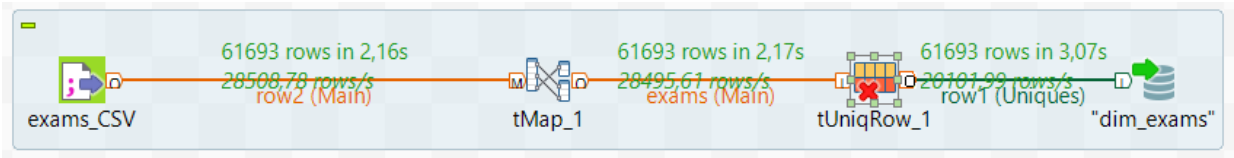


Figura 4.6: Povoamento da tabela `Dim_Exams`

A primeira tabela de factos `Fact_UEpisodes` foi povoada com recurso ao *.csv* `urgency_episodes_new`. Esta tabela, para cada entrada, necessita de saber a chave correspondente das tabelas `Dim_Admition`, `Dim_Triage`, `Dim_Diagnosis`, `Dim_Discharge`, `Dim_District`, `Dim_Cause`, `Dim_Destination` e `Dim_Reason`. Para saber as chaves das quatro primeiras tabelas referidas, recorremos ao uso de um `tMap` adicional que serve como auxílio para os *join*, uma vez que estas tabelas precisam, também, de saber a chave da data correspondente (sendo que, cada uma delas, tem uma data). Na figura abaixo podemos observar o que é que o `tMap` de auxílio irá fazer e a informação que passará ao próximo `tMap`, para descobrir as chaves das tabelas de dimensão referidas anteriormente.

[1]Novamente, a data que é lida no *lookup* e do *.csv* foi convertida para o formato *string*, mitingando a existência de erros nas comparações efetuadas.

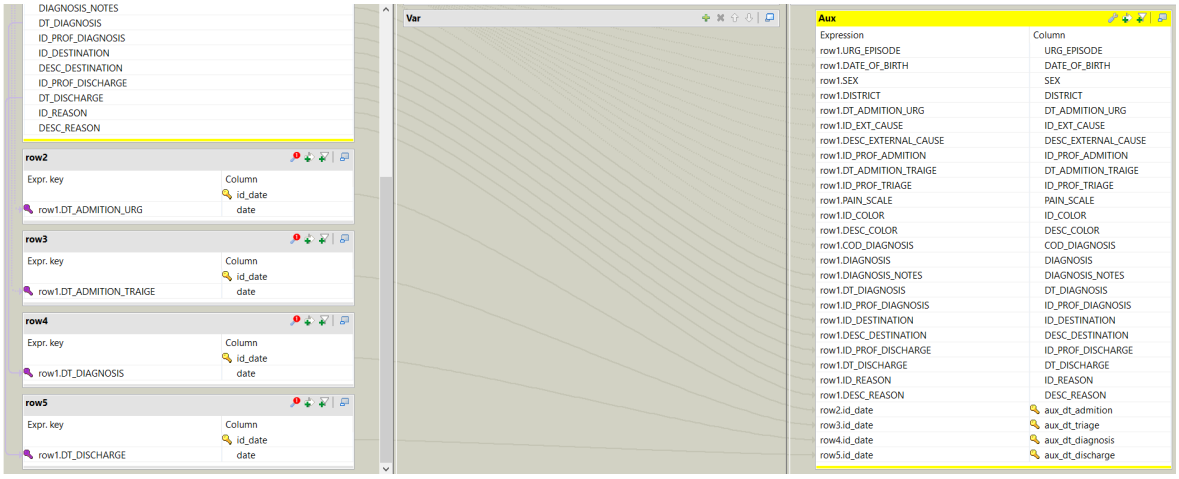


Figura 4.7: `tMap` que serve de auxílio para descoberta de chaves

De seguida, basta ligar este `tMap` de auxílio a um outro `tMap`, sendo que, este último, já conhece as chaves para as datas que necessita, tornando, assim, possível a descoberta das entradas das tabelas `Dim_Admition`, `Dim_Triage`, `Dim_Diagnosis` e `Dim_Discharge` que correspondem. Para todo este processo, é necessário efetuar quatro *lookups* à tabela `Dim_Date` e um *lookup* às tabelas `Dim_District`, `Dim_Discharge`, `Dim_Triage`, `Dim_Admition` e `Dim_Diagnosis`, conforme se pode verificar na figura seguinte.

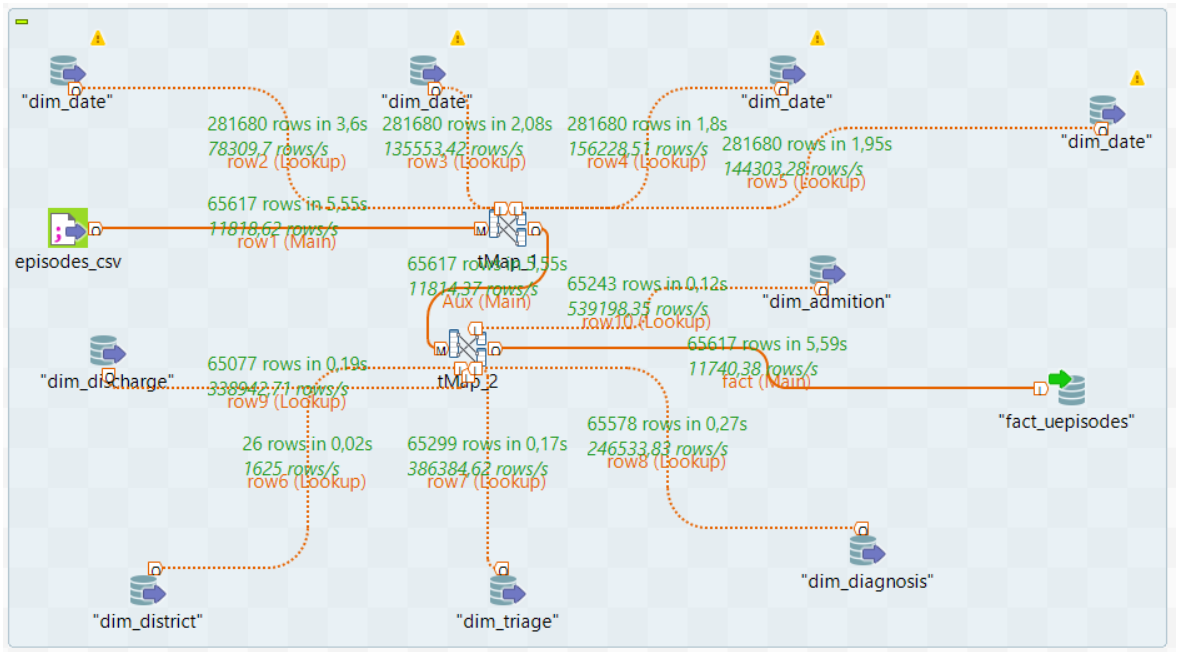


Figura 4.8: Povoamento da tabela de factos `Fact_UEpisodes`

Para o povoamento da tabela de dimensão `Dim_UPrescription`, foi necessário realizar um *lookup* à tabela `Dim_Date` para saber a chave a que corresponde a data da prescrição. Foi, ainda, necessário definir um `tUniqRow` para não permitir entradas com o mesmo *cod_prescription*.

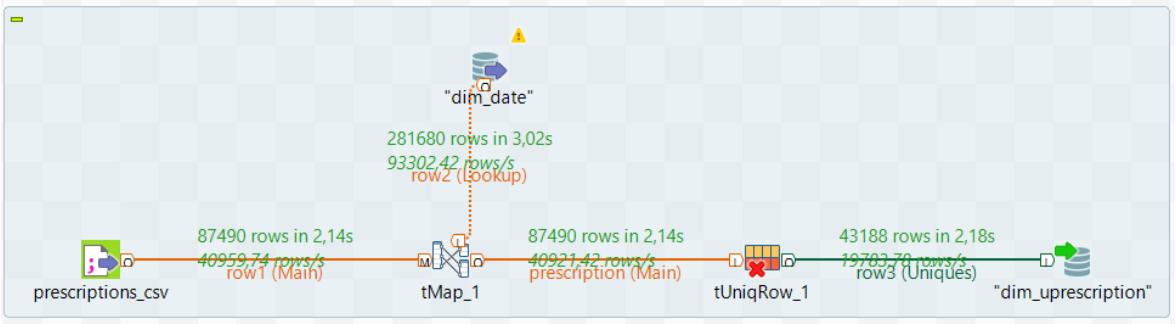


Figura 4.9: Povoamento da tabela de dimensão `Dim_UPrescription`

De seguida, avançámos para o povoamento da tabela `Prescription_Has_Drugs`, tabela esta que não necessita da definição de entradas únicas (isto é, um episódio pode ter várias prescrições de diferentes medicamentos receitados, mas estas não podem ser iguais). Foi necessário efetuar um *lookup* às tabelas `Dim_Drug` e `Dim_Posology`.

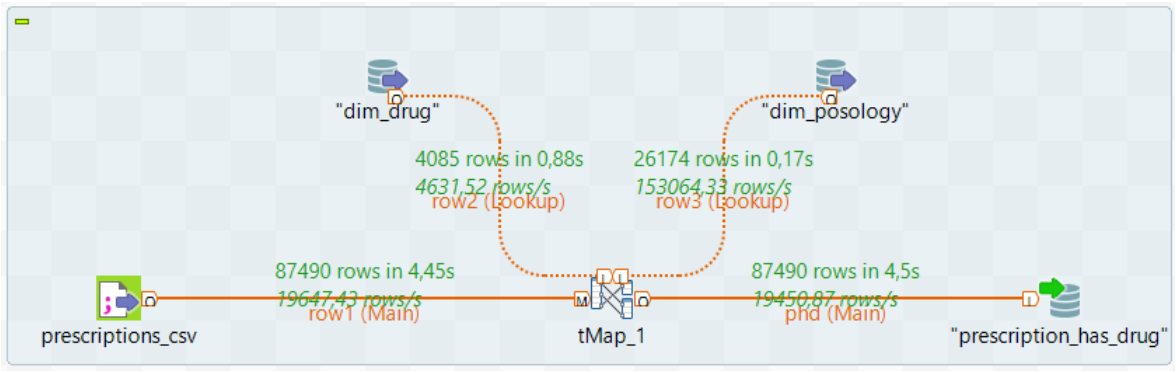


Figura 4.10: Povoamento da tabela `Prescription_Has_Drugs`

Por fim, a segunda tabela de factos `Fact_UProcedure` foi povoada com recurso a três *lookups* à tabela de dimensão `Dim_Date` e um *lookup* às tabelas `Dim_Intervention` e `Dim_Notes`.

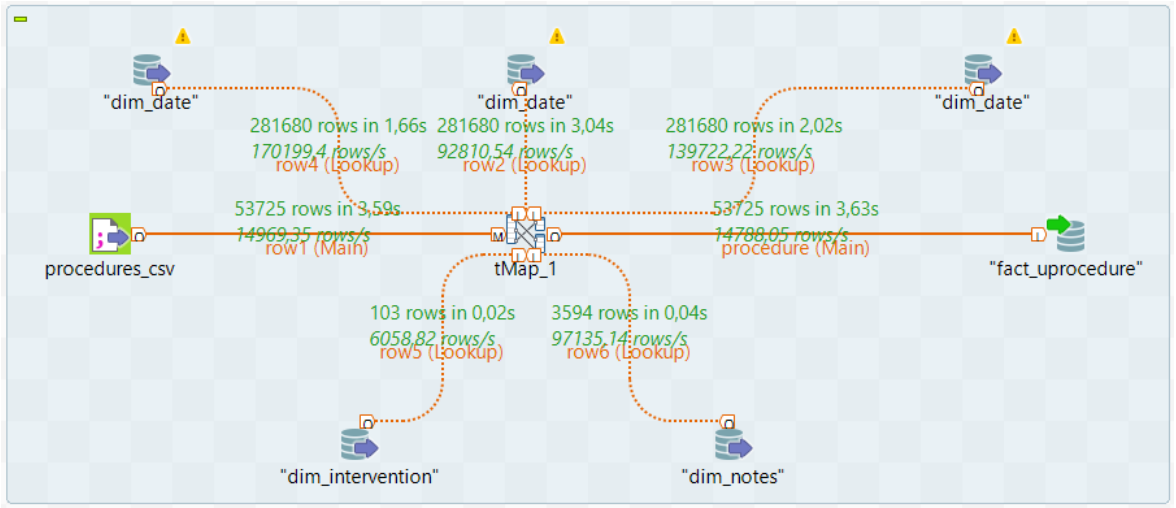


Figura 4.11: Povoamento da tabela de factos `Fact_UProcedure`

4.4 Tratamento de valores

Os valores *id_prof_discharge* (da tabela `Dim_Discharge`) e *id_prof_cancel* (da tabela `Dim_Notes`) foram tratados como casos especiais, isto é, sempre que apresentam o valor `NULL`, este é imediatamente alterado para o valor "`-1`", uma vez que um `NULL` pode ser interpretado de forma diferente da pretendida, pelo que o valor pelo qual se altera deve ter um significado mais claro, de forma a ser entendido por quem o irá utilizar.
É de realçar que todas as datas que se encontram a `NULL` foram deixadas com esse valor, pois, caso fosse atribuída uma data específica a um caso de valor `NULL`, esta pode alterar a veracidade da entrada em questão.
Por fim, foi, ainda, considerada a mudança dos valores de notas (da tabela `Dim_Notes`) que não continham informação para outros valores como, por exemplo, "*Não existe informação*", no entanto esta mudança pode ter uma interpretação muito variada no mundo da medicina[2], pelo que optámos por apenas colocar uma *string* vazia (" ").

```
DELIMITER $$
DROP PROCEDURE IF EXISTS handling_null_values;
CREATE PROCEDURE handling_null_values()
BEGIN
    START TRANSACTION;
        UPDATE Dim_Discharge SET id_prof_discharge = -1  WHERE id_prof_discharge IS NULL;
        UPDATE Dim_Notes SET id_prof_cancel = -1 WHERE (note_cancel = "" and id_prof_cancel is null);
    COMMIT;
END $$
```

Figura 4.12: Tratamento de valores `NULL` em `SQL`

4.5 Vantagens e Desvantagens Entre os Dois Processos

A utilização de uma linguagem como `Python` permite realizar um *debug* mais simples e oferece uma maior flexibilidade no tratamento de dados. No entanto, não tem os mecanismos automatizados como uma ferramenta como o `Talend`, sendo que é necessário definir todas as condições de combinação/ligação entre tabelas/dados, enquanto que no `Talend` podemos utilizar a funcionalidade *drag-and-drop*, isto é, arrastar os vários "componentes" para definir como é que os dados devem ser ligados (por exemplo, dizer que um *id* tem de ser o *id* de outra tabela). Além disto, o `Talend` permite compreender/analisar mais facilmente o fluxo de acontecimentos e, com a criação de diferentes *jobs*, as operações de normalização, mapeamento de colunas e filtragem de dados tornam-se ainda mais simples. Por outro lado, não existe muita documentação detalhada na *internet*, o que pode tornar o processo de povoamento mais demorado.
Outra possibilidade, seria a utilização de `MySQL` para o povoamento, contudo é algo complexo e demorado, pois é necessário fazer *joins* entre diferentes tabelas (para certas tabelas). Para além disso, é necessário ter alguma prática na linguagem e algum conhecimento mais extenso sobre o modelo do `Data Warehouse`.

[2]Por exemplo, o facto de não conter informação pode apenas significar que alguém não a escreveu e não que esta não existe.

4.6 MySQL - Criação de mecanismos de evolução

Dado que um `Data Warehouse` tem de possuir certas características (referidas na secção 4.1) é evidente que, ao longo da sua evolução, terá de as manter.
O `Data Warehouse` criado está orientado a um serviço de urgências hospitalares, sendo garantido que está orientado a uma área específica de saúde. Como será possível verificar no capítulo seguinte e aliado a um sistema de `Business Intelligence`, este foi pensado para possibilitar uma fácil modelação e análise de dados (*subject-oriented*). Este sistema serve-se de um classificador internacional de doenças (`ICD9`), mantendo-se, assim, integrado (*integrated*) com outras bases de dados "mundiais".
Um `Data Warehouse` tem, também, de possuir um histórico de operações realizadas, sendo que, para garantir isso, não foram realizados *procedures* que possibilitam a alteração/mutação de conhecimento (*time-variant*). Analogamente, os dados são não voláteis (*non-volatile*), isto é, não podem ser apagados/removidos, pelo que não foram realizados quaisquer *procedures/functions* que permitam tal operação.
Para garantir que era possível continuar a evoluir o `Data Warehouse` desenvolvido, decidimos criar *functions* (funções), *procedures* (procedimentos) e *triggers* em `MySQL`, para suportar a inserção de novos dados.
Como iremos ver ao longo desta secção, existem algumas funções que poderiam ser procedimentos, no entanto estas apresentam uma enorme vantagem, pois, caso seja pretendido inserir novo conhecimento numa tabela em que a sua chave primária é utilizada como chave estrangeira numa outra, podemos retornar diretamente a chave primária, evitando assim um *select* adicional, que teríamos de fazer caso utilizássemos um *procedure*. Decidimos, também, que, ao inserir um novo episódio, seria útil ser o sistema a gerar automaticamente o seu novo *id*, de modo a evitar enganos inseridos pelo utilizador, garantindo, assim, uma utilização sequencial dos episódios.

Primeiramente, iremos explicar todos os procedimentos/funções necessários para a inserção de conhecimento nas várias tabelas com ligação à tabela de factos `Fact_UEpisodes`. Devido à necessidade de suportar a evolução de conhecimento, é necessária a inserção de novos campos nas tabelas de dimensão ligadas à ela.

Começando por apresentar o procedimento que insere um entrada na tabela `Dim_Entry_Patient`, isto é, uma entrada de um paciente no sistema, que é constituída por um *id* do episódio, género e data de nascimento. É, ainda, de realçar que o *id* do episódio é gerado automaticamente, usando uma variável que será explicada posteriormente (figura 4.37). Esta é uma das tabelas, onde são guardados todos os episódios do sistema.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_Entry_Patient;$$
CREATE PROCEDURE ins_Entry_Patient(sex tinyint, birthday_date datetime)
BEGIN
  DECLARE u_ep INT DEFAULT 0;
  START TRANSACTION;
    INSERT INTO Dim_Entry_Patient (U_episode,sex,date_birthday)
    VALUES (@last_ep + 1,sex,ins_date(birthday_date));
  COMMIT;
END $$
```

Figura 4.13: Procedimento que insere uma entrada de um paciente no sistema

Visto que, quando um paciente entra no serviço de urgência de um hospital, as primei-

ras "etapas" são a admissão e triagem, é necessário associá-las ao episódio do paciente em questão. Portanto, é necessária a inserção de novo conhecimento nas tabelas `Dim_Admition` e `Dim_Triage`, levando, assim, à criação das seguintes funções:

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Admition;$$
CREATE FUNCTION ins_Admition(id_prof_ad INT, dt DATETIME) RETURNS INT
BEGIN
    INSERT INTO Dim_Admition (id_prof_admition,date_admition)
    VALUES (id_prof_ad,ins_date(dt));
    RETURN LAST_INSERT_ID();
END $$
```

Figura 4.14: Função responsável pela admissão de um paciente no serviço de urgência

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Triage;$$
CREATE FUNCTION ins_Triage(id_pt INT, painS INT,id_color INT,triage_date DATETIME)
RETURNS INT
BEGIN
    INSERT INTO Dim_Triage (id_prof_triage,pain_scale,id_color,date_triage)
    VALUES (id_pt,painS,id_color,ins_date(triage_date));
    RETURN LAST_INSERT_ID();
END $$
```

Figura 4.15: Função responsável pela inserção de entradas na triagem

Similar ao processo anterior, temos agora o caso do diagnóstico. Sempre que um paciente dá entrada e é tratado no serviço de urgência, este é diagnosticado por um profissional de saúde, tendo de classificar o problema do doente na hierarquia pré-definida `ICD9`, podendo, ainda, acrescentar uma nota relevante para o diagnóstico. Para isso, foi criada a seguinte função:

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Diagnosis;$$
CREATE FUNCTION ins_Diagnosis(note_dia TEXT,date_dia DATETIME,id_prof_dia
INT,id_level VARCHAR(10)) RETURNS INT
BEGIN
  IF (exist_level(id_level)) THEN
    INSERT INTO Dim_Diagnosis (note_diagnosis,date_diagnosis,id_prof_diagnosis,
    id_level)
    VALUES (note_dia,ins_date(date_dia),id_prof_dia,id_level);
    return LAST_INSERT_ID();
  END IF;
  return -1;
END $$
```

Figura 4.16: Função responsável pela inserção de diagnósticos

Uma vez que os exames auxiliam nos diagnósticos realizados pelos médicos e, visto que estes estão armazenados na tabela `Dim_Exams`, foi criado um procedimento que insere um exame realizado por um dado paciente:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_Exam;$$
CREATE PROCEDURE ins_Exam(U_ep INT,id_Exam VARCHAR(60),desc_Exam VARCHAR(150))
BEGIN
  START TRANSACTION;
    IF EXISTS(SELECT U_episode FROM Dim_Entry_Patient where U_episode=U_ep) THEN
      INSERT INTO Dim_Exams (U_episode,id_exam,desc_exam)
      VALUES (U_ep,id_Exam,desc_Exam);
    END IF;
  COMMIT;
END $$
```

Figura 4.17: Procedimento usado para inserir um exame de um paciente

A seguinte função permite a inserção de novos distritos no sistema, garantindo sempre que não existem distritos repetidos.

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_District;$$
CREATE FUNCTION ins_District(name1 VARCHAR(50)) RETURNS INT
BEGIN
  DECLARE id_max INT DEFAULT 0;
  Set id_max = (SELECT id_district FROM Dim_District as d where d.district=name1);
    IF (id_max is NULL) THEN
        INSERT INTO Dim_District (district) VALUES (name1);
        RETURN LAST_INSERT_ID();
    END IF;
    RETURN id_max;
END $$
```

Figura 4.18: Função que permite inserção de novos distritos

Para um paciente poder receber alta hospitalar, esta tem de ser dada por um médico, devendo, ainda, ser armazenada a data e hora da alta, informação esta que é guardada na tabela `Dim_Discharge`. De modo a ser possível inserir novas altas para os pacientes, foi criada a seguinte função:

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Discharge;$$
CREATE FUNCTION ins_Discharge(id_prof_dis INT, dt DATETIME) RETURNS INT
BEGIN
  INSERT INTO Dim_Discharge (id_prof_discharge,date_discharge)
  VALUES (id_prof_dis,ins_date(dt));
  RETURN LAST_INSERT_ID();
END $$
```

Figura 4.19: Função responsável pela inserção de altas no sistema

Sempre que um paciente tem alta, pode ter vários destinos (como o seu domicílio ou consulta externa), pelo que é necessário inserir essa informação na tabela `Dim_Destination`. Este processo é feito pelo seguinte procedimento:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_Destination;$$
CREATE PROCEDURE ins_Destination(id_dest INT ,dest VARCHAR(50))
BEGIN
  INSERT INTO Dim_Destination (id_destination,desc_destination)
  VALUES (id_dest,dest);
END $$
```

Figura 4.20: Procedimento responsável pela inserção do *id* de destino e da sua descrição

A seguinte função é similar ao procedimento anterior, porém apenas recebe o `Destino`, sendo calculado o *id* automaticamente (optámos pelo uso de uma função, pois é necessário retornar o *id* gerado).

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Destination2;$$
CREATE FUNCTION ins_Destination2(dest VARCHAR(50)) RETURNS INT
BEGIN
  DECLARE id INT DEFAULT 0;
  Set id = (SELECT id_destination FROM Dim_Destination as d where
  d.desc_destination=dest);
  IF (id is NULL) THEN
    SELECT MAX(id_destination)+1 INTO id FROM Dim_Destination;
    INSERT INTO Dim_Destination (id_destination,desc_destination) VALUES (id,dest);
    return id;
  END IF;
  return id;
END $$
```

Figura 4.21: Função responsável pela inserção de um destino (descrição)

Na tabela `Dim_Cause` são armazenados os dados relativos ao motivo do incidente, isto é, o que levou o paciente ao serviço de urgência. Estes são armazenados na tabela referida e, para suportar a inserção de novas causas, foi criado o seguinte procedimento:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_Cause;$$
CREATE PROCEDURE ins_Cause(id_c INT ,cause VARCHAR(100))
BEGIN
  INSERT INTO Dim_Cause (id_cause,desc_cause) VALUES (id_c,cause);
END $$
```

Figura 4.22: Procedimento responsável pela inserção do *id* da causa e da sua descrição

A seguinte função é similar ao procedimento anterior, porém apenas recebe a `Causa`, sendo o *id* gerado automaticamente (novamente, optámos pelo uso de uma função, pois é necessário retornar o *id* gerado).

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Cause2;$$
CREATE FUNCTION ins_Cause2(cause VARCHAR(100))RETURNS INT
BEGIN
  DECLARE id_max INT DEFAULT 0;
  Set id_max = (SELECT id_cause FROM Dim_Cause as d where d.desc_cause=cause);
  IF (id_max is NULL) THEN
        SELECT MAX(id_cause)+1 INTO id_max FROM Dim_Cause;
        INSERT INTO Dim_Cause (id_cause,desc_cause) VALUES (id_max,cause);
        return id_max;
  END IF;
  return id_max;
END $$
```

Figura 4.23: Função responsável pela inserção da causa (descrição)

A tabela `Dim_Reason` armazena informação relativa ao motivo de alta hospitalar. Deste modo, foi necessário criar um procedimento, para ser possível inserir uma nova causa para a alta:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_Reason;$$
CREATE PROCEDURE ins_Reason(id_r INT ,reason VARCHAR(50))
BEGIN
  INSERT INTO Dim_reason (id_reason,desc_reason) VALUES (id_r,reason);
END $$
```

Figura 4.24: Procedimento responsável pela inserção do *id* da razão da alta hospitalar e da sua descrição

A seguinte função é similar ao procedimento anterior, porém apenas recebe a `Razão` pela qual a alta foi dada, sendo calculado o *id* automaticamente (novamente, optámos pelo uso de uma função, pois é necessário retornar o *id* gerado).

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Reason2;$$
CREATE FUNCTION ins_Reason2(reason VARCHAR(50)) RETURNS INT
BEGIN
    DECLARE id_max INT DEFAULT 0;
    Set id_max = (SELECT id_reason FROM Dim_Reason as d where d.desc_reason=reason);
    IF (id_max is NULL) THEN
        SELECT MAX(id_reason)+1 INTO id_max FROM Dim_reason;
        INSERT INTO Dim_reason (id_reason,desc_reason) VALUES (id_max,reason);
        return id_max;
    END IF;
    return id_max;
END $$
```

Figura 4.25: Função responsável pela inserção da razão da alta hospitalar (descrição)

De modo a associar toda a informação diretamente relacionada com o paciente e o episódio, é utilizada a tabela de factos `Fact_UEpisodes`. Aqui são armazenados todos os identificadores das tabelas de dimensão ligadas a esta, de forma a ser possível inserir novo conhecimento nas mesmas. Este procedimento executa várias funções, descritas anteriormente, para facilitar a inserção nas tabelas dependentes.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_Fact_Uepisode;$$
CREATE PROCEDURE ins_Fact_Uepisode(ep INT,district VARCHAR(50),cause
VARCHAR(100),triage_prof INT,triage_pain INT,triage_color INT,triage_date
DATETIME,dia_note TEXT,dia_date DATETIME,dia_prof INT,dia_lvl VARCHAR(10),dest
VARCHAR(50),discharge_prof INT,discharge_date DATETIME,admition_prof INT,
admition_date DATETIME,reason VARCHAR(50))
BEGIN
    START TRANSACTION;
        INSERT INTO Fact_UEpisodes (U_episode,id_district,id_cause,id_triage,
        id_diagnosis,id_destination,id_discharge,id_admition,id_reason)
        VALUES (ep,ins_District(district),ins_Cause2(cause),
        ins_Triage(triage_prof,triage_pain,triage_color,triage_date),
        ins_Diagnosis(dia_note,dia_date,dia_prof,dia_lvl),ins_Destination2(dest),
        ins_Discharge(discharge_prof,discharge_date),
        ins_Admition(admition_prof,admition_date),ins_Reason2(reason));
    COMMIT;
END $$
```

Figura 4.26: Procedimento que insere conhecimento na tabela de factos `Fact_UEpisodes`

A diferença entre o procedimento seguinte e o anterior é que este apenas permite inserir um novo episódio com conhecimento já existente, isto é, todas informações do episódio devem existir no `Data Warehouse`, pois são utilizados os identificadores das tabelas de dimensão.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_Fact_Uepisode2;$$
CREATE PROCEDURE ins_Fact_Uepisode2(ep INT ,district INT, cause INT,triage INT,dia
INT,dest INT,discharge INT,admition INT,reason INT)
BEGIN
    INSERT INTO Fact_UEpisodes (U_episode,id_district,id_cause,id_triage,
    id_diagnosis,id_destination,id_discharge,id_admition,id_reason)
    VALUES (ep,district, cause,triage,dia,dest,discharge,admition,reason);
END $$
```

Figura 4.27: Procedimento que insere conhecimento na tabela de factos `Fact_UEpisodes` com *ids* já conhecidos

Passando agora para explicação de como é feita a inserção na tabela de factos `Fact_UProcedure`. Tendo em conta que esta tabela depende da existência de entradas nas tabelas de dimensão relacionadas, as chamadas às funções respetivas serão efetuadas aquando da inserção dos valores que irão tratar da adição dessas mesmas entradas. O procedimento resultante é o seguinte:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_UProcedure;$$
CREATE PROCEDURE ins_UProcedure(u_ep INT, id_pro INT, date_begin DATETIME,note
VARCHAR(550), note_cancel VARCHAR(550), id_prof_c INT,date_cancel DATETIME, id_i
INT,Desc_Inter VARCHAR(200), date_cli DATETIME, id_pres INT)
BEGIN
    START TRANSACTION;
        INSERT INTO Fact_UProcedure (U_episode,id_professional,date_begin,
        id_notes,date_cancel,id_intervention,date_clinic_prescription,id_prescription)
        VALUES (u_ep,id_pro,ins_date(date_begin),ins_Note(note,note_cancel,
        id_prof_c),ins_date(date_cancel),ins_Intervention(id_i,Desc_Inter),
        ins_date(date_cli),id_pres);
    COMMIT;
END $$
```

Figura 4.28: Procedimento que insere uma entrada de um procedimento no serviço de urgências

Abaixo são, então, apresentadas as duas funções principais chamadas no procedimento anterior.
A primeira apresenta dois casos para a inserção na tabela `Dim_Intervention` - caso o *id* da intervenção indicado exista/esteja já inserido na tabela, então este é retornado, caso contrário, é feita a adição dessa entrada na tabela das intervenções e é retornado o *id* criado.

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Intervention;$$
CREATE FUNCTION ins_Intervention(id INT, desc_int VARCHAR(200)) RETURNS INT
BEGIN
    DECLARE id_i INT DEFAULT 0;
    Set id_i = (SELECT id_intervention FROM Dim_Intervention where (id_intervention
    = id OR desc_intervention = desc_int));
    IF (id_i is NULL) THEN
        INSERT INTO Dim_Intervention (id_intervention,desc_intervention)
        VALUES (id, desc_int);
        RETURN id;
    END IF;
    RETURN id_i;
END $$
```

Figura 4.29: Função responsável pela inserção de uma intervenção

Já a segunda apresenta um método idêntico, mas para tabela `Dim_Notes` - caso o *id* indicado pertença a uma entrada da tabela, este é retornado, caso contrário, é passado o novo *id* da entrada criada na tabela.

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Note;$$
CREATE FUNCTION ins_Note(note VARCHAR(550), note_c VARCHAR(550), id_p INT) RETURNS
INT
BEGIN
  DECLARE id INT DEFAULT 0;
  Set id = (SELECT id_notes FROM Dim_Notes as d where d.note=note AND
  d.note_cancel=note_c AND d.id_prof_cancel=id_p);
  IF (id is NULL) THEN
        INSERT INTO Dim_Notes (note,note_cancel,id_prof_cancel)
        VALUES (note, note_c, id_p);
        RETURN LAST_INSERT_ID();
  END IF;
  RETURN id;
END $$
```

Figura 4.30: Função responsável pela inserção de uma nota

Semelhante às funções e procedimentos da tabela de factos `Fact_UProcedure`, existe a tabela `Prescription_Has_Drug` que também necessita de uma atenção extra durante o processo de inserção. Visto que esta se encontra ligada à tabela `Dim_Posology`, para realizar uma inserção, é preciso que exista o método responsável pelo tratamento das entradas da tabela relacionada. Assim, a inserção na tabela `Dim_Posology` é feita com base na descrição da posologia indicada. Se existir uma entrada em que esta descrição esteja presente, é retornado o *id* respetivo, caso contrário, é criada uma nova entrada com a descrição fornecida e um novo *id* gerado, retornando-o.

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_Posology;$$
CREATE FUNCTION ins_Posology(desc_p VARCHAR(400)) RETURNS INT
BEGIN
  DECLARE id INT DEFAULT 0;
  Set id = (SELECT id_posology FROM Dim_Posology as p where p.desc_posology =
  desc_p);
  IF (id is NULL) THEN
        INSERT INTO Dim_Posology (desc_posology) VALUES (desc_p);
        RETURN LAST_INSERT_ID();
  END IF;
  RETURN id;
END $$
```

Figura 4.31: Função responsável pela inserção de uma posologia

De seguida, na tabela `Dim_Drug`, são indicados quatro dados que servirão para comparação - código do medicamento, preço, comparticipação e descrição. Como não existe um medicamento igual com estes dados diferentes, apenas são procuradas na tabela as entradas com estas quatro informações em simultâneo. Caso essa entrada exista é, então, retornado esse *id*, caso contrário é criada uma nova entrada com essas informações e devolvido o *id* resultante.

```
DROP FUNCTION IF EXISTS ins_Drug;$$
CREATE FUNCTION ins_Drug(cod INT, pvp INT, comp INT, desc_d VARCHAR(300))  RETURNS
INT
BEGIN
  DECLARE id INT DEFAULT 0;
  Set id = (SELECT id_Drug FROM Dim_Drug as d where d.cod_drug = cod AND d.pvp = pvp
  AND d.comparticipation = comp AND d.desc_drug = desc_d);
  IF (id is NULL) THEN
      INSERT INTO Dim_Drug (cod_drug,pvp,comparticipation,desc_drug)
      VALUES (cod, pvp, comp, desc_d);
      RETURN LAST_INSERT_ID();
  END IF;
  RETURN id;
END $$
```

Figura 4.32: Função responsável pela inserção de um medicamento

Finalmente, para a tabela `Prescription_Has_Drug`, é fornecido o *id* do medicamento que servirá para procurar pela existência de uma entrada na tabela com esse valor e, tal como nos métodos anteriores, caso exista, é retornado esse mesmo *id*, caso contrário, é devolvido o *id* da entrada adicionada.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ins_Prescription_Has_Drug;$$
CREATE PROCEDURE ins_Prescription_Has_Drug(cod INT, id_drug INT, quant INT, posology
VARCHAR(400))
BEGIN
    DECLARE id_d INT DEFAULT 0;
    START TRANSACTION;
        Set id_d = (SELECT id_Drug FROM Dim_Drug as d where d.id_drug = id_drug);
        IF NOT (id_d is NULL) THEN
            INSERT INTO Prescription_Has_Drug
            (cod_prescription,id_drug,quantity,id_posology)
            VALUES (cod,id_drug,quant,ins_Posology(posology));
        END IF;
    COMMIT;
END $$
```

Figura 4.33: Procedimento que insere uma entrada na tabela em que uma prescrição refere um medicamento

Como realizado para todas as tabelas anteriores, resta, então, efetuar um método que introduza conhecimento na tabela `Dim_UPrescription` e, como se trata de uma tabela relativamente simples, apenas é necessário fornecer o identificador do episódio, o código de prescrição, o *id* de prescrição e a data da prescrição.

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_UPrescription;$$
CREATE FUNCTION ins_UPrescription(u_ep INT, cod INT, id_p INT, date_u DATETIME)
RETURNS INT
BEGIN
    INSERT INTO Dim_UPrescription (U_episode,cod_prescription,
    id_prof_prescription,date_urgency_prescription)
    VALUES (u_ep, cod, id_p, ins_date(date_u));
    RETURN cod;
END $$
```

Figura 4.34: Função responsável pela inserção de uma prescrição nas urgências

A função apresentada de seguida é usada para verificar se um dado nível do `ICD9` existe na tabela `Dim_Hierarchy_Diagnosis`. Esta pode ser vista como uma função auxiliar do *procedure* responsável por inserir um diagnóstico (`ins_Diagnosis`).

```
DELIMITER $$
DROP FUNCTION IF EXISTS exist_level $$
CREATE FUNCTION exist_level(lvl VARCHAR(10)) RETURNS TINYINT
BEGIN
    DECLARE var TINYINT DEFAULT false;
    IF((SELECT COUNT(id_level) FROM Dim_Hierarchy_Diagnosis where id_level LIKE lvl)
    = 1) THEN
        SET var=true;
    END IF;
    return var;
END $$
```

Figura 4.35: Função responsável por verificar se um certo nível do `ICD9` existe na tabela correspondente

Na tabela `Dim_Date` são armazenadas todas as datas do sistema, fazendo desta tabela a maior da base de dados, sendo, também, a razão pela qual utilizámos uma variável que guarda o último identificador inserido. Assim, sempre que uma data não exista na tabela, esta é inserida, sendo posteriormente retornado o seu *id*.

```
DELIMITER $$
DROP FUNCTION IF EXISTS ins_date $$
CREATE FUNCTION ins_date(date1 DATETIME) RETURNS INT
BEGIN
    DECLARE id_d INT DEFAULT 0;
    SET id_d = (SELECT id_date FROM Dim_Date as d WHERE d.date = date1);
    IF (id_d is NULL) THEN
        SET id_d = @last_date + 1;
        INSERT INTO Dim_Date VALUES (id_d,date1);
    END IF;
    return id_d;
END $$
```

Figura 4.36: Função responsável por inserir uma data no sistema

De forma a realizar menos *selects* aquando da inserção de datas e pacientes nas tabelas `Dim_Date` e `Dim_Entry_Patient` respetivamente, foram criadas duas variáveis inteiras que guardam o identificador mais alto para cada uma das tabelas. A variável `last_ep`, caso seja

declarada após o povoamento, é inicializada com o valor do *id* máximo dos episódios, caso contrário é inicializada com o valor `0`. A variável `last_date` é semelhante, porém, no caso de ser declarada previamente ao povoamento, é inicializada com o valor `-1`, devido à existência do identificador `0` na tabela `Dim_Date`.

```
SET @last_ep = (SELECT COALESCE(MAX(U_episode), 0)  FROM Dim_Entry_Patient);
SET @last_date = (SELECT COALESCE(MAX(id_date), -1)  FROM Dim_Date);
```

Figura 4.37: Variáveis que guardam o identificador mais alto dos episódios e das datas, respetivamente

Devido aos identificadores apresentados anteriormente serem uma espécie de *auto-increment*, sempre que existe uma inserção nas tabelas respetivas, é necessário aumentar o valores destas variáveis. Para isso, são usados os seguintes *triggers*:

```
DELIMITER $$
DROP TRIGGER IF EXISTS act_date;$$
CREATE TRIGGER act_date AFTER INSERT ON Dim_Date FOR EACH ROW
BEGIN
  SET @last_date = @last_date + 1 ;
END$$
```

Figura 4.38: *Trigger* responsável por aumentar a variável das datas

```
DELIMITER $$
DROP TRIGGER IF EXISTS act_Uep;$$
CREATE TRIGGER act_Uep AFTER INSERT ON Dim_Entry_Patient FOR EACH ROW
BEGIN
  SET @last_ep = @last_ep + 1 ;
END$$
```

Figura 4.39: *Trigger* responsável por aumentar a variável do episódio

Capítulo 5

Business Intelligence - Indicadores Clínicos

Para permitir criar uma camada de interação mais simples e rápida e desenvolver sistemas de suporte à decisão na área em que o `Data Warehouse` está inserido, desenvolvemos um sistema de `Business Intelligence`, com recurso à ferramenta `Tableau Desktop`, que permite a criação de *dashboards* com vários gráficos para análise.

5.1 *Dashboard* sobre Episódios de Urgência

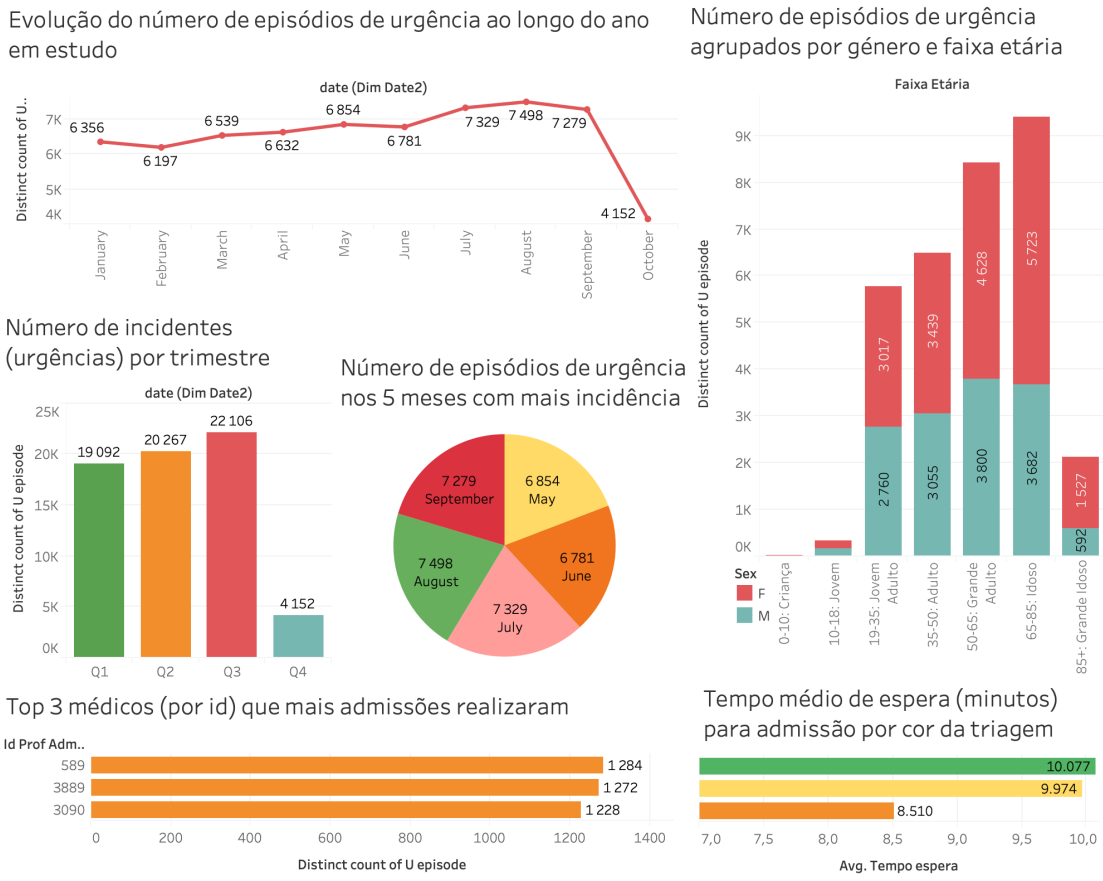


Figura 5.1: *Dashboard* Episódios de Urgência

Com este *dashboard* pretendemos efetuar uma análise sobre os dados relativos aos episódios de urgência em geral, incluindo, também, algumas informações sobre as admissões. Com isto é possível garantir/verificar que os tempos de espera estão dentro do esperado e analisar a afluência ao serviço de urgência para, por exemplo, construir escalas de serviço melhor distribuídas ou perceber quando é que os profissionais de saúde podem gozar os seus períodos de férias, sem prejudicar o fluxo. Assim, o objetivo global deste *dashboard* é fornecer informações relevantes sobre aspetos positivos ou negativos para o serviço de urgência e/ou para o doente.

No primeiro gráfico, podemos constatar que, ao longo do ano em estudo, os meses com mais episódios de urgência são os de Agosto, Julho e Setembro (por ordem decrescente), situação que é expectável, tendo em conta que estes constituem os meses de verão, em que a população se encontra de férias e, tendencialmente, existem mais quedas, acidentes de viação, afogamentos, entre outros. Os meses com menos episódios são os de Fevereiro (pois este mês tem menos dias (em 2018 teve 28)) e Outubro (visto que a última entrada para este é `18/10/2018 06:34`, acreditámos que a razão pela qual é o mês com menor afluência se deve ao facto de não estar completo, pois, caso a média de casos se mantivesse, teríamos, aproximadamente, `7150` casos no final do mês, tornando-o no quarto com maior afluência).

No segundo gráfico, podemos observar os valores do gráfico anterior agrupados por trimestre. Como seria de esperar, o terceiro trimestre (Julho, Agosto e Setembro) é onde se verifica o maior número de incidentes. É de ressalvar que o quarto trimestre tem bastante menos episódios de urgência, pois os dados não são completos (falta parte de Outubro, Novembro e Dezembro).

No terceiro gráfico, podemos confirmar os valores do primeiro gráfico, observando o *Top 5* de meses com mais incidentes, sendo estes, do maior para o menor, Agosto, Julho, Setembro, Maio e Junho.

No quarto gráfico, podemos constatar quais as faixas etárias que têm mais episódios de urgência e qual o género desses pacientes. A faixa etária de Idoso (`65-85`) é a que mais episódios tem, seguida da de Grande Adulto (`50-65`), o que se revela expectável, pois, a partir dos 50 anos, é natural a existência de mais doenças ou patologias próprias da idade (como a hipertensão)[1] e um aumento da probabilidade de descompensação das mesmas. A faixa etária como menos episódios é de Criança (`0-10`), possivelmente por não existir no hospital considerado uma urgência pediátrica que funcione a todas as horas do dia, levando à necessidade de os pais se deslocarem a outros hospitais. É também de notar que o género feminino é o mais predominante em todas as faixas etárias, o que pode parecer algo inesperado, pois, por norma, os homens tendem a exibir comportamentos de risco e alguma tendência à adoção de hábitos menos saudáveis que podem influenciar certas doenças, como a alimentação não regrada, consumo de tabaco ou álcool, entre outros (ou, ainda, pelo facto de a Lombalgia (referida na secção 5.3) ser uma patologia mais comum em indivíduos do género masculino). Todavia, usualmente, as mulheres têm mais facilidade em assumir/perceber que não se sentem bem e que necessitam de procurar cuidados médicos. Em particular, no caso das faixas etárias Jovem Adulto e Adulto, esta maior afluência também pode ser explicada por um maior número de mulheres grávidas nessas idades, enquanto que no caso da faixa etária Grande Idoso, esta pode ser explicada pelo facto de a esperança média de vida para indivíduos do género feminino ser maior do que para indivíduos do género oposto[2].

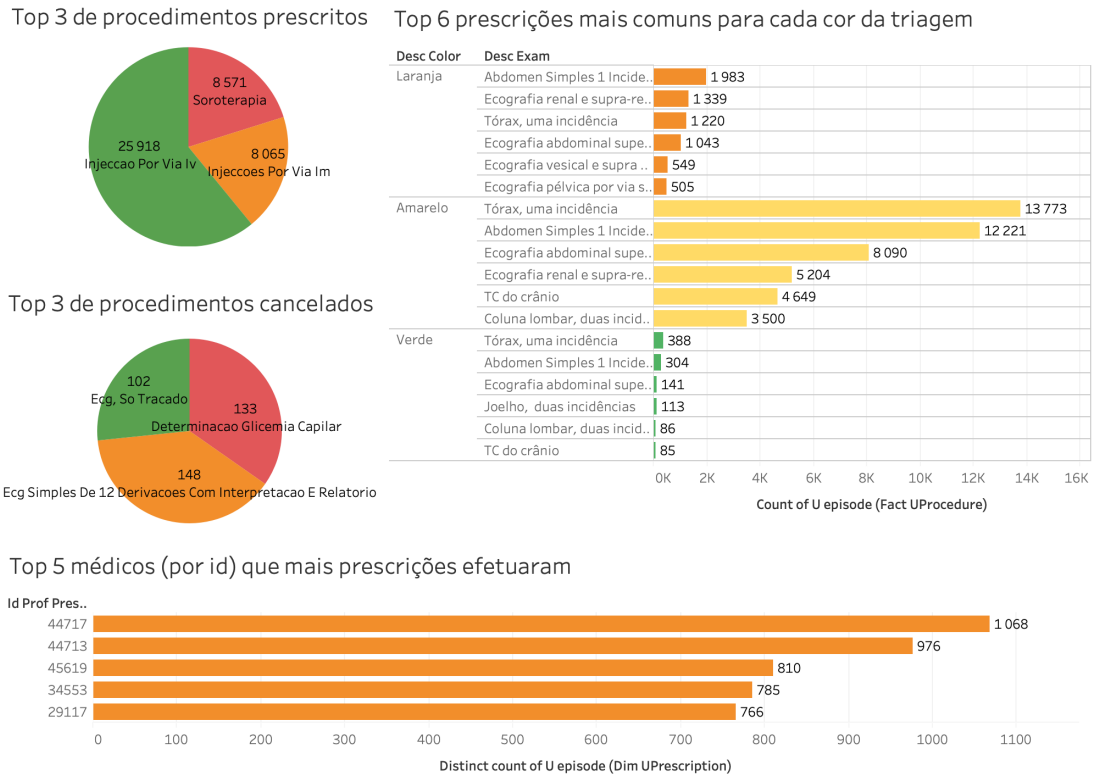
[1] É de ressalvar que, a partir dessa idade, também existem mais programas de rastreio para certos tipos de doenças, precisamente pelo risco acrescido que o aumento de idade traz.

[2] Fonte: `https://www.sns.gov.pt/noticias/2019/09/26/portugal-esperanca-de-vida-aumenta/`

No quinto gráfico, podemos verificar os três médicos que mais admissões realizaram, muito possivelmente, por serem aqueles que mais turnos fazem no serviço de urgência (podem até, por exemplo, ser "tarefeiros" contratados especificamente para estas funções).

Por fim, no sexto gráfico, podemos observar o tempo médio de espera (em minutos) para a admissão no serviço de urgência, por cor de triagem. Como seria de esperar, a cor Laranja é admitida mais rapidamente que a Amarela que, por sua vez, é mais célere que a Verde. Todavia, é de ressalvar que, de acordo com a escala da Triagem de *Manchester*[3], o tempo de espera para a cor Laranja encontra-se dentro do esperado, mas o tempo de espera para as cores Amarelo e Verde está bastante abaixo do esperado (60 e 120 minutos, respetivamente), o que é um forte indicador do bom funcionamento/organização deste serviço de urgência.

5.2 *Dashboard* sobre Procedimentos e Prescrições



[3]Fonte: `http://www.grupoportuguestriagem.pt/index.php?option=com_content&view=article&id=4&Itemid=110`

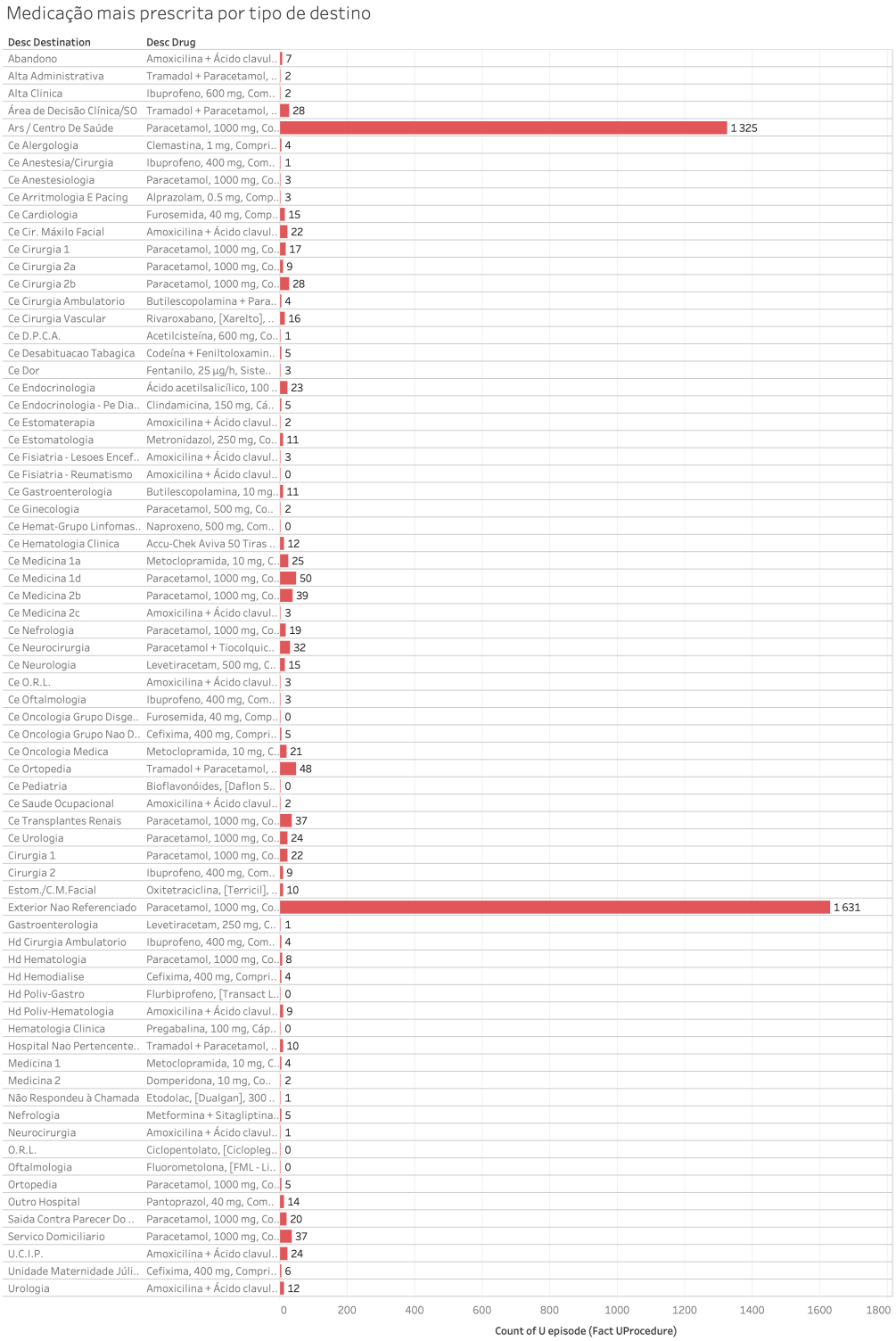


Figura 5.2: *Dashboard* Procedimentos e Prescrições

Com este *dashboard* pretendemos efetuar uma análise sobre os dados relativos aos procedimentos e prescrições realizados durante o ato de urgência. Como isto é possível perceber, por exemplo, quais os profissionais que são necessários para realizar certo tipo de procedimentos mais prescritos ou qual a medicação que deve ter o seu *stock* reposto mais frequentemente. Assim, o objetivo geral deste *dashboard* é explorar a dinâmica de funcionamento deste serviço.

No primeiro gráfico, podemos verificar o *Top 3* de procedimentos prescritos, sendo que o mais prescrito é a Injeção Por Via IV (intravenosa), seguido da Soroterapia e Injeção Por Via IM (intramuscular). Este resultado, no nosso entender, encontra-se muito próximo da realidade, pois, em bastantes casos, é necessário administrar medicação por via intravenosa ou através do soro (que serve não só este propósito, como também o de hidratar, por exemplo) ou, ainda, por via intramuscular (no caso de antibióticos, como a Penicilina).

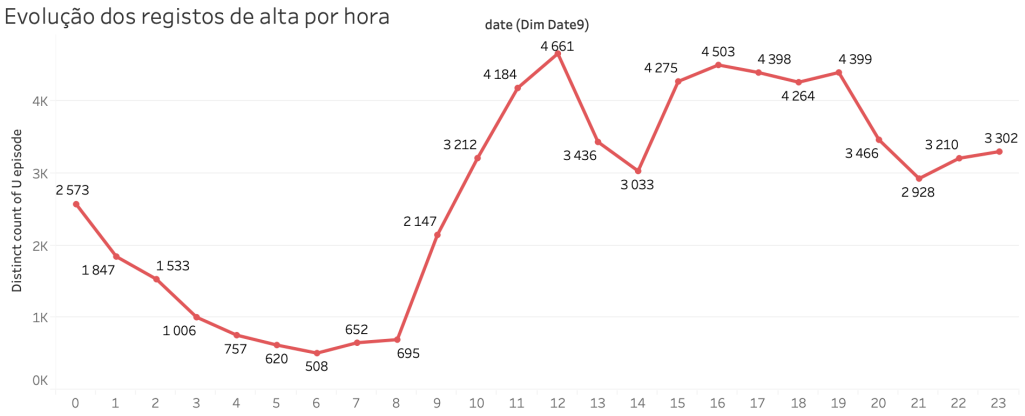
No segundo gráfico, podemos observar o *Top 3* de procedimentos cancelados, sendo que o mais cancelado é o ECG Simples de 12 derivações com interpretação e relatório, seguido da Determinação da Glicemia Capilar e ECG, só Traçado (sem relatório). Da análise realizada ao *.csv* `urgency_procedures` e com recurso a algumas fórmulas, encontrámos três possíveis explicações para o cancelamento: ou o código do procedimento foi mal introduzido na prescrição, pelo que é necessário cancelá-lo e prescrever o correto ou já haviam sido realizados outros procedimentos que apontavam para um diagnóstico e, após análise dos resultados, deixou de existir a necessidade de efetuar mais procedimentos ou, ainda, após a passagem de turno médico e posterior reavaliação dos pacientes, estes já não reuniam critérios clínicos para a realização dos procedimentos previamente prescritos.

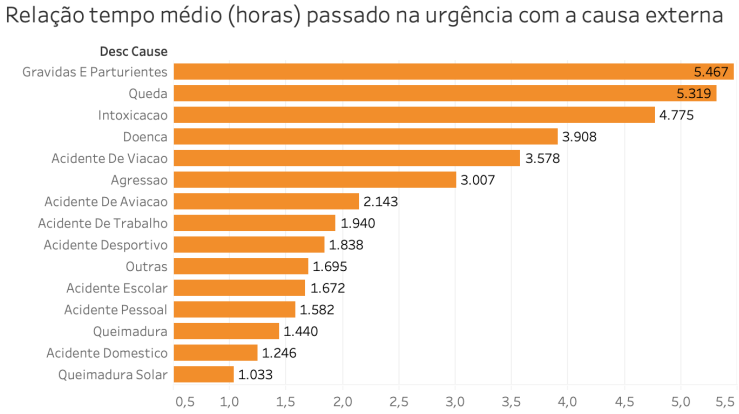
No terceiro gráfico, podemos analisar as seis prescrições mais comuns para cada cor da triagem. No nosso entender, seria expectável que cores menos prioritárias de triagem tivessem prescrições de exames mais "simples". Para as cores Verde e Amarelo os três exames mais comuns são realizados ao Tórax (radiografia) e Abdómen (radiografia e ecografia). No caso da cor Amarelo seguem-se exames realizados aos Rins, Crânio e Coluna Lombar (por ordem decrescente), já no caso da cor Verde seguem-se exames realizados ao Joelho, Coluna Lombar e Crânio (por ordem decrescente). No caso da cor mais prioritária, Laranja, os exames são realizados ao Abdómen (radiografia), Rins, Tórax, Abdómen (ecografia), Bexiga e Região Pélvica (por ordem decrescente). Como se pode perceber, a cor Laranja possui prescrições de exames para zonas mais específicas quando comparada com as restantes cores, como é o caso da Bexiga ou da Região Pélvica (exames estes que podem ser efetuados, por exemplo, a grávidas que, por norma, necessitam de atendimento prioritário). Uma possível explicação para a existência de exames ao Crânio para as cores Amarelo e Verde e não para a Laranja pode ser o facto de certas Quedas (referidas na secção 5.3) não serem tão severas (caso seja o próprio a deslocar-se ao serviço de urgência, por exemplo), contudo poderá ser necessária uma avaliação imagiológica no caso de o indivíduo ter sofrido embate com alguma região da cabeça.

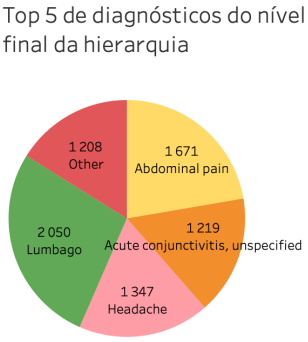
No quarto gráfico, podemos constatar os cinco médicos que mais prescrições realizaram, muito possivelmente, por serem aqueles que receberam casos menos sérios, com exames mais simples e rápidos e/ou que apenas necessitavam de prescrições de medicamentos para continuar o tratamento no domicílio.

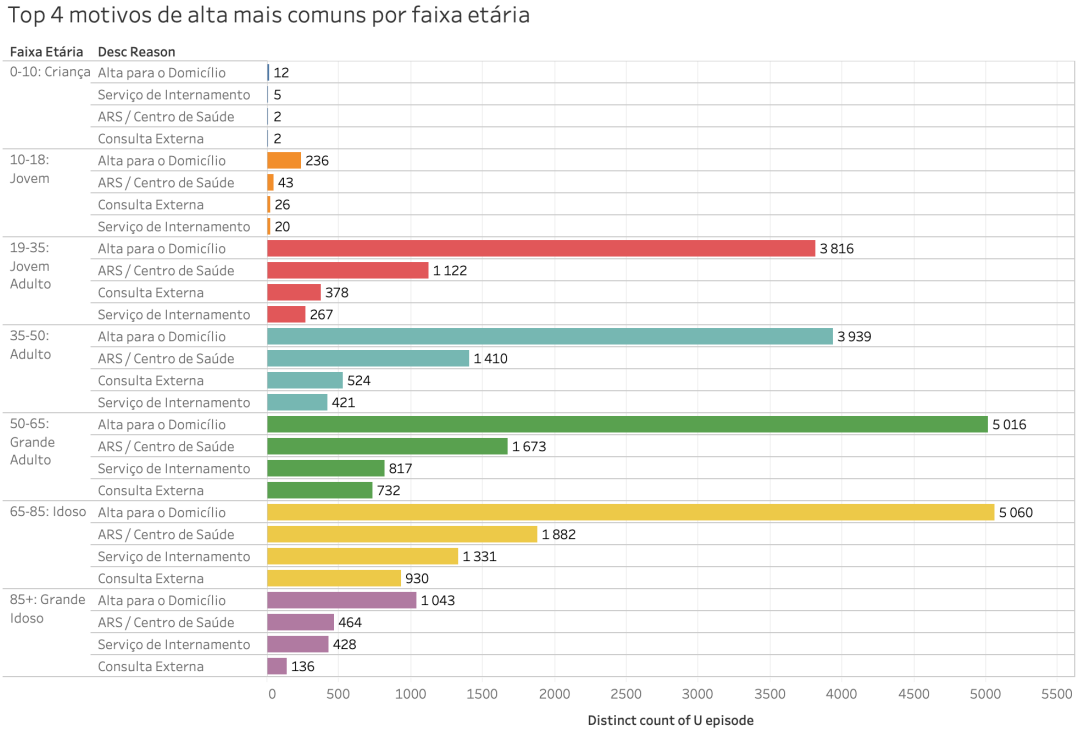
No quinto e último gráfico, podemos verificar a medicação mais prescrita por tipo de destino, sendo que o Paracetamol 1000mg é a medicação mais prescrita para múltiplos destinos desde ARS/Centro de Saúde a CE Endocrinologia ou até Cirurgia 1, entre outros. Sendo que o Paracetamol é um medicamento muito usado e para múltiplos fins, desde o alívio da dor à regulação da temperatura (febre), seria de esperar que este fosse o mais prescrito. O Ibuprofeno (anti-inflamatório), Tramadol + Paracetamol (alívio da dor) e Amoxicilina + Ácido Clavulânico (antibiótico) também são bastante prescritos para variados motivos de altas como Cirurgia 2 ou CE Ortopedia ou, ainda, CE ORL (respetivamente) não só por serem comuns, como também por serem bastante eficazes e toleráveis para a maioria dos indivíduos.

5.3 *Dashboard* sobre Diagnósticos e Altas









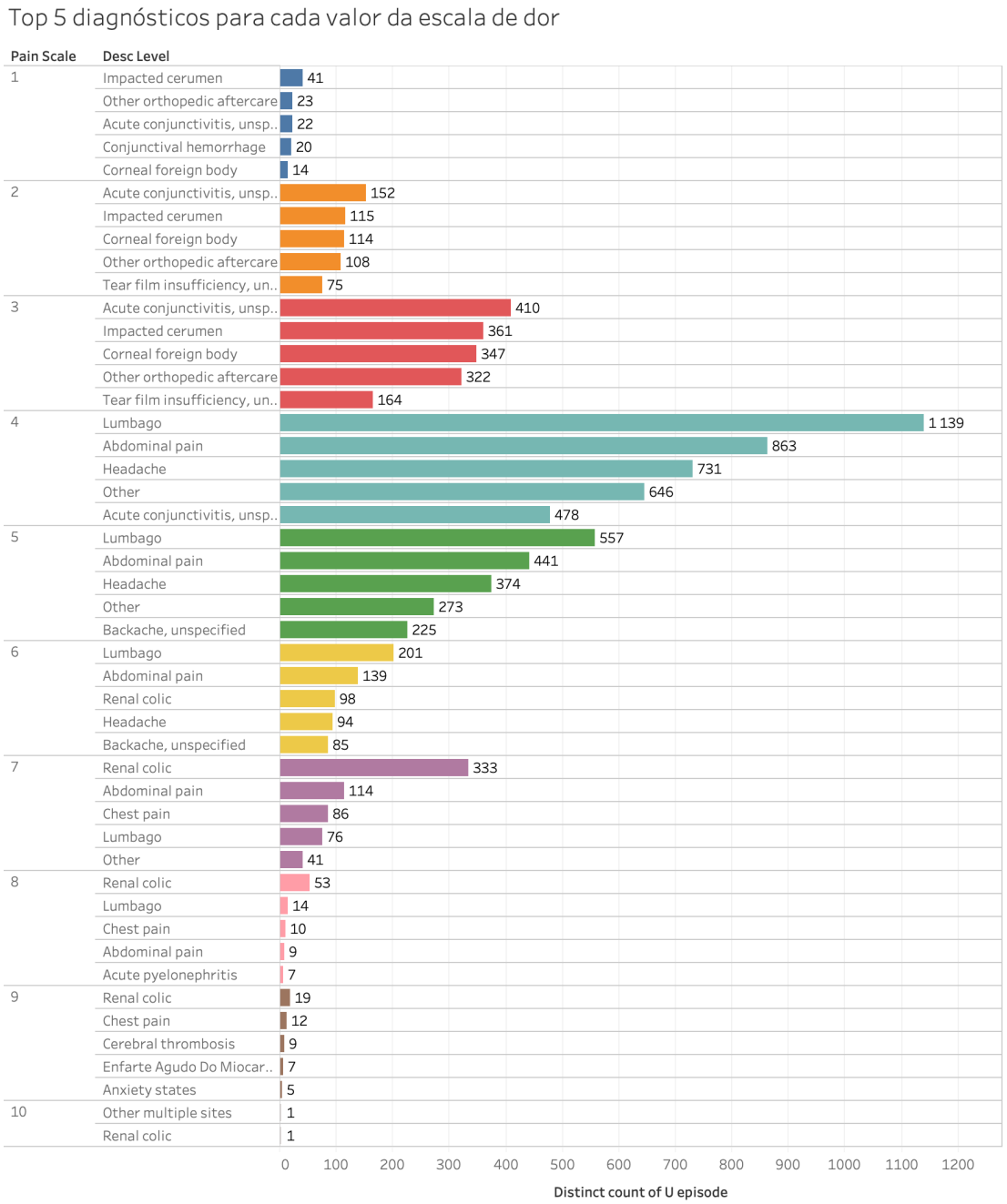


Figura 5.3: *Dashboard* Diagnósticos e Altas

Este *dashboard* representa uma análise sobre os dados relativos aos diagnósticos e aos registos de alta do serviço de urgência, tendo como objetivo geral refletir acerca da eficiência e produtividade do mesmo. Com isto é possível perceber, por exemplo, em que momentos a equipa terá de ser reforçada para conseguir agilizar os registos de alta.

No primeiro gráfico, podemos analisar a evolução do valor dos registos de alta por hora. Às 12 horas verifica-se o maior valor, seguido das 16, 19 e 17 horas, ou seja, antes da hora de almoço e durante o período da tarde, como seria de esperar. Podemos, ainda, verificar que, na madrugada, se observam os menores valores de registos, começando a aumentar a partir do período da manhã (9 horas). O facto de as 22, 23 e 00 horas também possuírem valores algo elevados pode dever-se, por exemplo, ao facto de os médicos que cumprem turnos de 24

horas darem alta aos utentes com condições para tal antes de iniciarem o período de pausa para descanso noturno.

No segundo gráfico, podemos observar o tempo médio que um utente passa no serviço de urgência (em horas) relacionado com a causa externa que o levou a recorrer ao mesmo. Tal como seria de esperar, casos menos sérios, como Queimaduras Solares ou Acidentes Domésticos, necessitam de menos tempo para serem atendidos e tratados quando comparados com casos mais sérios como Gravidez, Queda ou Intoxicação que, por norma, necessitam de mais cuidados e monitorização mais prolongada.

No terceiro gráfico, podemos verificar o *Top 5* de diagnósticos do nível final (mais específico) da hierarquia `ICD9`. O diagnóstico mais comum é o de *lumbago*, isto é, Lombalgia, seguido de dor Abdominal (*abdominal pain*), dor de Cabeça (*headache*), Conjutivite aguda (*acute conjunctivitis*) e Outros (*other*).

No quarto gráfico, podemos observar quais os quatro destinos de alta mais comuns por faixa etária. Para todas elas, o motivo mais comum é o de Alta para o Domicílio, isto é, os pacientes podem ir para a sua residência continuar (ou não) o tratamento, mediante a necessidade. Excetuando a faixa etária de Criança (`0-10`), todas as outras têm como segundo destino o de ARS/Centro de Saúde, ou seja, são referenciados para o seu centro de saúde e podem fazer o restante seguimento necessário com o seu médico de família. Já no caso da faixa referida, o destino anterior é o terceiro e o segundo é o de Serviço de Internamento, isto é, ser admitido e internado no hospital, numa ala específica, para ser acompanhado. Este último destino referido é semelhante para as restantes faixas etárias, contudo encontra-se na posição três (para as faixas Grande Adulto, Idoso e Grande Idoso, pois, por norma, com o aumento da idade, o sistema imunitário tende a ficar mais débil, pelo que, é mais difícil controlar certas patologias, sem o devido acompanhamento especializado) ou quatro (para as faixas Jovem, Jovem Adulto e Adulto). Existe, ainda, um último destino, o de Consulta Externa, para patologias com alguma granularidade extra e que necessitam de acompanhamento por parte de médicos especialistas naquela área específica.

No quinto e último gráfico, podemos constatar o *Top 5* de diagnósticos para cada valor da escala de dor (`1` a `10`). O valor mais comum desta escala é o `4`, cujo diagnóstico dominante é o de Lombalgia, seguido de dor Abdominal e dor de Cabeça (corroborando, assim, os dados evidenciados no terceiro gráfico). Para valores mais baixos que `4`, os diagnósticos mais comuns são os de Bloqueio do canal auditivo por Excesso de Cera (*impacted cerumen*, `1`) e Conjutivite aguda (*acute conjunctivitis*, `2` e `3`). Para valores mais altos, os diagnósticos mais comuns são os de Lombalgia (`5` e `6`), Cólica Renal (*renal colic*, `7`, `8`, `9` e `10`) e Infeção Respiratória em Múltiplos Sítios do pulmão (*other multiple sites*, `10`)[4]. Por experiência ou por investigação, todos estes diagnósticos estão, no nosso entender, bem atribuídos à escala de dor, principalmente o de Cólica Renal, conhecida pelas dores bastante fortes que provoca.

[4]O valor `10` tem o mesmo número de casos para os diagnósticos Cólica Renal e Infeção Respiratória em Múltiplos Sítios do pulmão, por isso aparece duas vezes.

Capítulo 6

Possível Aplicação e sua *Interface*

O desenvolvimento de uma aplicação direcionada aos profissionais do serviço de urgência de um hospital pode ter como principal foco a análise estatística de dados sobre o serviço, para uma melhor compreensão/gestão dos recursos alocados. Esta necessitaria, ainda, de ter em conta múltiplos fatores - a existência de um mecanismo de autenticação (para apenas profissionais autorizados terem acesso aos dados e, mesmo dentro dessa categoria, uns terem acesso a mais dados que outros), a capacidade de *update/refresh* (para os dados estarem sempre atualizados), a existência de vários menus (mediante a granularidade da informação procurada e para melhor separação dos departamentos), a disponibilização de contactos do serviço de informática ou um menu de ajuda da própria aplicação (para reportar algum *bug* que possa ser detetado ou alguma dúvida que surja sobre os dados), entre muitas outras.
Estas informações tornam-se bastante úteis, pois, num cenário em que existam múltiplas entradas com os mesmos sintomas, pode significar que estas contraíram a mesma doença (por exemplo, uma intoxicação alimentar, se os pacientes fizeram alguma refeição no mesmo restaurante, apesar de não se conhecerem), alertando um profissional que analise estes dados.
De seguida, mostramos alguns *mockups* ilustrativos do que seria expectável que a *interface* desta aplicação tivesse.

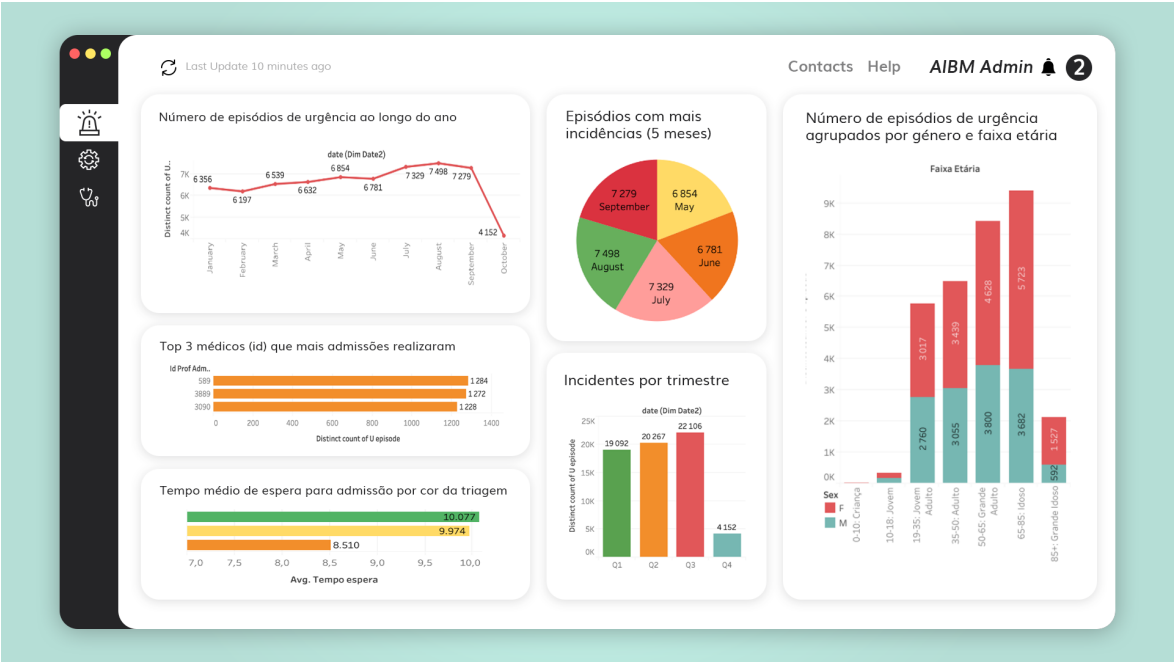


Figura 6.1: *Interface* de uma possível aplicação (menu 1)

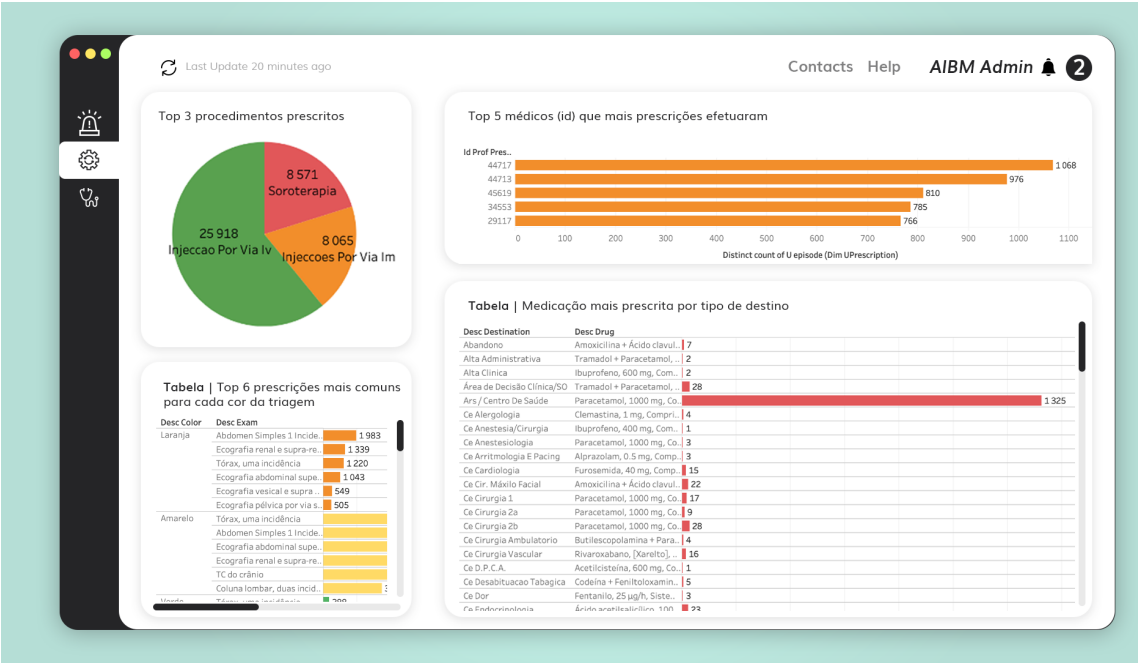


Figura 6.2: *Interface* de uma possível aplicação (menu 2)

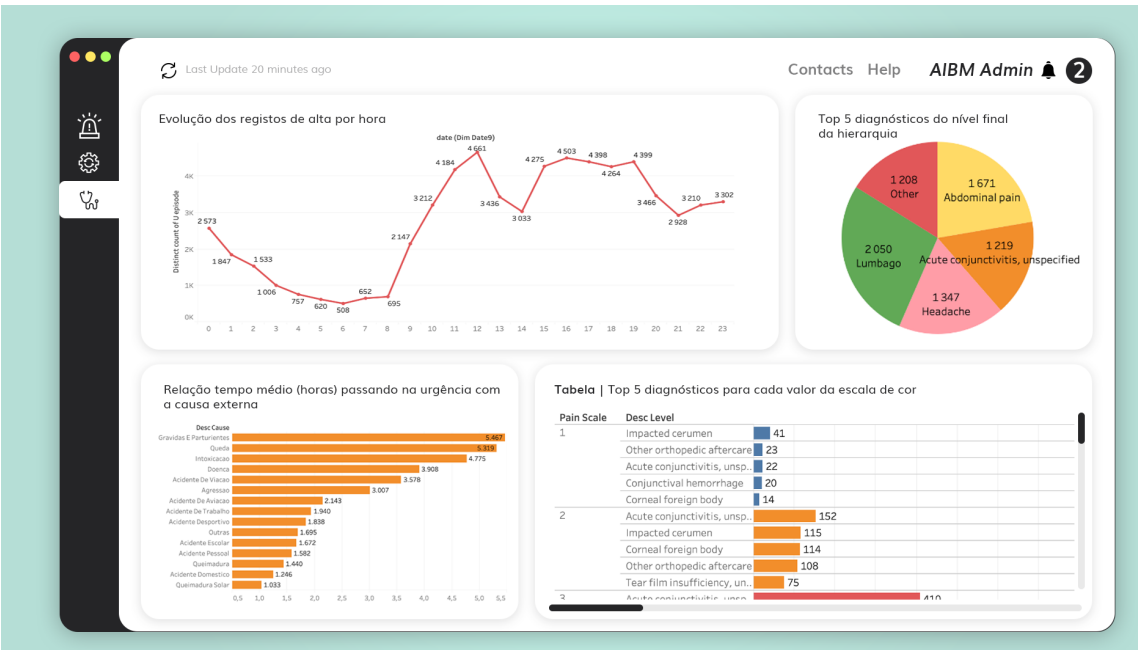


Figura 6.3: *Interface* de uma possível aplicação (menu 3)

É de ressalvar que, por questões práticas, esta aplicação deverá poder ser utilizada tanto num computador como num *smartphone* ou *tablet*.

Capítulo 7

Conclusões e Perspetivas Futuras

Durante a realização deste projeto, tivemos a oportunidade de consolidar e desenvolver os conhecimentos abordados nesta unidade curricular como, por exemplo, a necessidade de incluir dados de várias fontes (neste caso, vários ficheiros *.csv*) no processo `ETL` (que nos permitiu adquirir alguma experiência nos problemas que nos poderão ser apresentados no mundo "real"), para efetuar um povoamento mais completo do `Data Warehouse`. Pudemos, ainda, aprimorar conhecimentos com a construção do sistema de `Business Inteligence`, com vista à criação de indicadores relevantes que servirão de apoio na tomada de decisões clínicas. Adicionalmente, foi possível obter experiência com várias das ferramentas abordadas, como `MySQL`, `Talend` ou `Tableau Desktop`, que desempenham funções essenciais no desenvolvimentos dos sistemas de `Data Warehouse` (desde o desenvolvimento do modelo até ao seu povoamento) e de `Business Inteligence` (na criação de indicadores clínicos).
Para além destas ferramentas, optámos por utilizar, também, `Regex` e `Python`, para resolver os mais variados problemas relacionados com a formatação dos dados do *dataset*, poupando algum tempo precioso (que seria "gasto" na tentativa de execução do mesmo tratamento com as ferramentas anteriormente referidas) para o restante desenvolvimento do projeto.

O `Data Warehouse` foi estruturado de forma a permitir um acesso simples/organizado e a manutenção de dados consistentes, sem sobrecarregar nenhuma tabela, mas nunca perdendo de vista a possibilidade de evolução do modelo. Por exemplo, para podermos ter mais informação sobre pacientes (como nome ou número de telemóvel), criámos a tabela `Dim_Entry_Patient` para agrupar a informação de um dado paciente. Esta metologia era possível de aplicar a outros indivíduos, como os profissionais de saúde, contudo teria de existir uma diferenciação entre médicos, enfermeiros, auxiliares, entre outros. Optámos por colocar o género como *tiny int* para, futuramente, poder ser criada uma tabela para permitir a admissão de pacientes com géneros diferentes de masculino e feminino.

Em suma, considerámos que o trabalho produzido cumpre todos os requisitos, resultando num sistema capaz de armazenar toda a informação disponibilizada sem criar muita redundância, bem como um sistema de `Business Inteligence` com indicadores relevantes, baseados na informação armazenada no `Data Warehouse`, tanto para auxiliar na decisão clínica, como na organização das equipas médicas, reposição de *stock*, entre outros.
Este tipo de análise tem vindo a tornar-se cada vez mais importante, especialmente neste último ano, sendo que vários hospitais, por todo o mundo, entraram em estado de rutura, o que afeta não só a qualidade dos cuidados prestados aos pacientes, como também o estado mental dos profissionais que trabalham, diariamente, em esforço. Assim, com este tipo de análise do sistema, é possível tomar decisões melhor fundamentadas, garantindo uma melhor preparação por parte das instituições.

Apêndice A

Estrutura do modelo físico

```
-- -----------------------------------------------------
-- Schema urgency
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `urgency` DEFAULT CHARACTER SET utf8 ;
USE `urgency` ;
```

Figura A.1: *Header* inicial

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Date`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Date` (
  `id_date` INT NOT NULL,
  `date` DATETIME NULL DEFAULT NULL,
  PRIMARY KEY (`id_date`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.2: Modelo físico da tabela `Dim_Date`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Admition`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Admition` (
  `id_admition` INT NOT NULL AUTO_INCREMENT,
  `id_prof_admition` INT NULL DEFAULT NULL,
  `date_admition` INT NOT NULL,
  PRIMARY KEY (`id_admition`),
  INDEX `fk_Dim_Admition_Dim_Date1_idx` (`date_admition` ASC) VISIBLE,
  CONSTRAINT `fk_Dim_Admition_Dim_Date1`
    FOREIGN KEY (`date_admition`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8;
```

Figura A.3: Modelo físico da tabela `Dim_Admition`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Cause`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Cause` (
  `id_cause` INT NOT NULL,
  `desc_cause` VARCHAR(100) NULL DEFAULT NULL,
  PRIMARY KEY (`id_cause`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `urgency`.`Dim_Color`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Color` (
  `id_color` INT NOT NULL,
  `desc_color` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`id_color`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.4: Modelo físico das tabelas `Dim_Cause` e `Dim_Color`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Destination`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Destination` (
  `id_destination` INT NOT NULL,
  `desc_destination` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`id_destination`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.5: Modelo físico da tabela `Dim_Destination`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Hierarchy_Diagnosis`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Hierarchy_Diagnosis` (
  `id_level` VARCHAR(10) NOT NULL,
  `desc_level` VARCHAR(250) NULL DEFAULT NULL,
  `id_prev_level` VARCHAR(10) NULL DEFAULT NULL,
  PRIMARY KEY (`id_level`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.6: Modelo físico da tabela `Dim_Hierarchy_Diagnosis`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Diagnosis`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Diagnosis` (
  `id_diagnosis` INT NOT NULL AUTO_INCREMENT,
  `note_diagnosis` TEXT NULL DEFAULT NULL,
  `date_diagnosis` INT NOT NULL,
  `id_prof_diagnosis` INT NULL DEFAULT NULL,
  `id_level` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`id_diagnosis`),
  INDEX `fk_Diagnosis_Date1_idx` (`date_diagnosis` ASC) VISIBLE,
  INDEX `fk_Diagnosis_Hierarchy_diagnosis1_idx` (`id_level` ASC) VISIBLE,
  CONSTRAINT `fk_Diagnosis_Date1`
    FOREIGN KEY (`date_diagnosis`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`),
  CONSTRAINT `fk_Diagnosis_Hierarchy_diagnosis1`
    FOREIGN KEY (`id_level`)
    REFERENCES `urgency`.`Dim_Hierarchy_Diagnosis` (`id_level`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8;
```

Figura A.7: Modelo físico da tabela `Dim_Diagnosis`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Discharge`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Discharge` (
  `id_discharge` INT NOT NULL AUTO_INCREMENT,
  `id_prof_discharge` INT NULL DEFAULT NULL,
  `date_discharge` INT NOT NULL,
  PRIMARY KEY (`id_discharge`),
  INDEX `fk_Discharge_Date1_idx` (`date_discharge` ASC) VISIBLE,
  CONSTRAINT `fk_Discharge_Date1`
    FOREIGN KEY (`date_discharge`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `urgency`.`Dim_District`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_District` (
  `id_district` INT NOT NULL AUTO_INCREMENT,
  `district` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`id_district`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8;
```

Figura A.8: Modelo físico das tabelas `Dim_Discharge` e `Dim_District`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Drug`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Drug` (
  `id_drug` INT NOT NULL AUTO_INCREMENT,
  `cod_drug` INT NULL DEFAULT NULL,
  `pvp` INT NULL DEFAULT NULL,
  `comparticipation` INT NULL DEFAULT NULL,
  `desc_drug` VARCHAR(300) NULL DEFAULT NULL,
  PRIMARY KEY (`id_drug`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8;
```

Figura A.9: Modelo físico da tabela `Dim_Drug`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Entry_Patient`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Entry_Patient` (
  `U_episode` INT NOT NULL,
  `sex` TINYINT NULL DEFAULT NULL,
  `date_birthday` INT NOT NULL,
  PRIMARY KEY (`U_episode`),
  INDEX `fk_Dim_Patient_Dim_Date1_idx` (`date_birthday` ASC) VISIBLE,
  CONSTRAINT `fk_Dim_Patient_Dim_Date1`
    FOREIGN KEY (`date_birthday`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.10: Modelo físico da tabela `Dim_Entry_Patient`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Exams`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Exams` (
  `U_episode` INT NOT NULL,
  `id_Exam` VARCHAR(60) NULL DEFAULT NULL,
  `desc_exam` VARCHAR(150) NULL DEFAULT NULL,
  INDEX `fk_Dim_Exams_Dim_Patient1_idx` (`U_episode` ASC) VISIBLE,
  CONSTRAINT `fk_Dim_Exams_Dim_Patient1`
    FOREIGN KEY (`U_episode`)
    REFERENCES `urgency`.`Dim_Entry_Patient` (`U_episode`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.11: Modelo físico da tabela `Dim_Exams`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Intervention`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Intervention` (
  `id_intervention` INT NOT NULL,
  `desc_intervention` VARCHAR(200) NULL DEFAULT NULL,
  PRIMARY KEY (`id_intervention`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.12: Modelo físico da tabela Dim_Intervention

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Notes`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Notes` (
  `id_notes` INT NOT NULL AUTO_INCREMENT,
  `note` VARCHAR(550) NULL DEFAULT NULL,
  `note_cancel` VARCHAR(550) NULL DEFAULT NULL,
  `id_prof_cancel` INT NULL DEFAULT NULL,
  PRIMARY KEY (`id_notes`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `urgency`.`Dim_Posology`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Posology` (
  `id_posology` INT NOT NULL AUTO_INCREMENT,
  `desc_posology` VARCHAR(400) NULL DEFAULT NULL,
  PRIMARY KEY (`id_posology`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `urgency`.`Dim_Reason`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Reason` (
  `id_reason` INT NOT NULL,
  `desc_reason` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`id_reason`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.13: Modelo físico das tabelas Dim_Notes, Dim_Posology e Dim_Reason

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_Triage`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_Triage` (
  `id_triage` INT NOT NULL AUTO_INCREMENT,
  `id_prof_triage` INT NULL DEFAULT NULL,
  `pain_scale` INT NULL DEFAULT NULL,
  `id_color` INT NOT NULL,
  `date_triage` INT NOT NULL,
  PRIMARY KEY (`id_triage`),
  INDEX `fk_Triage_Color1_idx` (`id_color` ASC) VISIBLE,
  INDEX `fk_Dim_Triage_Dim_Date1_idx` (`date_triage` ASC) VISIBLE,
  CONSTRAINT `fk_Dim_Triage_Dim_Date1`
    FOREIGN KEY (`date_triage`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`),
  CONSTRAINT `fk_Triage_Color1`
    FOREIGN KEY (`id_color`)
    REFERENCES `urgency`.`Dim_Color` (`id_color`))
ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARACTER SET = utf8;
```

Figura A.14: Modelo físico da tabela `Dim_Triage`

```
-- -----------------------------------------------------
-- Table `urgency`.`Dim_UPrescription`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Dim_UPrescription` (
  `U_episode` INT NOT NULL,
  `cod_prescription` INT NOT NULL,
  `id_prof_prescription` INT NULL DEFAULT NULL,
  `date_urgency_prescription` INT NOT NULL,
  INDEX `fk_U_prescription_Date1_idx` (`date_urgency_prescription` ASC) VISIBLE,
  INDEX `fk_Dim_UPrescription_Dim_Patient1_idx` (`U_episode` ASC) VISIBLE,
  PRIMARY KEY (`cod_prescription`),
  CONSTRAINT `fk_Dim_UPrescription_Dim_Patient1`
    FOREIGN KEY (`U_episode`)
    REFERENCES `urgency`.`Dim_Entry_Patient` (`U_episode`),
  CONSTRAINT `fk_U_prescription_Date1`
    FOREIGN KEY (`date_urgency_prescription`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.15: Modelo físico da tabela `Dim_UPrescription`

```
-- -----------------------------------------------------
-- Table `urgency`.`Prescription_Has_Drug`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Prescription_Has_Drug` (
  `cod_prescription` INT NOT NULL,
  `id_drug` INT NOT NULL,
  `quantity` INT NULL,
  `id_posology` INT NOT NULL,
  PRIMARY KEY (`cod_prescription`, `id_drug`),
  INDEX `fk_Dim_UPrescription_has_Dim_Drug_Dim_Drug1_idx` (`id_drug` ASC) VISIBLE,
  INDEX `fk_Dim_UPrescription_has_Dim_Drug_Dim_UPrescription1_idx`
  (`cod_prescription` ASC) VISIBLE,
  INDEX `fk_Prescription_Has_Drug_Dim_Posology1_idx` (`id_posology` ASC) VISIBLE,
  CONSTRAINT `fk_Dim_UPrescription_has_Dim_Drug_Dim_UPrescription1`
    FOREIGN KEY (`cod_prescription`)
    REFERENCES `urgency`.`Dim_UPrescription` (`cod_prescription`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Dim_UPrescription_has_Dim_Drug_Dim_Drug1`
    FOREIGN KEY (`id_drug`)
    REFERENCES `urgency`.`Dim_Drug` (`id_drug`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Prescription_Has_Drug_Dim_Posology1`
    FOREIGN KEY (`id_posology`)
    REFERENCES `urgency`.`Dim_Posology` (`id_posology`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.16: Modelo físico da tabela `Prescription_Has_Drug`

```
-- -----------------------------------------------------
-- Table `urgency`.`Fact_UEpisodes`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Fact_UEpisodes` (
  `U_episode` INT NOT NULL,
  `id_district` INT NOT NULL,
  `id_cause` INT NOT NULL,
  `id_triage` INT NOT NULL,
  `id_diagnosis` INT NOT NULL,
  `id_destination` INT NOT NULL,
  `id_discharge` INT NOT NULL,
  `id_admition` INT NOT NULL,
  `id_reason` INT NOT NULL,
  PRIMARY KEY (`U_episode`),
  INDEX `fk_U_episodes_District1_idx` (`id_district` ASC) VISIBLE,
  INDEX `fk_U_episodes_Cause1_idx` (`id_cause` ASC) VISIBLE,
  INDEX `fk_U_episodes_Triage1_idx` (`id_triage` ASC) VISIBLE,
  INDEX `fk_U_episodes_Diagnosis1_idx` (`id_diagnosis` ASC) VISIBLE,
  INDEX `fk_U_episodes_Destination1_idx` (`id_destination` ASC) VISIBLE,
  INDEX `fk_U_episodes_Discharge1_idx` (`id_discharge` ASC) VISIBLE,
  INDEX `fk_U_episodes_Dim_Admition1_idx` (`id_admition` ASC) VISIBLE,
  INDEX `fk_U_episodes_Reason1_idx` (`id_reason` ASC) VISIBLE,
  INDEX `fk_Fact_UEpisodes_Dim_Patient1_idx` (`U_episode` ASC) VISIBLE,
  CONSTRAINT `fk_Fact_UEpisodes_Dim_Patient1`
    FOREIGN KEY (`U_episode`)
    REFERENCES `urgency`.`Dim_Entry_Patient` (`U_episode`),
  CONSTRAINT `fk_U_episodes_Cause1`
    FOREIGN KEY (`id_cause`)
    REFERENCES `urgency`.`Dim_Cause` (`id_cause`),
  CONSTRAINT `fk_U_episodes_Destination1`
    FOREIGN KEY (`id_destination`)
    REFERENCES `urgency`.`Dim_Destination` (`id_destination`),
  CONSTRAINT `fk_U_episodes_Diagnosis1`
    FOREIGN KEY (`id_diagnosis`)
    REFERENCES `urgency`.`Dim_Diagnosis` (`id_diagnosis`),
  CONSTRAINT `fk_U_episodes_Dim_Admition1`
    FOREIGN KEY (`id_admition`)
    REFERENCES `urgency`.`Dim_Admition` (`id_admition`),
  CONSTRAINT `fk_U_episodes_Discharge1`
    FOREIGN KEY (`id_discharge`)
    REFERENCES `urgency`.`Dim_Discharge` (`id_discharge`),
  CONSTRAINT `fk_U_episodes_District1`
    FOREIGN KEY (`id_district`)
    REFERENCES `urgency`.`Dim_District` (`id_district`),
  CONSTRAINT `fk_U_episodes_Reason1`
    FOREIGN KEY (`id_reason`)
    REFERENCES `urgency`.`Dim_Reason` (`id_reason`),
  CONSTRAINT `fk_U_episodes_Triage1`
    FOREIGN KEY (`id_triage`)
    REFERENCES `urgency`.`Dim_Triage` (`id_triage`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.17: Modelo físico da tabela `Fact_UEpisodes`

```
-- -----------------------------------------------------
-- Table `urgency`.`Fact_UProcedure`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `urgency`.`Fact_UProcedure` (
  `id_procedure` INT NOT NULL AUTO_INCREMENT,
  `U_episode` INT NOT NULL,
  `id_professional` INT NULL DEFAULT NULL,
  `date_begin` INT NOT NULL,
  `id_notes` INT NOT NULL,
  `date_cancel` INT NOT NULL,
  `id_intervention` INT NOT NULL,
  `date_clinic_prescription` INT NOT NULL,
  `id_prescription` INT NULL DEFAULT NULL,
  PRIMARY KEY (`id_procedure`),
  INDEX `fk_U_Procedure_Date1_idx` (`date_begin` ASC) VISIBLE,
  INDEX `fk_U_Procedure_Notes1_idx` (`id_notes` ASC) VISIBLE,
  INDEX `fk_U_Procedure_Date2_idx` (`date_cancel` ASC) VISIBLE,
  INDEX `fk_U_Procedure_intervention1_idx` (`id_intervention` ASC) VISIBLE,
  INDEX `fk_Fact_UProcedure_Dim_Date1_idx` (`date_clinic_prescription` ASC) VISIBLE,
  INDEX `fk_Dim_UProcedure_Dim_Patient1_idx` (`U_episode` ASC) VISIBLE,
  CONSTRAINT `fk_Dim_UProcedure_Dim_Patient1`
    FOREIGN KEY (`U_episode`)
    REFERENCES `urgency`.`Dim_Entry_Patient` (`U_episode`),
  CONSTRAINT `fk_Fact_UProcedure_Dim_Date1`
    FOREIGN KEY (`date_clinic_prescription`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`),
  CONSTRAINT `fk_U_Procedure_Date1`
    FOREIGN KEY (`date_begin`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`),
  CONSTRAINT `fk_U_Procedure_Date2`
    FOREIGN KEY (`date_cancel`)
    REFERENCES `urgency`.`Dim_Date` (`id_date`),
  CONSTRAINT `fk_U_Procedure_intervention1`
    FOREIGN KEY (`id_intervention`)
    REFERENCES `urgency`.`Dim_Intervention` (`id_intervention`),
  CONSTRAINT `fk_U_Procedure_Notes1`
    FOREIGN KEY (`id_notes`)
    REFERENCES `urgency`.`Dim_Notes` (`id_notes`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura A.18: Modelo físico da tabela `Fact_UProcedure`