MVP - Engenharia de Dados
Aluno: Gustavo de Almeida Ferreira

1. Objetivo

O objetivo deste projeto é desenvolver a aplicação dos conhecimentos adquiridos no módulo de Engenharia de Dados da Pós-Graduação em Ciência de Dados da PUC-Rio. Para tal, será utilizado o dataset Healthcare, que contém informações referentes ao emprego de planos de saúde nos EUA. Suas informações e estrutura serão aprofundados nas próximas seções. Conforme as orientações deste MVP, dois aspectos principais do dataset serão analisados:

- Qualidade: será verificado se todas as tabelas possuem valores nulos. Nas tabelas que contêm valores numéricos, será analisado também se tais dados possuem outliers, ou seja, valores
- Questões de negócio: cinco questões de negócio serão respondidas pela análise dos dados do dataset:
    a. Quais são os estados onde há mais registros no dataset?
    b. Quais são as dez idades de pacientes que possuem mais registros no dataset?
    c. Quais são as dez doenças que possuem mais registros no dataset?
    d. Quais são as categorias de pagador que possui mais registros no dataset?
    e. Quanto foi cobrado de cada uma das categorias de pagador?

2. Detalhamento

A base de dados utilizado para o MVP do Módulo de Engenharia de Dados foi o Healthcare, disponível no Kaggle por meio do seguinte link: https://www.kaggle.com/datasets/tomaslui/healthcare-dataset/discussion/469010. Nele pode-se encontrar milhares de registros de pacientes que fizeram uso de planos de saúde nos estados unidos, suas cidades, estados, localidades onde os procedimentos foram realizados, entre outros. As tabelas foram hospedadas no SGBP pgAdmin, onde também foram executadas todas as queries e operações de carregamento, transformação e análise dos dados.

3. Coleta

A coleta de dados foi realizada pelo simples download dos dados por meio do hiperlink mencionado acima.

4. Modelagem

Baseado numa análise preliminar dos dados que compõem o modelo, foi escolhido um esquema estrela de modelagem de dados, vide a imagem abaixo. Nela estão retratadas todas as tabelas que compõem o modelo, bem como suas chaves primárias, explicitadas em vermelho.

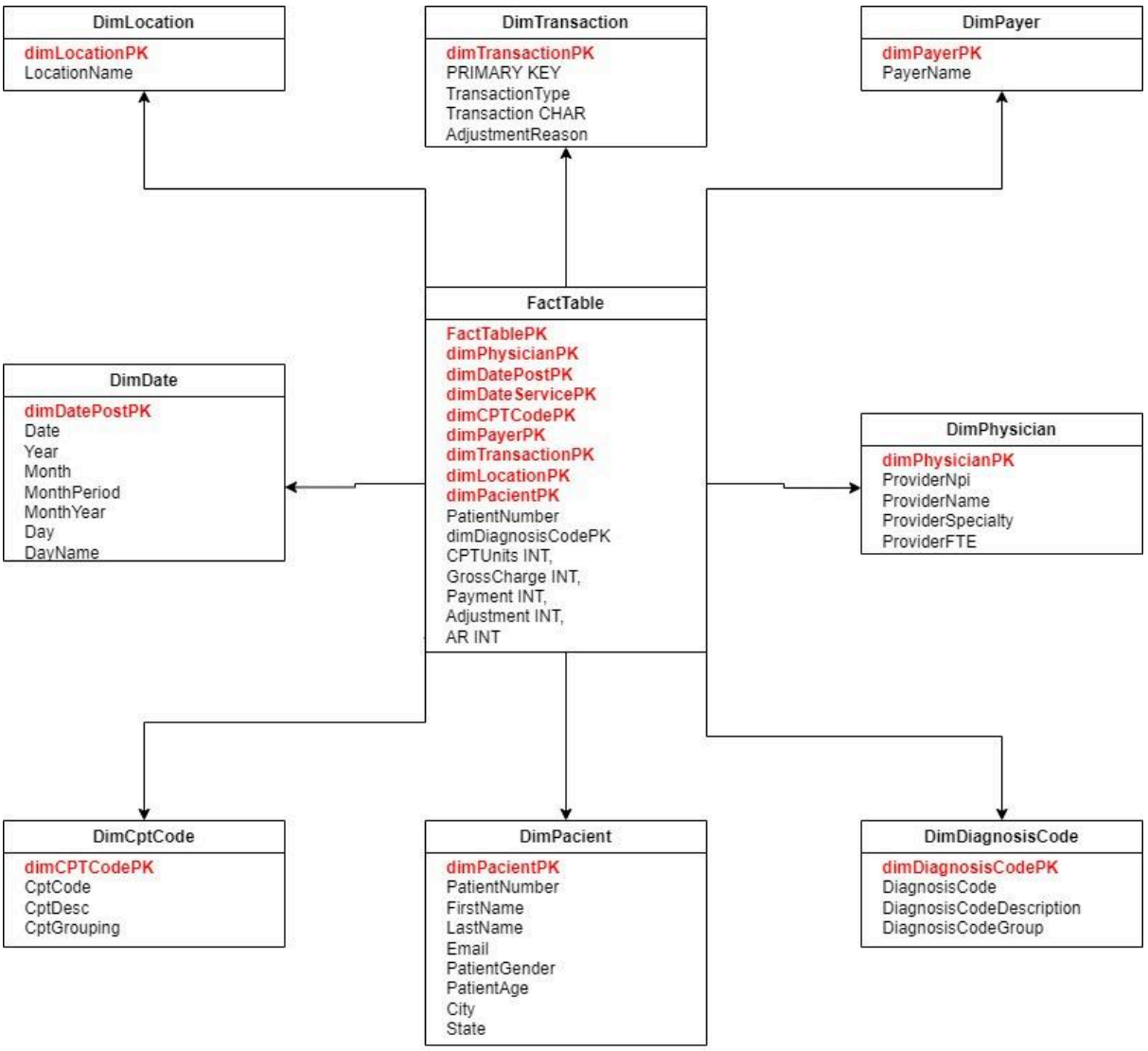

Figura 1 - Esquema dos dados usados

Abaixo consta um catálogo de dados que explica as informações de cada tabela:

Tabela	Coluna	Definição
FactTable	FactTablePK	Identificador exclusivo (chave primária) da tabela fato
	dimDateServicePK	Data formatada em texto que especifica o dia em que o paciente foi atendido pelo médico
	CPTUnits	O número de vezes que o Código CPT foi executado.
	GrossCharge	A cobrança bruta do Código CPT. Isso inclui o reajuste contratual
	Payment	Pagamento recebido do pagador e do paciente
	Adjustment	Qualquer valor da cobrança bruta que não serão cobrados
	AR	Contas a receber pendentes que ainda não foram cobradas ou baixadas.
dimPatient	dimPatientPK	Identificador exclusivo que une a tabela de pacientes à tabela

		fato
	PatientNumber	Identificador exclusivo do paciente. Muitas vezes isso é chamado de Número de Registro Médico (MRN, em inglês)
	FirstName	Nome do paciente
	LastName	Sobrenome do paciente
	Email	Endereço de e-mail dos pacientes
	PatientGender	Gênero/sexo do paciente
	PatientAge	Idade dos pacientes
	City	Cidade onde o paciente mora
	State	Estado onde o paciente mora
dimTransaction	dimTransactionPK	Identificador exclusivo que une a tabela de transações à tabela fato
	TransactionType	Esclarece se a transação foi uma cobrança, pagamento, reajuste, etc.
	Transaction	Descrições de transações específicas
	AdjustmentReason	Agrupa ajustes em grupos de motivos operacionais
dimPhysician	dimPhysicianPK	Identificador exclusivo que une a tabela de médicos à tabela fato
	ProviderNpi	Número único atribuído a cada médico após a conclusão da faculdade de medicina
	ProviderName	Sobrenome dos médicos
	ProviderSpecialty	Área em que o médico se especializou
	ProviderFTE	Tempo gasto pelo médico trabalhando semanalmente
dimPayer	dimPayerPK	Identificador único que une a tabela de pagadores à tabela fato
	PayerName	Categoria do pagador
dimLocation	dimLocationPK	Identificador exclusivo que une a tabela de localização à tabela fato
	LocationName	Campo que especifica onde o serviço foi prestado
dimDiagnosisCode	dimDiagnosisCodePK	Identificador exclusivo que une a tabela de pacientes à tabela fato
	DiagnosisCode	Código atribuído pelo médico para especificar o diagnóstico do paciente
	DiagnosisCodeDescription	Descrição para esclarecer o código de diagnóstico
	DiagnosisCodeGroup	Grupo de códigos de diagnóstico
dimCptCode	dimCPTCodePK	Identificador exclusivo que une a tabela CPT à tabela fato
	CptCode	Terminologia processual atual de cinco dígitos usada para atribuir serviços prestados ao paciente
	CptDesc	Descrição específica do CPT

	CptGrouping	Agrupa os códigos CPT em tipos de serviços
dimDate	dimDatePostPK	Data formatada em texto que especifica o dia em que a transação foi lançada no sistema de faturamento
	Date	Data em que o serviço foi prestado ou data de uma transação
	Year	Ano em que o serviço foi prestado
	Month	Mês em que o serviço foi prestado
	MonthPeriod	Data em que o serviço foi prestado ou data de uma transação
	MonthYear	Mês e ano em que um serviço foi prestado
	Day	Dia do mês em que o serviço foi prestado
	DayName	Dia da semana em que o serviço foi prestado

4.Carga

Primeiramente foram criadas dentro do SGBD as tabelas que irão receber os dados do modelo. O código de criação das tabelas podem ser visualizados nas imagens abaixo:

```
CREATE TABLE FactTable (
    FactTablePK VARCHAR PRIMARY KEY,
    dimPatientPK VARCHAR,
    dimPhysicianPK VARCHAR,
    dimDatePostPK VARCHAR,
    dimDateServicePK VARCHAR,
    dimCPTCodePK VARCHAR,
    dimPayerPK VARCHAR,
    dimTransactionPK VARCHAR,
    dimLocationPK VARCHAR,
    PatientNumber VARCHAR,
    dimDiagnosisCodePK VARCHAR,
    CPTUnits NUMERIC(10,2),
    GrossCharge NUMERIC(10,2),
    Payment NUMERIC(10,2),
    Adjustment NUMERIC(10,2),
    AR NUMERIC(10,2),
    FOREIGN KEY (dimPatientPK) REFERENCES dimPacient(dimpatientpk),
    FOREIGN KEY (dimPhysicianPK) REFERENCES dimPhysician(dimPhysicianPK),
    FOREIGN KEY (dimDatePostPK) REFERENCES dimDate(dimDatePostPK),
    FOREIGN KEY (dimCPTCodePK) REFERENCES dimCPTCode(dimCPTCodePK),
    FOREIGN KEY (dimPayerPK) REFERENCES dimPayer(dimPayerPK),
    FOREIGN KEY (dimTransactionPK) REFERENCES dimTransaction(dimTransactionPK),
    FOREIGN KEY (dimLocationPK) REFERENCES dimLocation(dimLocationPK),
    FOREIGN KEY (dimDiagnosisCodePK) REFERENCES dimDiagnosisCode(dimDiagnosisCodePK)
)
```

Figura 2 - Criação da tabela FactTable

```
CREATE TABLE dimDiagnosisCode (
    dimDiagnosisCodePK VARCHAR PRIMARY KEY,
    DiagnosisCode VARCHAR,
    DiagnosisCodeDescription VARCHAR,
    DiagnosisCodeGroup VARCHAR
)
```

Figura 3 - Criação da tabela dimDiagnosisCode

```
CREATE TABLE dimCptCode (
    dimCPTCodePK VARCHAR PRIMARY KEY,
    CptCode VARCHAR,
    CptDesc VARCHAR,
    CptGrouping VARCHAR
)
```

Figura 4 - Criação da tabela dimCptCode

```
CREATE TABLE Dimlocation (
    dimLocationPK VARCHAR PRIMARY KEY,
    LocationName VARCHAR
)
```

Figura 5 - Criação da tabela Dimlocation

```
CREATE TABLE Dimpacient (
    dimPatientPK VARCHAR PRIMARY KEY,
    PatientNumber VARCHAR,
    FirstName VARCHAR,
    LastName VARCHAR,
    Email VARCHAR,
    PatientGender VARCHAR,
    PatientAge VARCHAR,
    City VARCHAR,
    State VARCHAR
)
```

Figura 6 - Criação da tabela Dimpacient

```
CREATE TABLE Dimphysician (
    dimPhysicianPK VARCHAR PRIMARY KEY,
    ProviderNpi VARCHAR,
    ProviderName VARCHAR,
    ProviderSpecialty VARCHAR,
    ProviderFTE VARCHAR
)
```

Figura 7 - Criação da tabela Dimphysician

```
CREATE TABLE Dimpayer (
    dimPayerPK VARCHAR PRIMARY KEY,
    PayerName VARCHAR
)
```

Figura 8 - Criação da tabela Dimpayer

```
CREATE TABLE Dimtransaction (
    dimTransactionPK VARCHAR PRIMARY KEY,
    TransactionType VARCHAR,
    Transaction VARCHAR,
    AdjustmentReason VARCHAR
)
```

Figura 9 - Criação da tabela Dimtransaction

```
CREATE TABLE Dimdate (
    dimDatePostPK VARCHAR PRIMARY KEY,
    Date VARCHAR,
    Year VARCHAR,
    Month VARCHAR,
    MonthPeriod VARCHAR,
    MonthYear VARCHAR,
    Day VARCHAR,
    DayName VARCHAR
)
```

Figura 10 - Criação da tabela Dimdate

Depois de criadas as tabelas, foi necessário carregá-las com os dados baixados. Para isso, foi preciso clicar com o botão direito sobre o nome de cada tabela na barra lateral esquerda do SGBD e clicar no campo “Import/Export Data”:

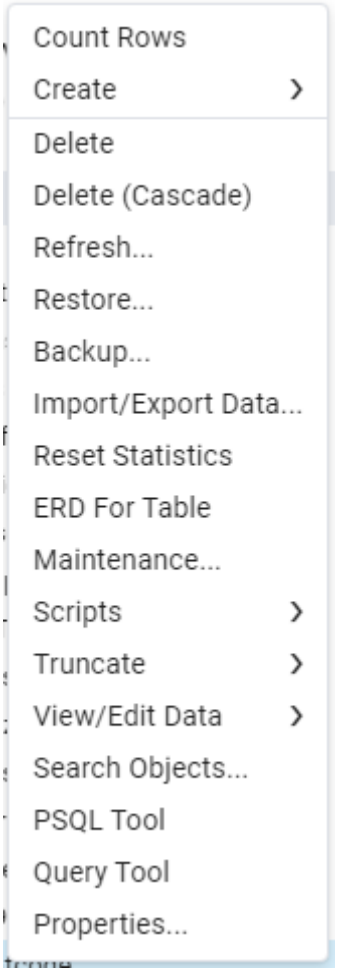

Figura 11 - Carga de dados

Em seguida, foi necessário selecionar o arquivo csv contendo cada uma das fontes de dados baixadas:

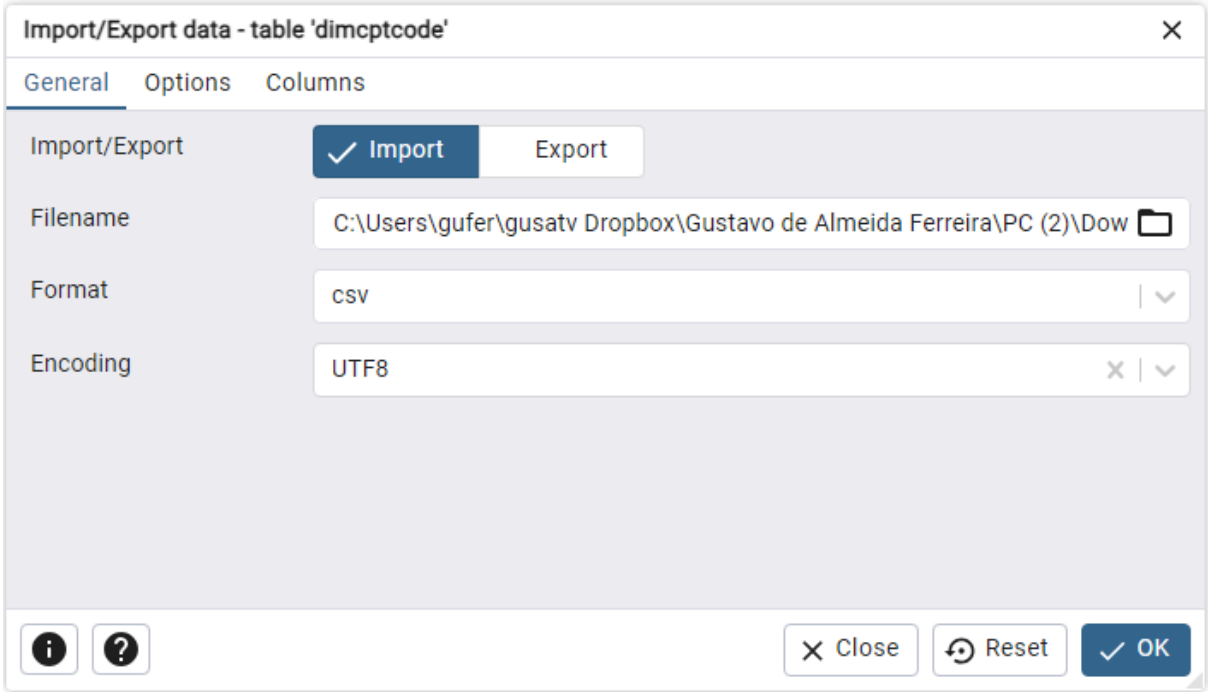

Figura 12 - Seleção de arquivos com os dados

As seguintes opções foram selecionadas para que os dados fossem carregados corretamente:

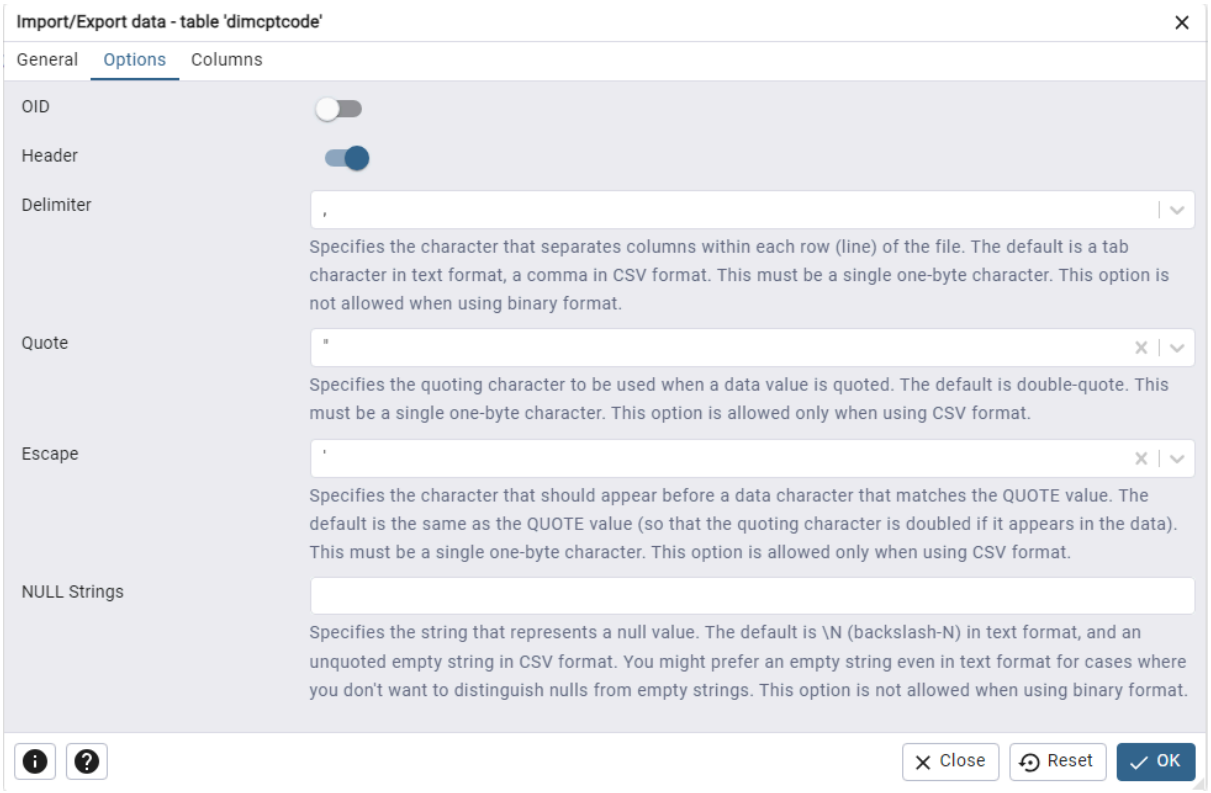

Figura 13 - Configurações de carga de dados

De modo geral, os dados carregados dispensam qualquer tipo de transformação pois, como será provado posteriormente, eles apresentam um alto grau de qualidade. No entanto, a tabela DimPacient apresenta um caso curioso: conforme observado no catálogo de dados apresentado na Seção 2, as colunas dimPatientPK e PacientNumber pretendem ser registros únicos. No entanto, analisando, por meio de uma análise mais profunda, percebe-se que há duplicidade de registros na coluna PacientNumber e nas demais colunas da tabela, de modo que os registros da coluna dimPatientPK, embora contenha registros únicos, faz parte de tuplas que representam exatamente a mesma informação. Por essa razão, para garantir maior rigor referencial da FactTable, optou-se por retirar a coluna dimPatientPK da tabela DimPatient e restringir a coluna PacientNumber aos seus registros únicos.

Por meio do código abaixo, a coluna dimPatientPK foi retirada da FactTable, bem como a relação de chave estrangeira:

```
ALTER TABLE facttable DROP dimPatientPK;
CONSTRAINT dimPacientPK
```

Figura 14 - Código de retirada da coluna dimPacientPK da FactTable

Foi criada uma nova tabela chamada dimpacientupdt constando somente as tuplas cujos valores da coluna PatientNumber eram distintos. Ao mesmo tempo, a tabela

dimpacient foi deletada do esquema, por não possuir mais serventia para a análise dos dados. Desse modo, abserva-se que o esquema determinado na seção 4 teve de ser alterado. A coluna PatientNumber foi definida como chave primária da nova tabela. O código abaixo, por sua vez, estabeleceu a relação de chave estrangeira

```
ALTER TABLE facttable
ADD CONSTRAINT patientnumberfk
FOREIGN KEY (patientnumber) REFERENCES dimpacientupdt(patientnumber)
```

Figura 15 - Estabelecimento de referência de chave estrangeira entre a coluna patientnumber da tabela FatcTable com a coluna de mesmo nome da tabela dimpacientupdt

Na figura abaixo consta o esquema do dataset atualizado:

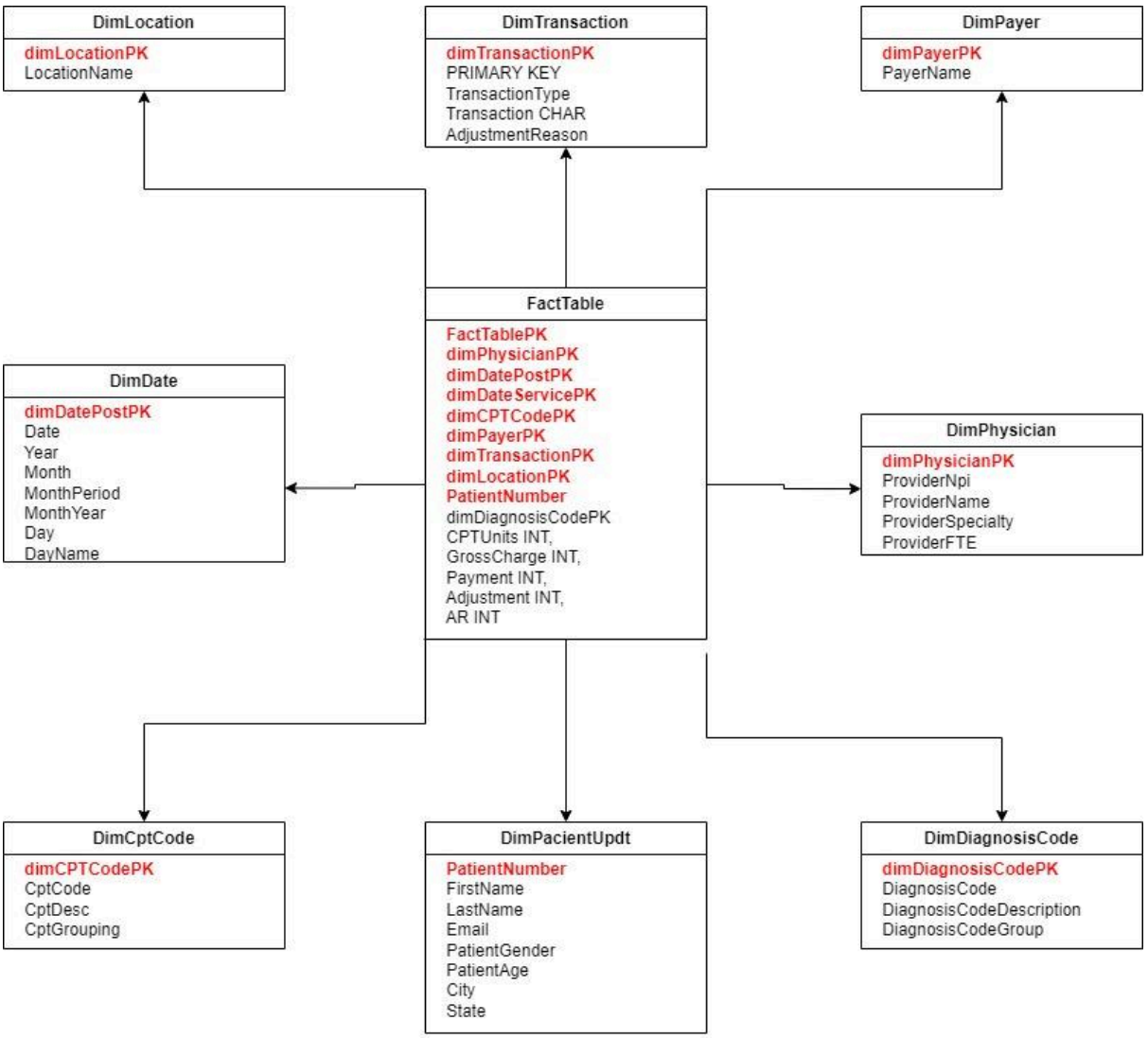

Figura 16 - Esquema de dados atualizado

5. Análise

5.1 Qualidade dos dados

5.1.1 Tabela FactTable:

Para a tabela FactTable, foi verificado se ela contém algum valor nulo e se ela continha valores outliers em suas colunas numéricas. Por meio da query abaixo procurou-se avaliar se a tabela continha valores nulos.

Para verificar se haviam valores nulos, foi escrito o código abaixo, o qual subtrai do número total de registros presente na tabela o número de registros de cada coluna, resultando no número de registros vazios:

```
SELECT
    COUNT(*) - COUNT(FactTablePK) AS FactTablePK_null,
    COUNT(*) - COUNT(PatientNumber) AS PatientNumber_null,
    COUNT(*) - COUNT(dimPhysicianPK) AS dimPhysicianPK_null,
    COUNT(*) - COUNT(dimDatePostPK) AS dimDatePostPK_null,
    COUNT(*) - COUNT(dimDateServicePK) AS dimDateServicePK_null,
    COUNT(*) - COUNT(dimCPTCodePK) AS dimCPTCodePK_null,
    COUNT(*) - COUNT(dimPayerPK) AS dimPayerPK_null,
    COUNT(*) - COUNT(dimTransactionPK) AS dimTransactionPK_null,
    COUNT(*) - COUNT(dimLocationPK) AS dimLocationPK_null,
    COUNT(*) - COUNT(dimDiagnosisCodePK) AS dimDiagnosisCodePK_null,
    COUNT(*) - COUNT(CPTUnits) AS CPTUnits_null,
    COUNT(*) - COUNT(GrossCharge) AS GrossCharge_null,
    COUNT(*) - COUNT(Payment) AS Payment_null,
    COUNT(*) - COUNT(Adjustment) AS Adjustment_null,
    COUNT(*) - COUNT(AR) AS AR_null
FROM facttable
```

Figura 17 - Query de contagem de valores nulos na tabela FactTable

Os resultados do código acima estão contidos na tabela abaixo, que mostra que não registros vazios na tabela FactTable:

	facttablepk_n bigint	patientnumbe bigint	dimphysician bigint	dimdatepostp bigint	dimdateservi bigint	dimcptcodepl bigint	dimpayerpk_r bigint	dimtransactic bigint	dimlocationpl bigint	dimdiagnosis bigint	cptunits_null bigint	grosscharge_ bigint	payment_null bigint	adjustment_n bigint	ar_null bigint
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 18 - Resultado da query da Figura 16

O código da figura aaixo, por sua vez, visa determinar se há registros duplicados na tabela FactTable:

```sql
SELECT
    FactTablePK,
    PatientNumber,
    dimPhysicianPK,
    dimDatePostPK,
    dimDateServicePK,
    dimCPTCodePK,
    dimPayerPK,
    dimTransactionPK,
    dimLocationPK,
    dimDiagnosisCodePK,
    CPTUnits,
    GrossCharge,
    Payment,
    Adjustment,
    AR,
    COUNT(*) AS duplicatas
FROM facttable
GROUP BY
    FactTablePK,
    PatientNumber,
    dimPhysicianPK,
    dimDatePostPK,
    dimDateServicePK,
    dimCPTCodePK,
    dimPayerPK,
    dimTransactionPK,
    dimLocationPK,
    dimDiagnosisCodePK,
    CPTUnits,
    GrossCharge,
    Payment,
    Adjustment,
    AR
HAVING
    COUNT(*) > 1
```

Figura 19 - Query de contagem de registros duplicados na tabela FactTable

A figura abaixo mostra a tabela resultante da query da figura acima. Foi verificado que a tabela FactTable não possui valores duplicados:

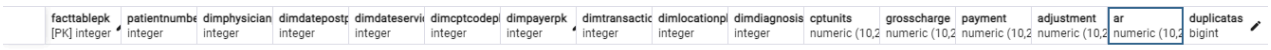

Figura 20 - Tabela resultante da query da Figura 18

A query da figura abaixo, por sua vez, tem como intuito verificar se as colunas numéricas da tabela FactTable possuem outliers, ou seja, valores que fogem desviam do comportamento normal dos registros do dataset. Foram considerados outliers todos os valores que estavam além ou abaixo da média mais ou menos 1.5 vezes o desvio padrão dos valores de suas respectivas colunas:

```
WITH OUTLIERS AS (
    SELECT
        AVG(CPTUnits) AS avg_cptunits,
        STDDEV(CPTUnits) AS stddev_cptunits,
        AVG(GrossCharge) AS avg_grosscharge,
        STDDEV(GrossCharge) AS stddev_grosscharge,
        AVG(Payment) AS avg_payment,
        STDDEV(Payment) AS stddev_payment,
        AVG(Adjustment) AS avg_adjustment,
        STDDEV(Adjustment) AS stddev_adjustment,
        AVG(AR) AS avg_ar,
        STDDEV(AR) AS stddev_ar
    FROM
        FactTable
)
SELECT
    CPTUnits,
    GrossCharge,
    Payment,
    Adjustment,
    AR
FROM
    FactTable,
    OUTLIERS
WHERE
    CPTUnits < (avg_cptunits - 1.5 * stddev_cptunits) OR
    CPTUnits > (avg_cptunits + 1.5 * stddev_cptunits) OR
    GrossCharge < (avg_grosscharge - 1.5 * stddev_grosscharge) OR
    GrossCharge > (avg_grosscharge + 1.5 * stddev_grosscharge) OR
    Payment < (avg_payment - 1.5 * stddev_payment) OR
    Payment > (avg_payment + 1.5 * stddev_payment) OR
    Adjustment < (avg_adjustment - 1.5 * stddev_adjustment) OR
    Adjustment > (avg_adjustment + 1.5 * stddev_adjustment) OR
    AR < (avg_ar - 1.5 * stddev_ar) OR
    AR > (avg_ar + 1.5 * stddev_ar)
```

Figura 21 - Query para verificação de outlier nas colunas numéricas da tabela FactTable

Abaixo se encontra a tabela resultante do código acima. Pode-se constatar que o dataset possui 7126 outliers.

	cptunits numeric (10,2)	grosscharge numeric (10,2)	payment numeric (10,2)	adjustment numeric (10,2)	ar numeric (10,2)
13	-1.00	-993.00	0.00	0.00	-993.00
14	-1.00	-864.00	0.00	0.00	-864.00
15	-1.00	-864.00	0.00	0.00	-864.00
16	-1.00	-864.00	0.00	0.00	-864.00
17	-1.00	-864.00	0.00	0.00	-864.00
18	-1.00	-864.00	0.00	0.00	-864.00
19	-1.00	-864.00	0.00	0.00	-864.00
20	-1.00	-864.00	0.00	0.00	-864.00
21	-1.00	-864.00	0.00	0.00	-864.00
22	-1.00	-864.00	0.00	0.00	-864.00

Total rows: 2000 of 7126 | Query complete 00:00:00.968

Figura 22 - Tabela resultante do código da Figura 20

5.1.2 Tabela dimtransaction:

Para a tabela dimtransaction, foi verificado se ela contém alguma valor nulo por meio da query abaixo:

```
SELECT
    COUNT(*) - COUNT(dimTransactionPK) AS dimTransactionPK_null,
    COUNT(*) - COUNT(TransactionType)AS TransactionType_null,
    COUNT(*) - COUNT(Transaction) AS Transaction_null,
    COUNT(*) - COUNT(AdjustmentReason) AS AdjustmentReason_null
FROM
    dimtransaction
```

Figura 23 - Query de contagem de valores nulos na tabela dimtransaction

Os resultados do código acima estão contidos na tabela abaixo, que mostra que não registros vazios na tabela dimtransaction:

	dimtransactionpk_null bigint	transactiontype_null bigint	transaction_null bigint	adjustmentreason_null bigint
1	0	0	0	0

Figura 24 - Tabela resultante da query da Figura 22

5.1.3 Tabela dimphysician:

Para a tabela dimphysician, foi verificado se ela contém alguma valor nulo por meio da query abaixo:

```
SELECT
    COUNT(*) - COUNT(dimPhysicianPK) AS dimPhysicianPK_null,
    COUNT(*) - COUNT(ProviderNpi) AS ProviderNpi_null,
    COUNT(*) - COUNT(ProviderName) AS ProviderName_null,
    COUNT(*) - COUNT(ProviderSpecialty) AS ProviderSpecialty_NULL,
    COUNT(*) - COUNT(ProviderFTE) AS ProviderFTE_null
FROM
    dimphysician
```

Figura 25 - Query de contagem de valores nulos na tabela dimphysician

Os resultados do código acima estão contidos na tabela abaixo, que mostra que não registros vazios na tabela dimphysician:

	dimtransactionpk_null bigint	transactiontype_null bigint	transaction_null bigint	adjustmentreason_null bigint
1	0	0	0	0

Figura 26 - Tabela resultante da query da Figura 24

5.1.4 Tabela dimpayer:

Para a tabela dimpayer, foi verificado se ela contém algum valor nulo. Como se tratava de uma tabela de apenas 4 registros, foi possível averiguar que não havia valor vazia apenas olhando para a tabela, vide a figura abaixo:

	dimpayerpk [PK] integer	payername character varying
1	58987	Commercial
2	92873	Other
3	98735	Medicare
4	765829	Medicaid

Figura 27 - Tabela dimpayer

5.1.5 Tabela dimpacientupdt:

Para a tabela dimpacientupdt, foi verificado se ela contém alguma valor nulo por meio da query abaixo:

```sql
SELECT
    COUNT(*) - COUNT(PatientNumber) AS PatientNumber_null,
    COUNT(*) - COUNT(FirstName) AS FirstName_null,
    COUNT(*) - COUNT(LastName) AS LastName_null,
    COUNT(*) - COUNT(Email) AS Email_null,
    COUNT(*) - COUNT(PatientGender) AS PatientGender_null,
    COUNT(*) - COUNT(PatientAge) AS PatientAge_null,
    COUNT(*) - COUNT(City) AS City_null,
    COUNT(*) - COUNT(State) AS State_null
FROM
    dimpacientupdt
```

Figura 28 - Query de contagem de valores nulos na tabela dimpacientupdt

Os resultados do código acima estão contidos na tabela abaixo, que mostra que não registros vazios na tabela dimpacientupdt:

	patientnumber_null bigint	firstname_null bigint	lastname_null bigint	email_null bigint	patientgender_null bigint	patientage_null bigint	city_null bigint	state_null bigint
1	0	0	0	0	0	0	0	0

Figura 29 - Tabela resultante da query da Figura 27

5.1.6 Tabela dimlocation:

Para a tabela dimdiagnosiscode, foi verificado se ela contém algum valor nulo. Como se tratava de uma tabela de apenas 11 registros, foi possível averiguar que não havia valor vazia apenas olhando para a tabela, vide a figura abaixo:

	dimlocationpk [PK] integer	locationname character varying
1	785522	Evergreen Clinic
2	785623	Twin Mountains Hospital
3	785724	Big Heart Community Hospital
4	785825	Pioneer Clinic
5	785926	Fairmont Hospital Center
6	786027	Angelstone Community Hospital
7	786128	Genesis Hospital Center
8	786229	Principal Medical Clinic
9	786330	Fairview General Hospital
10	786431	Guardian Medical Clinic
11	786532	Memorial Medical Center

Figura 30 - Tabela dimlocation

5.1.7 Tabela dimdiagnosiscode:

Para a tabela dimdiagnosiscode, foi verificado se ela contém alguma valor nulo por meio da query abaixo:

```
SELECT
    COUNT(*) - COUNT(dimDiagnosisCodePK) AS dimDiagnosisCodePK_null,
    COUNT(*) - COUNT(DiagnosisCode) AS DiagnosisCode_null,
    COUNT(*) - COUNT(DiagnosisCodeDescription) AS DiagnosisCodeDescription_null,
    COUNT(*) - COUNT(DiagnosisCodeGroup) AS DiagnosisCodeGroup_null
FROM
    dimdiagnosiscode
```

Figura 31 - Query de contagem de valores nulos na tabela dimdiagnosiscode

Os resultados do código acima estão contidos na tabela abaixo:

	dimdiagnosiscodepk_null bigint	diagnosiscode_null bigint	diagnosiscodedescription_null bigint	diagnosiscodegroup_null bigint
1	0	0	0	0

Figura 32 - Tabela resultante da query da Figura 30

5.1.8 Tabela dimdate:

Para a tabela dimdate, foi verificado se ela contém alguma valor nulo por meio da query abaixo:

```
SELECT
    COUNT(*) - COUNT(dimDatePostPK) AS dimDatePostPK_null,
    COUNT(*) - COUNT(Date) AS Date_null,
    COUNT(*) - COUNT(Year) AS Year_null,
    COUNT(*) - COUNT(Month) AS Month_null,
    COUNT(*) - COUNT(MonthPeriod) AS MonthPeriod_null,
    COUNT(*) - COUNT(MonthYear) AS MonthYear_null,
    COUNT(*) - COUNT(Day) AS Day_null,
    COUNT(*) - COUNT(DayName) AS DayName_null
FROM
    dimdate
```

Figura 33 - Query de contagem de valores nulos na tabela dimdate

Os resultados do código acima estão contidos na tabela abaixo, que mostra que não registros vazios na tabela dimdate:

	dimdatepostpk_null bigint	date_null bigint	year_null bigint	month_null bigint	monthperiod_null bigint	monthyear_null bigint	day_null bigint	dayname_null bigint
1	0	0	0	0	0	0	0	0

Figura 34 - Tabela resultante da query da Figura 32

No caso da tabela dimdate pode-se ainda avaliar um outro aspecto: se o registro de datas está adequadamente formatado. Para isso, é necessário avaliar a coluna Date por meio da query abaixo:

```
SELECT Date
FROM dimdate
LIMIT 5
```

Figura 35 - Query para exibir 5 primeiros registros da coluna Date

Observando a tabela abaixo, resultante da query da Figura 35, percebe-se que os dias, meses e anos que compõem as datas do dataset são divididos por barras (“/”), o que impede que tal coluna seja criada com o tipo “date”, o que pode dificultar operações para análise de dados no tempo.

	date character varying
1	12/15/2019
2	12/16/2019
3	12/17/2019
4	12/18/2019
5	12/19/2019

Figura 36 - Tabela resultante do código da Figura 35

5.1.9 Tabela dimcptcode:

Para a tabela dimcptcode, foi verificado se ela contém alguma valor nulo por meio da query abaixo:

```
SELECT
    COUNT(*) - COUNT(dimCPTCodePK) AS dimCPTCodePK_null,
    COUNT(*) - COUNT(CptCode) AS CptCode_null,
    COUNT(*) - COUNT(CptDesc) AS CptDesc_null,
    COUNT(*) - COUNT(CptGrouping) AS CptGrouping_null
FROM
    dimcptcode
```

Figura 37 - Query de contagem de valores nulos na tabela dimcptcode

Os resultados do código acima estão contidos na tabela abaixo, que mostra que não registros vazios na tabela dimcptcode:

	dimcptcodepk_null bigint	cptcode_null bigint	cptdesc_null bigint	cptgrouping_null bigint
1	0	0	0	0

Figura 38 - Tabela resultante da query da Figura 34

5.2 Respostas às perguntas de negócio

5.2.1 Quais são os estados onde há mais registros no dataset?

Para responder essa questão, foi usada a ação COUNT conta o número de idades presentes na tabela dimpacientupdt e a função de junção INNER JOIN para associar as tabelas FactTable e dimpacientupdt por meio da coluna PatientNumber. O comando GROUP BY foi utilizado para segmentar a contagem de itens da coluna patientage por idade. A ação ORDER BY ordenou os resultados em ordem descendente e, por fim, o comando LIMIT 10 restringe o resultado aos dez estados com mais registros no dataset:

```
SELECT dimpacientupdt.state, COUNT(dimpacientupdt.state)
FROM FactTable
INNER JOIN dimpacientupdt ON FactTable.PatientNumber=dimpacientupdt.PatientNumber
GROUP BY dimpacientupdt.state
ORDER BY COUNT(dimpacientupdt.state) DESC
```

Figura 39 - Código para contar estados que possuem mais registros no dataset

Abaixo consta a tabela gerada a partir do código acima. Ela nos mostra que o estado do Texas é o que possui mais registros no dataset:

	state character varying	count bigint
1	TX	16300
2	MT	12927
3	MA	9843
4	IL	9732
5	WV	7692
6	PA	6594
7	AL	5139
8	MS	3655
9	NY	3513
10	OK	2910
11	NH	1808
12	ME	1456
13	AZ	1166
14	KS	918
15	IN	267
16	FL	224
17	IA	55
18	OH	20

Figura 40 - Tabela resultante da query da Figura 36

5.2.2 Quais são as dez idades de pacientes que possuem mais registros no dataset?

Para responder essa questão, foi usada a ação COUNT conta o número de idades presentes na tabela dimpacientupdt e a função de junção INNER JOIN para associar as tabelas FactTable e dimpacientupdt por meio da coluna PatientNumber. O comando GROUP BY foi utilizado para segmentar a contagem de itens da coluna patientage por idade. A ação ORDER BY ordenou os resultados em ordem descendente e, por fim, o comando LIMIT 10 restringe o resultado às dez idades com mais registros no dataset:

```
SELECT dimpacientupdt.patientage, COUNT(dimpacientupdt.patientage)
FROM FactTable
INNER JOIN dimpacientupdt ON FactTable.PatientNumber=dimpacientupdt.PatientNumber
GROUP BY dimpacientupdt.patientage
ORDER BY COUNT(dimpacientupdt.patientage) DESC
LIMIT 10
```

Figura 41 - Código para contar idades que possuem mais registros no dataset

Abaixo consta a tabela gerada a partir do código acima. Ela nos mostra as dez idades que possuem mais registros no dataset:

	patientage character varying	count bigint
1	15	1484
2	27	1303
3	29	1228
4	4	1162
5	11	1158
6	65	1156
7	51	1137
8	16	1130
9	46	1115
10	69	1108

Figura 42 - Tabela resultante da query da Figura 38

5.2.3 Quais são as dez doenças que possuem mais registros no dataset?

Para responder essa questão, foi usada a ação COUNT conta o número de doenças presentes na tabela dimdiagnosiscode e a função de junção INNER JOIN para associar as tabelas FactTable e dimdiagnosiscode por meio da coluna dimdiagnosiscodepk. O comando GROUP BY foi utilizado para segmentar a contagem de itens da coluna dimdiagnosiscodepkpor idade. A ação ORDER BY ordenou os resultados em ordem descendente e, por fim, o comando LIMIT 10 restringe o resultado às dez doenças com mais registros no dataset:

```
SELECT dimdiagnosiscode.diagnosiscodedescription, COUNT(dimdiagnosiscode.diagnosiscodedescription)
FROM FactTable
INNER JOIN dimdiagnosiscode ON FactTable.dimdiagnosiscodepk=dimdiagnosiscode.dimdiagnosiscodepk
GROUP BY dimdiagnosiscode.diagnosiscodedescription
ORDER BY COUNT(dimdiagnosiscode.diagnosiscodedescription) DESC
LIMIT 10
```

Figura 43 - Código para contar as dez doenças mais registradas no dataset

Abaixo consta a tabela gerada a partir do código acima. Ela nos mostra as dez doenças que possuem mais registros no dataset:

	diagnosiscodedescription character varying	count bigint
1	Essential (primary) hypertension	4267
2	General medical examination	3991
3	Need for immunization against single bacterial diseas...	2897
4	Type 2 diabetes mellitus	2807
5	Atherosclerotic heart disease	2373
6	Special screening examination for neoplasm of breast	1782
7	Pain in throat and chest	1643
8	Dyspnoea	1601
9	Pain in joint	970
10	Adjustment and management of cardiac pacemaker	960

Figura 44 - Tabela resultante da query da Figura 40

5.2.4 Quais são as categorias de pagador que possui mais registros no dataset?

Para responder essa questão, foi usada a ação COUNT conta o número de pagadores presentes na tabela dimpayer e a função de junção INNER JOIN para associar as tabelas FactTable e dimpayer por meio da coluna dimpayerpk. O comando GROUP BY foi utilizado para segmentar a contagem de itens da coluna payername. A ação ORDER BY ordenou os resultados em ordem descendente:

```
SELECT dimpayer.payername, COUNT(dimpayer.payername)
FROM FactTable
INNER JOIN dimpayer ON FactTable.dimpayerpk=dimpayer.dimpayerpk
GROUP BY dimpayer.payername
ORDER BY COUNT(dimpayer.payername) DESC
```

Figura 43 - Código para contar o número de registros de cada tipo de pagador no dataset

	payername character varying	count bigint
1	Medicare	42452
2	Commercial	38455
3	Medicaid	2643
4	Other	669

Figura 45 - Tabela resultante da query da Figura 42

5.2.5 Quanto foi cobrado de cada uma das categorias de pagador?

Para responder essa questão, foi usada a ação SUM soma o valor cobrado de cada tipo de pagador presente na tabela dimpayer e a função de junção INNER JOIN para associar as tabelas FactTable e dimpayer por meio da coluna dimpayerpk. O comando

GROUP BY foi utilizado para segmentar a contagem de itens da coluna payername. A ação ORDER BY ordenou os resultados em ordem descendente:

```
SELECT dimpayer.payername, SUM(FactTable.grosscharge)
FROM FactTable
INNER JOIN dimpayer ON FactTable.dimpayerpk=dimpayer.dimpayerpk
GROUP BY dimpayer.payername
ORDER BY SUM(FactTable.grosscharge) DESC
```

Figura 46 - Código para determinar o valor cobrado para cada tipo de pagador presente no dataset

	payername character varying	sum numeric
1	Commercial	749129.79
2	Medicare	649989.02
3	Medicaid	68074.00
4	Other	5074.00

Figura 47 - Tabela resultante da query da Figura 44

5.3 Discussão dos resultados

Com base nos dados avaliados, pode-se afirmar que, de modo geral, os dados possuem alta qualidade. Em nenhuma das tabelas foram encontrados valores nulos ou duplicados. No caso da tabela dimdate, foi verificado que a coluna Date não suporta o tipo de dados “date”, o que pode dificultar análises envolvendo tempo. A respeito dos outliers, eles representam menos de 10% dos registros totais da FactTable do dataset, representando a minoria dos seus dados totais.

Em relação às perguntas de negócio, de modo geral não foi possível estabelecer um padrão claro entre os dados por meio de sua mera contagem ou soma. Para entender mais claramente as razões que levam os dados a serem como são, é necessário um maior aprofundamento nas características do negócio analisado, no caso, o sistema de planos de saúde americano, de modo a cruzar as diferentes informações para a determinação de conhecimento relevante. No entanto, avaliando a tabela da Figura 45, percebe-se que os valores cobrados para os pagadores do tipo “Commercial” e “Medicare” foram muito superiores aos demais, podendo ser um bom ponto de partida para análises mais aprofundadas dos dados do dataset.

6. Autoavaliação

Por meio deste trabalho foi possível exercitar e aprofundar os conceitos e técnicas ensinados no módulo de Engenharia de Dados da Pós-Graduação em Ciência de Dados da PUC-Rio. Entre os ensinamentos valiosos, pode-se citar a modelagem, o desenvolvimento de queries em SQL e a análise da qualidade dos dados.

Em relação a boas práticas que podem ser adotadas em futuros projetos, pode-se citar uma melhor análise do dataset original. No caso tratado neste trabalho, foi necessário alterar a modelagem do esquema do dataset, o que poderia ter sido evitado por meio de

uma análise mais minuciosa dos dados existentes, de modo a evitar retrabalhos. Além disso, é inegável que um maior aprofundamento no negócio aos quais os dados se referem é essencial para que análises mais aprofundadas e relevantes possam ser feitas. Outra melhoria para possíveis projetos futuros é a adoção de ferramentas que possam tornar a análise de dados mais visível.