Relatório trabalho final de banco de dados

Universidade Federal do Rio de Janeiro - UFRJ
Mateus Santos (115062262), Lucas Clemente (115159700), Rafaela Fernandes (118059739), Diogo Rocco Crivaro Nunes (115042725)
Matheus da Silva Oliveira (118178020)

Resumo: Coletamos os dados do HSL (Hospital Sírio Libanês), normalizamos e criamos uma infraestrutura de banco de dados a partir destes. Criamos uma aplicação do tipo cliente-servidor onde uma API consulta o banco de dados e oferece para a interface de usuário. Na interface há gráficos que ajudam a visualizar melhor os dados coletados.

1. Modelagem

Começamos a modelagem do Banco de Dados a partir de três arquivos .csv desnormalizados que podem ser acessados através do link que se encontra na área de referências. São eles:

- HSL_Desfechos_3
- HSL_Exames_3
- HSL_Pacientes_3

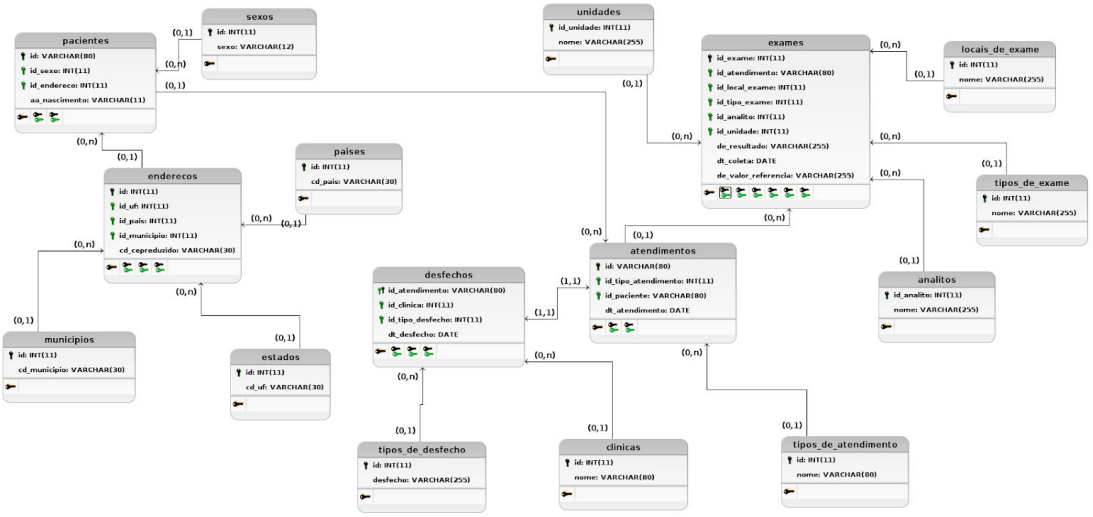
A descrição dos arquivos pode ser lida através do link que está no tópico de referências.

1.1 Normalização

Um dos primeiros arquivos analisados para a normalização foi o que mostrava os dados dos pacientes anonimizados. Percebemos que os dados dos campos IC_SEXO, CD_PAIS, CD_UF, CD_MUNICIPIO e CD_CEPREDUZIDO se repetiam com frequência e em alguns casos, havia poucos dados distintos, por exemplo em IC_SEXO, que apresentava apenas dois dados: “M” e “F”. Dessa forma, decidimos criar tabelas à parte para reduzir estas repetições e economizar espaço no banco. Além disso, em alguns casos, não foi possível normalizar perfeitamente pois haveria inconsistências. Foi o que aconteceu na tabela “enderecos” por exemplo, quando tentamos usar o campo CD_CEPREDUZIDO como chave estrangeira. Se isto fosse possível, não seria necessário usar como chaves estrangeiras os campos CD_UF e CD_MUNICIPIO, já que cada CEP já se relaciona com esses dados. Contudo, como no dataset os CEP 's não estavam completos, utilizamos a abordagem comentada acima.

Também percebemos problemas na tabela “exames”, pois checamos que no dataset que apresentava os dados destes, nos campos DE_ANALITO e DE_VALOR_REFERENCIA havia inconsistências. Em alguns momentos, no campo DE_ANALITO há dois analitos sendo observados, enquanto no campo DE_VALOR_REFERENCIA não é especificado o valor de referência para cada um deles. Dessa forma, a informação é muito restrita a quem inseriu os dados, e não para quem consome, com isso, apenas separamos em uma tabela à parte os analitos analisados, enquanto os valores de referência não foram trabalhados diretamente, ou seja, colocados em uma tabela separada.

Ao final de todo o processo de normalização, o trabalho final pode ser visto na imagem abaixo:



Como podemos notar, buscamos ao máximo a normalização do esquema. Infelizmente, isso não foi possível totalmente já que o dataset apresentava algumas inconsistências em sua lógica. Dessa forma, focamos em economizar espaço nos campos para que fosse possível a carga dos dados para o banco e no intuito de evitar o uso desnecessário de armazenamento em disco.

2. Queries

Com base no banco de dados resultante da nossa normalização, decidimos montar queries que pudessem explorar ao máximo os atendimentos e seus desfechos, além da ligação que cada paciente teria com estes. A seguir, mostraremos as queries e o objetivo que se espera alcançar com cada uma.

```
SELECT
TIP.desfecho as Desfecho,
  COUNT(*) as Quantidade
FROM
atendimentos AS ATD
INNER JOIN
desfechos AS DES on ATD.id = DES.id_atendimento
INNER JOIN
tipos_de_desfecho AS TIP on TIP.id =
DES.id_tipo_desfecho
GROUP BY
TIP.desfecho
```

```
SELECT
EST.cd_uf as Estado,

COUNT(*) as Quantidade_de_Atendimentos
FROM
 estados as EST
INNER JOIN
 enderecos as ENDE on ENDE.id_uf = EST.id
INNER JOIN
 pacientes as PAC on PAC.id_endereco = ENDE.id
INNER JOIN
 atendimentos as AT on AT.id_paciente = PAC.id
GROUP BY
 EST.cd_uf
```

A primeira query é de forma geral bem simples, onde teremos uma tabela resultando na quantidade que cada desfecho possuiu. Realizando um agrupamento pelos desfechos e um Count nas linhas, tivemos resultados

A segunda query buscava a quantidade de atendimentos por estado, na mesma linha que a query anterior, onde conseguimos analisar que a esmagadora maioria dos pacientes eram do estado de São Paulo. A segunda posição ficou com o Distrito Federal.

```
SELECT
        estado,
        COUNT(DISTINCT atendimentos.id) AS internações
FROM
         atendimentos
JOIN
        (SELECT pacientes.id AS id, cd_uf AS estado FROM pacientes
        JOIN enderecos ON enderecos.id = id_endereco
        JOIN estados ON estados.id = id_uf
        ) AS state_patient ON state_patient.id = id_paciente
WHERE
        atendimentos.id IN  (SELECT id FROM atendimentos
                        WHERE id_tipo_atendimento IN (SELECT id
                                                FROM tipos_de_atendimento
                                                WHERE nome LIKE '%Internado%')
                        )
GROUP BY
        estado
```

Essa query utiliza consultas aninhadas para retornar a quantidade de internações por estado, agrupando os resultados pelo estado onde os pacientes residem.

```
SELECT
        EST.cd_uf as Estado,
        COUNT(*) as Quantidade_de_Atendimentos
FROM            estados as EST
INNER JOIN      enderecos as ENDE on ENDE.id_uf = EST.id
INNER JOIN      pacientes as PAC on PAC.id_endereco = ENDE.id
```

```
INNER JOIN      atendimentos as AT on AT.id_paciente = PAC.id
INNER JOIN      desfechos as DES on DES.id_atendimento = AT.id
INNER JOIN      tipos_de_desfecho as TIP on TIP.id = DES.id_tipo_desfecho
GROUP BY        EST.cd_uf
```

Semelhante a anterior, porém mais simples, essa query retorna a quantidade de atendimentos por estado utilizando um conjunto de inner joins e agrupamento por UF.

```
SELECT
        dt_atendimento,
        COUNT(DISTINCT atendimentos.id) AS
internações
FROM
        atendimentos
WHERE
        id_tipo_atendimento IN (SELECT id
                                FROM
                            tipos_de_atendimento
                                WHERE nome LIKE
                            '%Internado%')
GROUP BY dt_atendimento
ORDER BY dt_atendimento ASC
```

```
SELECT
        dt_desfecho,
        COUNT(DISTINCT id_atendimento) AS
        obitos
FROM
        desfechos
WHERE
        id_tipo_desfecho IN (SELECT id
                              FROM
                            tipos_de_desfecho
                              WHERE desfecho
                            LIKE '%Óbito%')
GROUP BY dt_desfecho
ORDER BY dt_desfecho ASC
```

Novamente utilizamos sub-consultas para podermos recuperar dados que consideramos essenciais na análise, agora analisando duas séries temporais, sendo o primeiro o número de internações por data, e o segundo número de mortes por data.

```
SELECT
        CASE
            WHEN T2.Ano_Nascimento between 1930 and 1940 then 'Entre 80 e 90 anos'
            WHEN T2.Ano_Nascimento between 1941 and 1950 then 'Entre 70 e 80 anos'
            WHEN T2.Ano_Nascimento between 1951 and 1960 then 'Entre 60 e 70 anos'
            WHEN T2.Ano_Nascimento between 1961 and 1990 then 'Entre 30 e 60 anos'
            WHEN T2.Ano_Nascimento between 1991 and 2021 then 'Entre 0 e 30 anos'
        END as Faixa_Idade,
        SUM(Soma_Mortes) / Soma_Total as Taxa_Mortalidade
FROM
(SELECT
         Count(*) as Soma_Total
FROM
        pacientes as PAC
INNER JOIN
        atendimentos AS ATD on ATD.id_paciente = PAC.id
INNER JOIN
        desfechos AS DES on ATD.id = DES.id_atendimento
INNER JOIN
         tipos_de_desfecho AS TIP on TIP.id = DES.id_tipo_desfecho
WHERE
        TIP.desfecho like '%Óbito%') AS T
INNER JOIN
(SELECT
          PAC.aa_nascimento as Ano_Nascimento,
          Count(*) as Soma_Mortes
FROM
          pacientes as PAC
INNER JOIN
          atendimentos AS ATD on ATD.id_paciente = PAC.id
INNER JOIN
          desfechos AS DES on ATD.id = DES.id_atendimento
INNER JOIN
          tipos_de_desfecho AS TIP on TIP.id = DES.id_tipo_desfecho
```

```
WHERE
        TIP.desfecho like '%Óbito%'
GROUP BY
        Ano_Nascimento) AS T2 on 1=1
GROUP BY
        Faixa_Idade
```

Por último, encerramos com a query que mostra a taxa de mortalidade dado um agrupamento de idades, onde os resultados trazem que a maior taxa está em pacientes entre 80 e 90 anos, onde a mortalidade vai caindo conforme a idade vai diminuindo. Também percebemos uma taxa relativamente alta em pacientes que não identificaram a idade, como vemos abaixo, porém os resultados são inconclusivos para este caso.

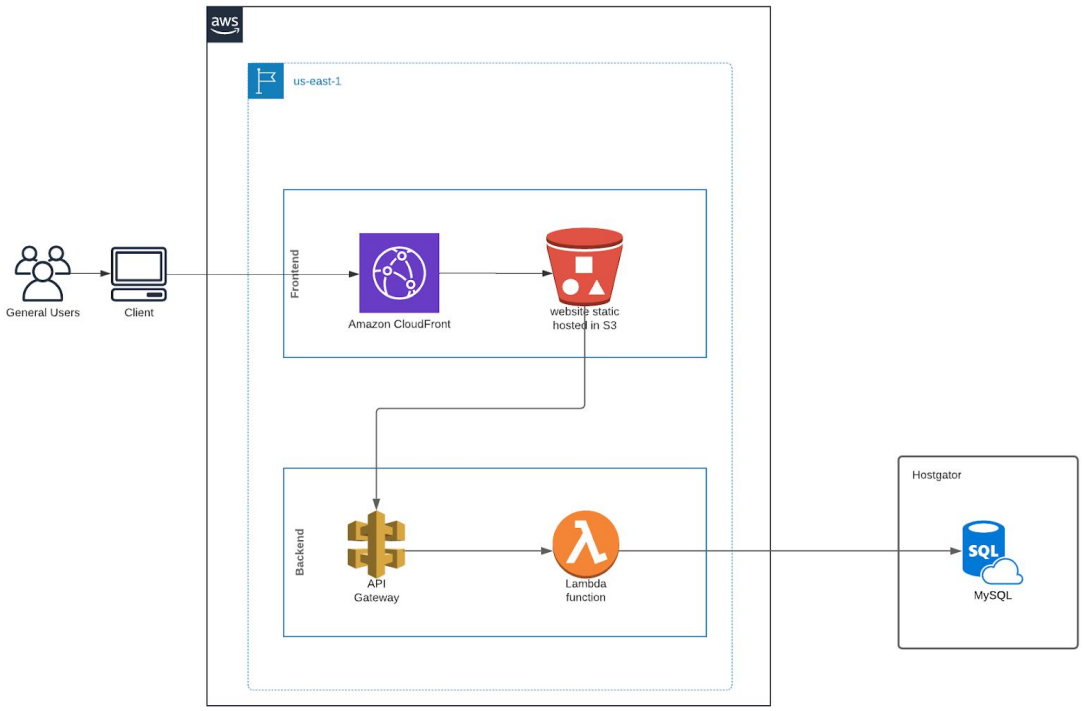
Faixa_Idade	Taxa_Mortalidade
Entre 30 e 60 anos	0.1176
Entre 60 e 70 anos	0.0588
Entre 70 e 80 anos	0.1176
Entre 80 e 90 anos	0.4706
Não informado	0.2353

A imagem da tabela foi retirada após uma consulta ao nosso banco de dados feita via *phpmyadmin*. Os resultados desta e das queries restantes poderá ser visto via *App Web*, como será explicado na próxima seção.

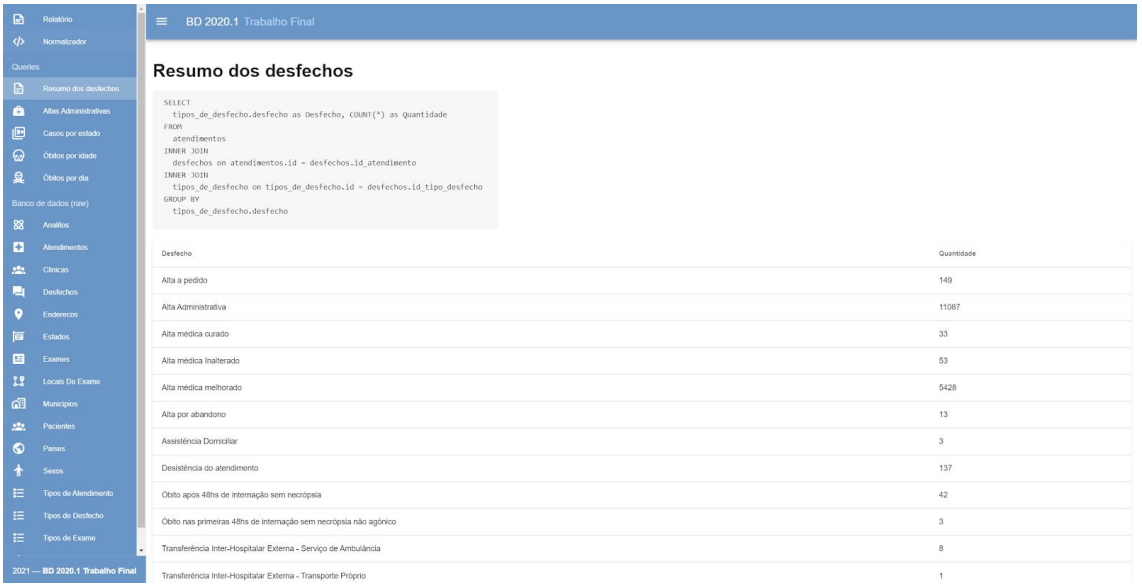
3. App

A aplicação usa arquitetura client-server, os respectivos códigos se encontram nas pastas front (client) e back (server). A interface se encontra disponível no seguinte link: http://bd-2020-1-trabalho-final.s3-website-us-east-1.amazonaws.com/

O backend hospeda o banco de dados e oferece uma API para o frontend, que por sua vez consome a API e disponibiliza uma interface de usuário. O backend foi escrito usando Python e o Serverless Framework, os serviços foram hospedados no AWS (Amazon Web Services) e no Hostgator. O frontend foi desenvolvido usando a linguagem Javascript e a framework VueJS. Abaixo há um diagrama da arquitetura da aplicação.



A interface é simples e possui uma barra de navegação lateral para facilitar o acesso à informação.



A navegação pelo site pode ser feita de forma simples e prática através da barra lateral, onde podemos acessar uma versão do relatório disponibilizada online, além do algoritmo de normalização utilizado para gerarmos o modelo.

Podemos também visualizar o retorno que as consultas da seção anterior nos trazem, tanto em formato de tabela como em formatos gráficos para melhor visualização e interpretação do usuário. Junto ao resultado, também é possível visualizar a query em questão em cada aba.
Por mim, também é possível visualizar um select simples de todas as tabelas resultantes do processo de normalização e que constituem o modelo final do nosso banco de dados, assim como os resultados de cada uma.

Referências

Hospital Sírio-Libanês (2020) “Dados COVID Hospital Sírio-Libanês” https://repositoriodatasharingfapesp.uspdigital.usp.br/handle/item/97, Junho