UNIVERSIDADE DE BRASÍLIA
2º SEMESTRE DE 2016

PROFESSOR: Maristela Holanda
TURMA: A
ALUNOS: Davi Rabbouni de Carvalho Freitas – 15/0033010
Marcelo de Araújo 15/0016794
Rafael Chehab 15/0045123
Marcelo Axel 15/0080727

BANCO DE DADOS
PROJETO FINAL - DOCUMENTAÇÃO

Introdução

O projeto proposto consiste na construção de uma estrutura de Banco de Dados baseada nas informações de Diárias e Passagens encontradas no portal de transparência (http://www.portaltransparencia.gov.br/), assim como a importação dos dados disponibilizados pelo governo federal para o período entre Julho-Dezembro de 2015.

A construção do Banco de Dados propriamente dito abrange o Projeto do Banco de Dados, com a elaboração do Diagrama de Entidade Relacionamento e o Modelo Relacional, a implementação física da estrutura - com o script de definição do banco -, a inserção dos dados obtidos no sítio do governo e, por último, operações de manipulação e controle do BD, como queries, triggers, procedures e views.

O projeto descrito, assim, trabalha a grande maioria dos conceitos vistos em sala de aula, e permite o docente avaliar os conhecimentos adquiridos pelos alunos ao longo do curso.

Diagrama de Entidade Relacionamento

O Diagrama de Entidade Relacionamento apresentado abaixo foi construído com base no arquivo *csv* obtido do portal da transparência relativo aos meses mencionados. Construiu-se o modelo utilizando a ferramenta *PgModeler*.

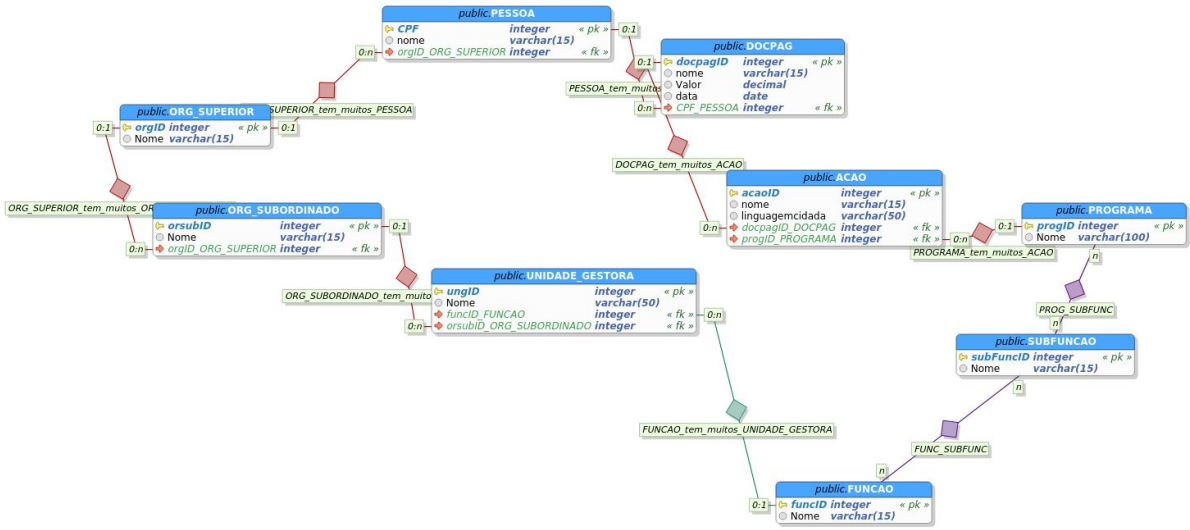

Figura 1. DER do BD

Modelo Relacional

O modelo relacional foi construído tendo como base o DER construído acima, tendo sido elaborado com a ferramenta de modelagem do *MySQLWorkbench*.

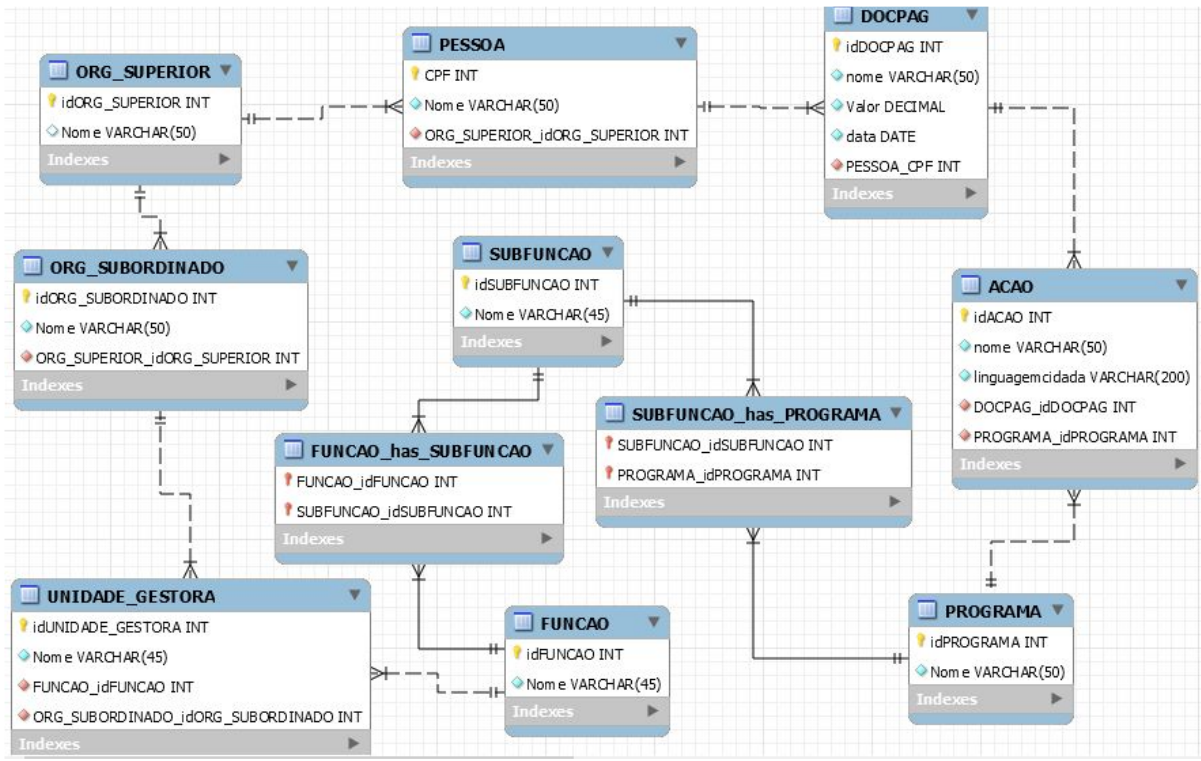

Fig 2. MR do BD

Normalização

1FN

A tabela encontra-se inicialmente da seguinte forma, com todas as colunas formando uma tabela só:

Cód. Órg. Superior	**Nome Órg. Superior**	**Cód Órg. Subordinado**	**Nome Órg. Subordinado**	**Cód Unidade Gestora**	**Nome Unidade Gestora**	**Cód Função**	**Nome Função**	**Cód Subfunção**	**Nome Subfunção**

Cód. Programa	**Nome Programa**	**Cód. Ação**	**Nome Ação**	**Linguagem Cidadã**	**CPF Favorecido**	**Nome Favorecido**	**Documento Pagamento**	**Gestão Pagamento**	**Data Pagamento**	**Valor Pagamento**

Pode-se ver que o valor de todas as colunas da tabela são indivisíveis, o que garante a 1° forma normal.

2FN

Apesar dos valores serem indivisíveis, muitas redundâncias são produzidas com isso. Assim, utiliza-se a propriedade de dependência funcional para obter:

Primeiramente, tem-se como candidatos a chave:

- Cód. Órg. Superior,
- Cód Órg. Subordinado,
- Cód Unidade Gestora ,
- Cód Função,
- Cód Subfunção,
- Cód. Programa,
- Cód. Ação,
- CPF Favorecido,

- Documento Pagamento.

Cód. Órg. Superior → Nome Órg. Superior;
Cód Órg. Subordinado → Nome Órg. Subordinado;
Cód Unidade Gestora → Nome Unidade Gestora;
Cód Função → Nome Função;
Cód Subfunção → Nome Subfunção;
Cód. Programa → Nome Programa;
Cód. Ação → Nome Ação;
Cód. Ação → Linguagem Cidadã;
CPF Favorecido → Nome Favorecido;
Documento Pagamento → Gestão Pagamento;
Documento Pagamento → Data Pagamento;
Documento Pagamento → Valor Pagamento

Isso produz:

Cód. Órg. Superior	**Nome Órg. Superior**

Tabela 1

Cód Órg. Subordinado	**Nome Órg. Subordinado**

Tabela 2

Cód Unidade Gestora	**Nome Unidade Gestora**

Tabela 3

Cód Função	**Nome Função**

Tabela 4

Cód Subfunção	**Nome Subfunção**

Tabela 5

Cód. Programa	**Nome Programa**

Tabela 6

Cód. Ação	**Nome Ação**	**Linguagem Cidadã**

Tabela 7

CPF Favorecido	**Nome Favorecido**

Tabela 8

Documento Pagamento	**Gestão Pagamento**	**Data Pagamento**	**Valor Pagamento**

Tabela 9

3FN

Observando as tabelas da 2° forma normal do Banco de Dados proposto, tem-se que o atributo Linguagem Cidadã possui uma dependência transitiva da chave primária *Cód. Ação,* por ser transitivamente dependente do atributo Nome Ação.

Com isso, tem-se:

Cód. Órg. Superior	**Nome Órg. Superior**

Tabela 1

Cód Órg. Subordinado	**Nome Órg. Subordinado**

Tabela 2

Cód Unidade Gestora	**Nome Unidade Gestora**

Tabela 3

Cód Função	**Nome Função**

Tabela 4

Cód Subfunção	**Nome Subfunção**

Tabela 5

Cód. Programa	**Nome Programa**

Tabela 6

Cód. Ação	**Nome Ação**

Tabela 7

Nome Ação	**Linguagem Cidadã**

Tabela 8

CPF Favorecido	**Nome Favorecido**

Tabela 9

Documento Pagamento	Gestão Pagamento	Data Pagamento	Valor Pagamento

Tabela 10

Script SQL

Com base no Modelo Relacional previamente apresentado, elaborou-se o seguinte script para o postgreSQL:

```
1  DROP TABLE IF EXISTS public.ORG_SUPERIOR CASCADE;
2  CREATE TABLE public.ORG_SUPERIOR(
3          orgID integer NOT NULL,
4          Nome varchar(50),
5          CONSTRAINT PK PRIMARY KEY (orgID)
6  );
7
8  ALTER TABLE public.ORG_SUPERIOR OWNER TO postgres;
9
10 DROP TABLE IF EXISTS public.ORG_SUBORDINADO CASCADE;
11 CREATE TABLE public.ORG_SUBORDINADO(
12         orsubID integer NOT NULL,
13         Nome varchar(50),
14         orgID_ORG_SUPERIOR integer,
15         CONSTRAINT SUBPK PRIMARY KEY (orsubID)
16 );
17
18 ALTER TABLE public.ORG_SUBORDINADO OWNER TO postgres;
19
20 ALTER TABLE public.ORG_SUBORDINADO DROP CONSTRAINT IF EXISTS
21 ORG_SUPERIOR_fk CASCADE;
22 ALTER TABLE public.ORG_SUBORDINADO ADD CONSTRAINT
23 ORG_SUPERIOR_fk FOREIGN KEY (orgID_ORG_SUPERIOR)
24 REFERENCES public.ORG_SUPERIOR (orgID) MATCH FULL
25 ON DELETE SET NULL ON UPDATE CASCADE;
26
27 DROP TABLE IF EXISTS public.UNIDADE_GESTORA CASCADE;
28 CREATE TABLE public.UNIDADE_GESTORA(
29         ungID integer NOT NULL,
30         Nome varchar(50),
31         funcID_FUNCAO integer,
32         orsubID_ORG_SUBORDINADO integer,
33         CONSTRAINT uniPK PRIMARY KEY (ungID)
34
```

```
);
ALTER TABLE public.UNIDADE_GESTORA OWNER TO postgres;

DROP TABLE IF EXISTS public.FUNCAO CASCADE;
CREATE TABLE public.FUNCAO(
	funcID integer NOT NULL,
	Nome varchar(50),
	CONSTRAINT funcPK PRIMARY KEY (funcID)
);
ALTER TABLE public.FUNCAO OWNER TO postgres;

ALTER TABLE public.UNIDADE_GESTORA DROP CONSTRAINT IF EXISTS
FUNCAO_fk CASCADE;
ALTER TABLE public.UNIDADE_GESTORA ADD CONSTRAINT FUNCAO_fk
FOREIGN KEY (funcID_FUNCAO)
REFERENCES public.FUNCAO (funcID) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE public.UNIDADE_GESTORA DROP CONSTRAINT IF EXISTS
ORG_SUBORDINADO_fk CASCADE;
ALTER TABLE public.UNIDADE_GESTORA ADD CONSTRAINT
ORG_SUBORDINADO_fk FOREIGN KEY (orsubID_ORG_SUBORDINADO)
REFERENCES public.ORG_SUBORDINADO (orsubID) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;

DROP TABLE IF EXISTS public.SUBFUNCAO CASCADE;
CREATE TABLE public.SUBFUNCAO(
	subFuncID integer NOT NULL,
	Nome varchar(50),
	CONSTRAINT subfuncPK PRIMARY KEY (subFuncID)
);


ALTER TABLE public.SUBFUNCAO OWNER TO postgres;

DROP TABLE IF EXISTS public.FUNC_SUBFUNC CASCADE;
CREATE TABLE public.FUNC_SUBFUNC(
	subFuncID_SUBFUNCAO integer,
	funcID_FUNCAO integer,
	CONSTRAINT FUNC_SUBFUNC_pk PRIMARY KEY
(subFuncID_SUBFUNCAO,funcID_FUNCAO)
);

ALTER TABLE public.FUNC_SUBFUNC DROP CONSTRAINT IF EXISTS
SUBFUNCAO_fk CASCADE;
ALTER TABLE public.FUNC_SUBFUNC ADD CONSTRAINT SUBFUNCAO_fk
FOREIGN KEY (subFuncID_SUBFUNCAO)
REFERENCES public.SUBFUNCAO (subFuncID) MATCH FULL

```

```
ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE public.FUNC_SUBFUNC DROP CONSTRAINT IF EXISTS
FUNCAO_fk CASCADE;
ALTER TABLE public.FUNC_SUBFUNC ADD CONSTRAINT FUNCAO_fk FOREIGN
KEY (funcID_FUNCAO)
REFERENCES public.FUNCAO (funcID) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;

DROP TABLE IF EXISTS public.PROGRAMA CASCADE;
CREATE TABLE public.PROGRAMA(
        progID integer NOT NULL,
        Nome varchar(100),
        CONSTRAINT COD PRIMARY KEY (progID)
);

ALTER TABLE public.PROGRAMA OWNER TO postgres;

DROP TABLE IF EXISTS public.PROG_SUBFUNC CASCADE;
CREATE TABLE public.PROG_SUBFUNC(
        progID_PROGRAMA integer,
        subFuncID_SUBFUNCAO integer,
        CONSTRAINT PROG_SUBFUNC_pk PRIMARY KEY
(progID_PROGRAMA,subFuncID_SUBFUNCAO)

);

ALTER TABLE public.PROG_SUBFUNC DROP CONSTRAINT IF EXISTS
PROGRAMA_fk CASCADE;
ALTER TABLE public.PROG_SUBFUNC ADD CONSTRAINT PROGRAMA_fk
FOREIGN KEY (progID_PROGRAMA)
REFERENCES public.PROGRAMA (progID) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE public.PROG_SUBFUNC DROP CONSTRAINT IF EXISTS
SUBFUNCAO_fk CASCADE;
ALTER TABLE public.PROG_SUBFUNC ADD CONSTRAINT SUBFUNCAO_fk
FOREIGN KEY (subFuncID_SUBFUNCAO)
REFERENCES public.SUBFUNCAO (subFuncID) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;

DROP TABLE IF EXISTS public.ACAO CASCADE;
CREATE TABLE public.ACAO(
        acaoID integer NOT NULL,
        nome varchar(200),
        linguagemcidada varchar(200),
        progID_PROGRAMA integer,
        CONSTRAINT acaoPK PRIMARY KEY (acaoID)
```

```
);

ALTER TABLE public.ACAO OWNER TO postgres;

DROP TABLE IF EXISTS public.DOCPAG CASCADE;
CREATE TABLE public.DOCPAG(
	docpagID integer NOT NULL,
	nome varchar(50),
	Valor decimal,
	data date,
	gestao integer,
	CPF_PESSOA integer,
	acaoID_ACAO integer,
	CONSTRAINT dogpagPK PRIMARY KEY (docpagID)
);

ALTER TABLE public.DOCPAG OWNER TO postgres;

DROP TABLE IF EXISTS public.PESSOA CASCADE;
CREATE TABLE public.PESSOA(
	CPF integer NOT NULL,
	nome varchar(50),
	orgID_ORG_SUPERIOR integer,
	CONSTRAINT cpfPK PRIMARY KEY (CPF)
);

ALTER TABLE public.PESSOA OWNER TO postgres;

ALTER TABLE public.DOCPAG DROP CONSTRAINT IF EXISTS PESSOA_fk
CASCADE;
ALTER TABLE public.DOCPAG ADD CONSTRAINT PESSOA_fk FOREIGN KEY
(CPF_PESSOA)
REFERENCES public.PESSOA (CPF) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE public.ACAO DROP CONSTRAINT IF EXISTS PROGRAMA_fk
CASCADE;
ALTER TABLE public.ACAO ADD CONSTRAINT PROGRAMA_fk FOREIGN KEY
(progID_PROGRAMA)
REFERENCES public.PROGRAMA (progID) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE public.PESSOA DROP CONSTRAINT IF EXISTS ORG_SUPERIOR_fk
CASCADE;
ALTER TABLE public.PESSOA ADD CONSTRAINT ORG_SUPERIOR_fk FOREIGN
KEY (orgID_ORG_SUPERIOR)
REFERENCES public.ORG_SUPERIOR (orgID) MATCH FULL
ON DELETE SET NULL ON UPDATE CASCADE;
```

```
179
180 ALTER TABLE public.DOCPAG DROP CONSTRAINT IF EXISTS ACAO_fk
181 CASCADE;
182 ALTER TABLE public.DOCPAG ADD CONSTRAINT ACAO_fk FOREIGN KEY
183 (acaoID_ACAO)
184 REFERENCES public.ACAO (acaoID) MATCH FULL
185
186 ON DELETE SET NULL ON UPDATE CASCADE;
```

ETL

ETL é um processo de extração, transformação e "carga" para a construção de um Data Warehouse (DW). Antes de prosseguir com os conceitos derivados da sentença acima, é conveniente explicar brevemente o que é um DW.

Data Warehouse é um banco de dados construído com o objetivo de proporcionar um base para uma empresa, podendo suportar uma organização de um grande volume de dados. Ele permite a integração de dados de diversas fontes diferentes em apenas uma estrutura, a fim de facilitar a visualização e manipulação desses.

O processo de ETL é, assim, fundamental nesse processo de integração: os três passos desse método permitem a transformação de um arquivo de dados, com um arquivo CSV, em um banco de dados organizado e utilizável.

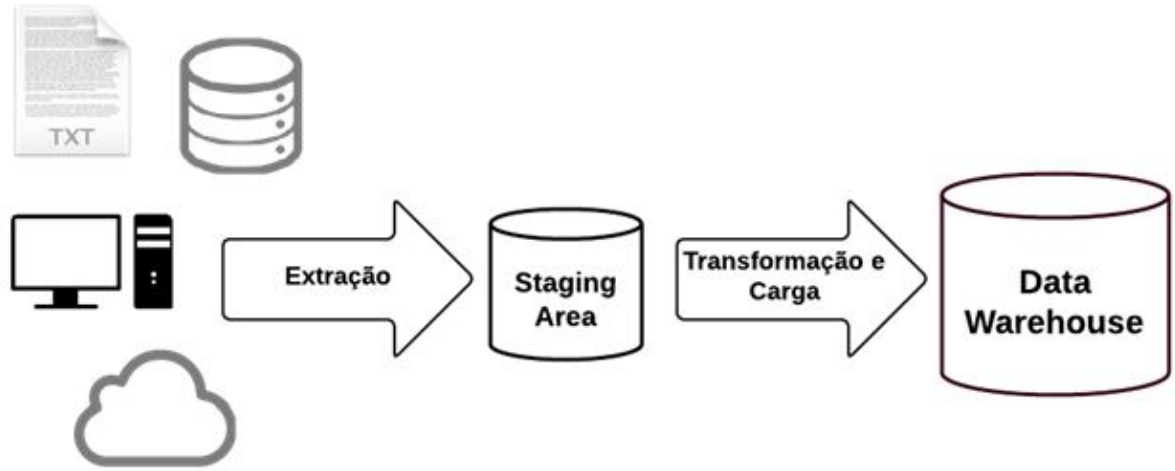

Fig. 3 - Processo de ETL

A primeira etapa – extração -, permite a definição dos conteúdos mais relevantes que constituirão o Banco, e elaboração da estrutura a partir dos dados extraídos dos diversas fontes.

Já a etapa de transformação (também chamada de tratamento) é caracterizada basicamente pela "limpeza" dos dados obtidos, como a padronização dos dados, definição dos tipos, correção de caracteres desconhecidos e erros de digitação, para citar alguns.

A última etapa – carga ou *Load* - consiste no carregamento de dados para o DW e o comprometimento com uma boa performance da estrutura, o que depende da quantidade de dados a ser tratada.

No caso do projeto em questão, o processo de ETL foi feito basicamente em 4 etapas:

1) Formatação da tipagem e manipulação dos dados obtidos do arquivo csv por meio *Talend Data Preparation;*
2) Montagem das colunas de interesse para criar arquivos csv contendo as colunas de cada tabela elaborada;
3) Remoção das linhas duplicadas utilizando um filtro no LibreOfficeCalc;
4) Por último, fez-se um script no postgreSQL para importar os dados contidos nos arquivos nas tabelas propriamente ditas:

```
copy public.org_superior from
'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/ORG_SUPERIOR.csv' with (FORMAT csv,
DELIMITER E'\t');
copy public.org_subordinado from
'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/ORG_SUBORDINADO.csv' with (FORMAT csv,
DELIMITER E'\t');
copy public.pessoa from 'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/PESSOA.csv' with
(FORMAT csv, DELIMITER E'\t');
copy public.funcao from 'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/FUNCAO.csv' with
(FORMAT csv, DELIMITER E'\t');
copy public.unidade_gestora from
'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/UNIDADE_GESTORA.csv' with (FORMAT csv,
DELIMITER E'\t');
copy public.subfuncao from
'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/SUBFUNCAO.csv' with (FORMAT csv,
DELIMITER E'\t');
copy public.programa from 'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/PROGRAMA.csv'
with (FORMAT csv, DELIMITER E'\t');
copy public.acao from 'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/ACAO.csv' with
(FORMAT csv, DELIMITER E'\t');
copy public.docpag from 'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/DOCPAG.csv' with
(FORMAT csv, DELIMITER E'\t');
copy public.func_subfunc from
'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/muitos_SUBFUNCAO_tem_muitos_FUNCAO.csv'
with (FORMAT csv, DELIMITER E'\t');
copy public.prog_subfunc from
'path_to_bd/BD-FINAL-2016-2/Diarias/D-1507/PROG_SUBFUNC.csv' with (FORMAT csv,
DELIMITER E'\t');
```

Camada de Persistência e Interface Gráfica

A camada de persistência foi implementada na linguagem de programação Java, utilizando-se o framework de mapeamento objeto relacional *Hibernate* associado ao framework de persistência *Java Persistence API* (JPA).

Com base nos frameworks em questão, utilizou-se para a camada de persistência o padrão de projeto de nome *Data Access Object* (DAO), que permite a criação de classes que possuem métodos de acesso ao banco de dados do projeto. Vale acrescentar, ainda, que a consulta nessas classes foi feita pela linguagem de consulta fornecida pelo *Hibernate*, *Hibernate Query Language*.

Tendo em vista a construção das classes em questão, foi possível a elaboração de queries, as quais poderão ser utilizadas pela interface gráfica:

1) Quanto uma determinada pessoa ganhou de diárias no período entre Julho e Dezembro - busca por CPF da pessoa:

- *SELECT SUM(dp.valor) FROM docpag AS dp*
  JOIN pessoa AS p ON dp.cpf_pessoa = p.cpf
  WHERE p.cpf = :pmtCPF;

  Onde :pmtCPF é o CPF da pessoa a ser digitado pelo usuário

2) Quanto um determinado Órgão Superior gastou em diárias no período entre Julho e Dezembro - busca pelo nome do Órgão:

- *SELECT SUM(dp.valor) FROM docpag AS dp*
  JOIN pessoa AS p ON dp.cpf_pessoa = p.cpf
  JOIN org_superior AS os ON os.orgid = p.orgid_org_superior
  WHERE os.nome LIKE :pmtNome;

  Onde :pmtNome é o nome do Órgão a ser fornecido pelo usuário.

3) Quanto cada uma das pessoas do Banco gastou no período inteiro de Julho-Dezembro, ordenado de forma decrescente pelo valor gasto:

- *select p.nome, sum(dp.valor) from docpag as dp*
  join pessoa as p on dp.cpf_pessoa = p.cpf
  group by p.nome
  order by sum(dp.valor) desc, p.nome;

4) Conjunto das 100 pessoas que mais viajaram no período Julho-Dezembro:

- *select p.nome, count(dp.valor) from docpag as dp*
  join pessoa as p on dp.cpf_pessoa = p.cpf
  group by p.nome
  order by count(dp.valor) desc, p.nome
  limit 100
  offset 0;

5) Conjunto de todas as viagens pela pessoa que mais gastou no período

- *select sel1.nome, dp2.* from*
  (select p.nome, p.cpf, count(dp.valor) from docpag as dp
  join pessoa as p on dp.cpf_pessoa = p.cpf
  group by p.cpf, p.nome
  order by count(dp.valor) desc, p.nome
  limit 1
  offset 0) as sel1
  join docpag as dp2 on dp2.cpf_pessoa = sel1.cpf
  order by dp2.data;

A interface gráfica, por sua vez, foi concebida por meio do *Java Swing,* de forma a chamar as classes e métodos implementados na Camada de Persistência, sem a necessidade de possuir alguma linguagem de consulta em sua estrutura.

Views

1) View que representa a porcentagem dos gastos totais que um Órgão Superior teve, e a soma de seus respectivos gastos:

- *create view gastos_org_superiores as*
  (select os., 100 * sum(dp.valor) / tot.total as porcentagem_de_gastos, sum(dp.valor) as soma_gastos from org_superior as os*
  join pessoa as p on os.orgID = p.orgID_ORG_SUPERIOR
  join docpag as dp on p.cpf = dp.cpf_pessoa
  join (select sum(dp.valor) as total from docpag as dp) as tot on true
  group by os.orgID, tot.total
  *order by 100 * sum(dp.valor) / tot.total desc);*

2) View que representa a porcentagem dos gastos totais que uma Pessoa teve, e a soma de seus gasto:

- *create view gastos_pessoas as*
  (select p., 100 * sum(dp.valor) / tot.total as porcentagem_de_gastos, sum(dp.valor) as soma_gastos from pessoa as p*
  join docpag as dp on p.cpf = dp.cpf_pessoa
  join (select sum(dp.valor) as total from docpag as dp) as tot on true
  group by p.cpf, tot.total
  *order by 100 * sum(dp.valor) / tot.total desc);*

Triggers e Procedures

Utilizou-se um conjunto trigger com procedures para armazenar, em uma tabela de backup, os dados das pessoas excluídas.

```
CREATE TABLE bk_pessoa (
  cpf integer NOT NULL,
  nome character varying(50),
  orgid_org_superior integer
);


-- Stored Procedure
CREATE OR REPLACE FUNCTION salvaexcluido()
RETURNS trigger AS
$BODY$ BEGIN
INSERT INTO bk_pessoa VALUES (old.cpf, old.nome, old.orgid_org_superior);

RETURN NULL;

END; $BODY$
LANGUAGE 'plpgsql';

CREATE TRIGGER excluir_pessoa
AFTER DELETE
ON pessoa
FOR EACH ROW
EXECUTE PROCEDURE salvaexcluido();

```

Conclusão

Por tudo o que foi apresentado, pode-se afirmar que o projeto apresentado foi realizado de forma satisfatória, uma vez que os principais objetivos iniciais foram cumpridos e os conhecimentos desejados na conclusão da disciplina foram aplicados na prática.