Relatório do Projeto de Banco de Dados 1 (INF1383)

Professor Sérgio Lifschitz

Grupo:
Carlos Augusto Lima Mattoso - 1210553
Felipe Luiz Teixeira da Rocha - 1210586
Gabriel da Silva Siqueira - 1210689
Guilherme de Campos Lima Berger - 1210518
Leonardo Krause Lipet Slipoi Kaplan - 1212509

Especificação

O projeto deste grupo consiste no desenvolvimento de uma aplicação para registro e gerenciamento de voluntários e funcionários de uma ONG e das atividades exercidas por estes. Tal projeto teve origem na necessidade da Cruz Vermelha, filial RJ, de reorganizar o processo vigente, que consiste em um arquivo físico de formulários preenchidos e na manutenção de uma planilha no Microsoft Excel, sendo, portanto, ineficiente.

O modelo de dados necessário deverá armazenar os dados pessoais básicos de cada pessoa (nome, filiação, endereço residencial, número e órgão emissor do RG, CPF, naturalidade, estado civil, sexo) e informações complementares (profissão, e-mail, tipo sanguíneo, se possui e tipo da CNH, telefones para contato, formação, habilidade em línguas estrangeiras, outras qualificações e experiências).

Os dados das filiais também são cadastrados. O modelo administrativo da ONG dita que só pode haver uma filial por estado; logo, este é um dado essencial e único no seu cadastro. Também deve ser armazenado o endereço de cada filial.

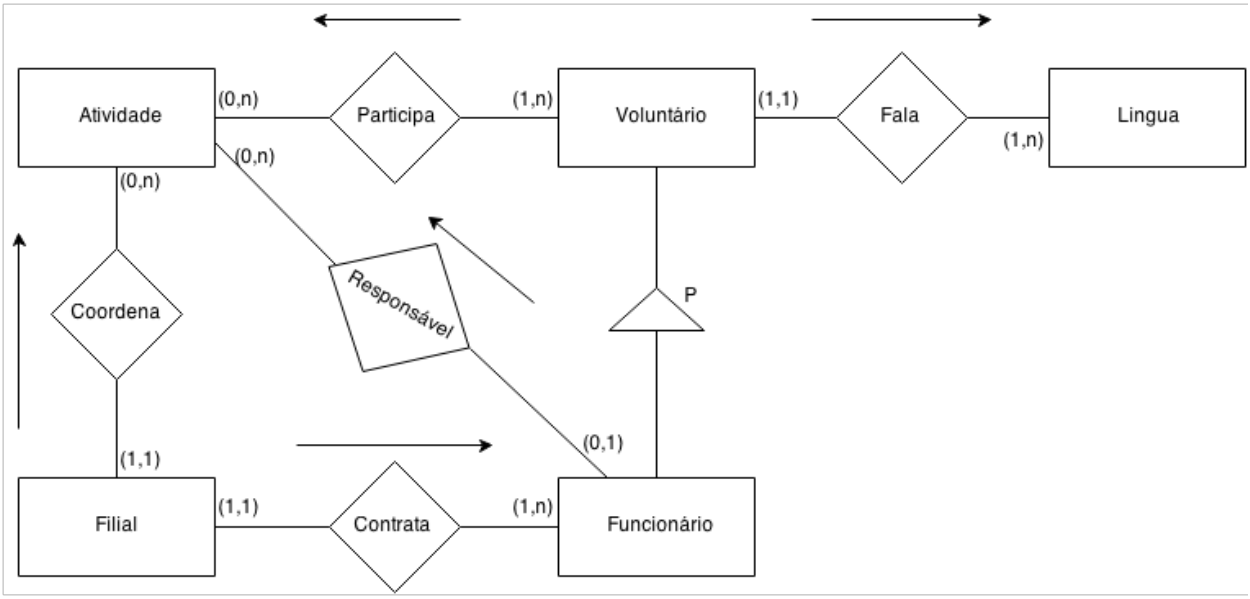
Haverá um registro de funcionários da organização que, além dos dados já armazenados para uma pessoa, possuem nome de usuário e senha para que possam acessar a aplicação de cadastro de voluntários. Cada funcionário pertence a uma filial da organizaçao.

Além disso, todas as atividades exercidas pelo voluntário deverão ter nome, data, local e breve descrição registrados. A cada atividade poderá ser atribuída um funcionário responsável. Serão também salvas a quantidade de horas trabalhadas e as tarefas realizadas por cada voluntário em cada atividade.

As possíveis consultas são: busca de voluntários que apresentem habilidades necessárias para um evento, como domínio de uma certa língua estrangeira; exibir todas as atividades realizadas por um voluntário no último mês; obter estatísticas acerca da realização de atividades por filias e dos segmentos sociais dos voluntários cadastrados; dentre outras.

Modelo Entidade Relacionamento

Diagrama



Lista de Atributos

Os atributos abaixo sublinhados compõem o identificador da entidade.

Pessoa

- CPF (1,1)
- nome (1, 1)
- dataNasc (1,1)
- nomeMae (0, 1)
- nomePai (0, 1)
- endereçoResidencial (1, 1)
- númeroDoRG (1, 1)
- órgãoEmissorRG (1, 1)
- naturalidade (1,1)
- estadoCivil (1,1)
- sexo(1,1)
- profissão (0, 1)
- e-mail (0,1)

- tipoSanguíneo (1,1)
- tipoCNH (0,1)
- telefones (1,n)
- formação (0,1)
- obs (0,n)

Atividade

- código (1,1)
- nome(1,1)
- data (1,1)
- endereço (1,1)
- descrição (1,1)

Participação

- tarefasRealizadas (1,1)
- horasTrabalhadas (1,1)

Lingua

- codigo(1,1)
- nome (1,1)
- dialeto (0,n)

Fala

- nivel (1,1)

Filial

- codigo(1,1)
- estado(1,1)
- endereço(1,1)

Funcionario

- username(1,1)
- password(1,1)

Modelo Relacional

Os nomes de colunas abaixo sublinhados compõem as chaves primárias de cada tabela, enquanto os nomes em itálico constituem uma chave estrangeira.

Tabelas

Pessoa(CPF, Nome,DataNasc, NomeMae, NomePai, Endereco, NumRG, EmissorRG, Natural, EstCivil, Sexo, Profissao, Email, TipoSangue, TipoCNH, Tel1, Tel2, Formacao, Obs)
Filial(Codigo, Estado, Endereço)
Funcionario(*CPF*, *CodFilial*, Username, Password)

CPF referencia Pessoa

CodFilial referencia Filial

Atividade(Codigo, *CPFResponsavel*, Nome, Data, Endereco, Descricao)

CPFResponsavel referencia Funcionario

Lingua(Codigo, Nome, Dialeto)

Fala(*VolCPF*, *CodLing*, Nivel)

VolCPF referencia Pessoa

CodLing referencia Lingua

Participacao(*VolCPF*, *CodAtiv*, HorasTrab, TarefasRealizadas)

VolCPF referencia Pessoa

CodAtiv referencia Atividade

Descrição dos Atributos

Pessoa

- CPF [string / não nulo]
    - número do CPF da pessoa, conforme emitido pela Receita Federal, contendo apenas algarismos
    - o tamanho fixo de um CPF, sem considerar '-' e '.', é de 11

dígitos

- Nome [string / não nulo]
    - nome completo da pessoa, conforme documento de identificação oficial
- DataNasc [date / não nulo]
    - data de nascimento da pessoa, no formato yyyy-mm-dd.
- NomeMae [string]
    - nome completo da mãe da pessoa, conforme documento de identificação oficial
- NomePai [string]
    - nome completo da mãe da pessoa, conforme documento de identificação oficial
- Endereço [string / não nulo]
    - endereço residencial da pessoa, que deve ser suficiente para envio de correspondências
- NumRG [string / não nulo]
    - número do documento de identificação (passaporte, caso seja estrangeiro) da pessoa, contendo o dígito verificador, se houver, e utilizando apenas algarismos
- EmissorRG [string / não nulo]
    - sigla do órgão emissor do RG cujo número foi informado em NumRG e a sigla de seu estado, separados por uma barra
- Natural [string / não nulo]
    - estado de nascimento do indivíduo, caso o mesmo seja brasileiro, ou o país de nascimento, caso contrário
- EstCivil [string / não nulo]
    - estado civil da pessoa, sendo as seguintes opções válidas: "solteiro(a)", "casado(a)", "viúvo(a)" ou "divorciado(a)"
- Sexo [char / não nulo]
    - sexo da pessoa, sendo o valor 'M' para masculino e 'F' para feminino
- Profissão [string]

        - nome da ocupação atual da pessoa
- Email [string]
    - e-mail da pessoa, válido para correspondências
    - o tamanho máximo de um endereço de e-mail é 254 [1]
- TipoSangue [string / não nulo]
    - tipo sanguíneo da pessoa ('A', 'B', 'AB' ou 'O'), seguido do fator Rh ('+' ou '-')
- TipoCNH [string]
    - categoria da Carteira Nacional de Habilitação da pessoa, caso este possua. Valores válidos são 'A', 'B', 'C', 'D', 'E', 'AB', 'AC', 'AD' e 'AE'
- Tel1 [string / não nulo]
    - número de telefone da pessoa.
    - o comprimento númerico máximo de um número de telefone internacional é de 15 dígitos[2]
- Tel2 [string]
    - número de telefone alternativo da pessoa, caso haja.
- Formação [string]
    - formação atual da pessoa, dentre as opções 'fundamental incompleto', 'fundamental completo', 'médio incompleto', 'médio completo', 'superior incompleto' ou 'superior completo'
- Obs [string]
    - campo opcional para inclusão de quaisquer comentários relevantes à ficha cadastral da pessoa

Atividade

- Código [string / não nulo]
    - código utilizado para identificar uma determinada atividade
- Nome [string / não nulo]
    - nome da atividade

[1] http://stackoverflow.com/questions/386294/what-is-the-maximum-length-of-a-valid-email-address
[2] http://en.wikipedia.org/wiki/E.164

- Data [data / não nulo]
    - data de realização da atividade
    - é utilizado o formato yyyy-mm-dd
- Endereço [string / não nulo]
    - local de realização da atividade
- Descrição [string / não nulo]
    - breve descrição da atividade

Lingua

- Codigo [string / não nulo]
    - código utilizado para identificar uma determinada língua
- Nome [string / não nulo]
    - nome da língua
- Dialeto [string / não nulo]
    - variedade de uma determinada língua, por exemplo, português brasileiro ou de Portugal

Fala

- Nivel [enum / não nulo]
    - grau de proficiência de um voluntário em determinada língua

Participacao

- HorasTrab [int / não nulo]
    - quantidades de horas trabalhadas por um voluntário em uma atividade
    - não pode ser superior a 8h
- TarefasRealizadas[string / não nulo]
    - breve descrições das tarefas realizadas por um voluntário em determinada atividade

Filial

- Codigo [int / não nulo]
    - código utilizado para identificar uma determinada filial

- Estado [string / não nulo]
    - estado do Brasil onde a filial se encontra
    - é únique index, pois só existe uma filial por estado
- Endereço [string / não nulo]
    - endereço da filial, que deve ser suficiente para envio de correspondências

Funcionario

- Username[string / não nulo]
    - Nome de usuário para login no sistema
- Password [string / não nulo]
    - Senha do usuário para login no sistema

Restrições Semânticas

- Um voluntário não pode participar mais de 8 horas numa atividade;
- Os telefones não podem ser ambos os mesmos
- A coluna nível da tabela Fala pode assumir três valores: 'iniciante', 'medio', 'fluente'.
- Um voluntário não pode ter menos de 18 anos.

Avaliação de Normalização

Dependências funcionais

Pessoa (CPF, Nome, NomeMae, NomePai, Endereco, NumRG, EmissorRG, Natural, EstCivil, Sexo, Profissao, Email, TipoSangue, TipoCNH, Tel1, Tel2, Formacao, Outros)

- {CPF} → [Nome NomeMae NomePai Endereco NumRG EmissorRG Natural EstCivil Sexo Profissao Email TipoSangue TipoCNH Tel1 Tel2 Formacao Outros]
- {NumRG, EmissorRG} → [NomePai NomeMae Natural DataNasc]

Atividade (Codigo, Nome, Data, Endereco, Descricao)

- {Codigo} → Nome Data Endereco Descricao

Lingua (Codigo, Nome, Dialeto)

- {Codigo} → Nome Dialeto

Fala (*VolCPF*, *CodLing*, Nivel)

- {VolCPF, CodLing} → Nivel

Filial (Codigo, Estado, Endereco)

- {Codigo} → Estado Endereco
- {Estado} → Endereco

Participacao (*VolCPF*, *CodAtiv*, HorasTrab, Descricao)

- {VolCPF, CodAtiv} → HorasTrab Descricao

Funcionario(*CPF*, *CodFilial*, Username, Password)

- {CPF}→ CodFilial Username Password

Avaliação

Primeiramente, é valido enunciar, para evitarmos redundâncias no texto, que os comentários abaixos sobre dependências funcionais consideram que estas são completamente não-triviais.

A tabela **Pessoa** apresenta desnormalização. Ela quebra a propriedade de atributos atômicos da **1ª FN** no atributo **endereco**, que tem valor composto (rua, número, bloco/apartamento, etc). Portanto, descumpre o primeiro requisito para estar nas demais formas. A tabela

As tabelas **Atividade** e **Filial** descumprem também a **1ª FN**, visto que assim como no caso da tabela **Pessoa**, endereço é um valor composto. No caso de **Filial** existe ainda uma dependência transitiva, que constitui outro aspecto de desnormalização.

Lingua, por outro lado, respeita todas as formas normais. Todos seus atributos são atômicos (**1ª FN**), não existem dependências funcionais parciais (**2ª FN**), não existem dependências funcionais transitivas (**3ª FN**) e, por fim, todas as dependências funcionais estabelecem relações apenas entre superchaves e algum outro atributo (**FNBC**).

Fala e **Funcionário**, pelos mesmos motivos de **Lingua**, atende todas as formais normais.

A tabela **Filial**, assim como Voluntario, contém um atributo multivalorado: **endereco**. Portanto, não está na **1ª FN**.

Por fim, a tabela **Participacao** tem um atributo relativamente problemático. O atributo **descricao** contém um texto sobre as tarefas realizadas por um dado voluntário em alguma atividade. De certa forma, consiste em uma agregação de valores semanticamente distintos, o que quebra a **1ª FN**.

Uma maneira de resolver isso, seria modificar descrição para um conjunto de tipos de atividades possíveis, armazenando então quantas horas um candidato dedicou uma certa atividade em um dado evento. Isto contudo tornaria a aplicação mais complicada, fugindo também um pouco da maneira como a organização atualmente armazena tais dados.

Como pode-se ver algumas tabelas não encontram-se normalizadas, tendo um certo destaque a tabela **Pessoa**, que tem um número relativamente alto de atributos. Sua normalização seria possível, mas consideramos que a necessidade de realizar um número mais alto de junções de modo a reagregar os dados de um funcionário poderia ser demasiado custoso.

Por exemplo, consultas ao endereço de um voluntário para determinar se ele mora perto de um evento da ONG podem ser bastante frequentes. A fim de normalizar **Pessoa**, teríamos primeiramente que resolver o problema do atributo **endereco**, criando uma nova tabela que armazena todos os endereços e outra que associa um voluntário ao endereço de sua residência.

Agora, toda vez que quiséssemos consultar onde um voluntário mora, uma junção de três tabelas seria necessária, exigindo uma query SQL muito mais complicada que a utilizada para o caso da tabela totalmente desnormalizada. Mesmo que utilizássemos uma visão para facilitar o acesso ao endereço normalizado, ainda haveria uma perda de eficiência no acesso aos dados e, além disso, sua inclusão também seria mais complicada.

Exemplos de Consultas Gerais

Abaixo utiliza-se a visão Voluntários, cuja declaração encontra-se na seção sobre visões. Abaixo apresento a declaração de Voluntario em álgebra relacional, pois esta também é usada em consultas de tal tipo.

$$T1 \leftarrow \pi_{CPF} Pessoa - \pi_{CPF} Funcionario$$
$$Voluntario \leftarrow Pessoa \bowtie T1$$

1. Mostrar o CPF e o nome de voluntários que falem espanhol com nivel fluente

Álgebra Relacional

$$T1 \leftarrow \sigma_{Nome = 'Espanhol'} \ Lingua$$

$$T2 \leftarrow T1 \bowtie_{Codigo = CodLing} (\sigma_{Nivel = 3} \ Fala)$$

$$T3 \leftarrow Voluntario \bowtie_{CPF = VolCPF} (\pi_{VolCPF} \ T2)$$

$$Resp \leftarrow \pi_{CPF, Nome} T2$$

SQL

```
SELECT v.cpf, v.nome
FROM Voluntario AS  v
INNER JOIN Lingua AS  l
INNER JOIN Fala AS  f
ON l.codigo = f.cod_ling
ON v.cpf = f.cpf
WHERE f.nivel = 'fluente' AND l.nome = 'Espanhol'
```

2. Exibir pelo menos o nome e data das atividades realizadas por voluntárias

Álgebra Relacional

$$T1 \leftarrow \pi_{CPF, Nome}(\sigma_{Sexo = 'F'} Voluntario)$$
$$T2 \leftarrow T1 \bowtie_{CPF = VolCPF} (\delta_{Descricao \rightarrow PartDescr} \ Participacao)$$
$$T3 \leftarrow T2 \bowtie_{CodAtiv = Codigo} (\delta_{Nome \rightarrow NomeAtiv} \ Atividade)$$
$$Resp \leftarrow \pi_{CPF, Nome, CodAtiv, NomeAtiv, Data} \ T3$$

SQL

```
SELECT v.cpf, v.nome, a.codigo, a.nome, a.data
```

```sql
FROM Voluntario AS  v
INNER JOIN Participacao as  p
INNER JOIN Atividade AS  a
ON p.cod_ativ = a.codigo
ON v.cpf = p.cpf
WHERE v.sexo = 'F'
```

3. Exibe a quantidade de voluntários do sexo masculino que não fizeram atividade alguma

Álgebra Relacional

$$T1 \leftarrow \pi_{CPF}\ (\sigma_{Sexo = 'M'} Voluntário)$$

$$T2 \leftarrow T1 \rtimes_{CPF = VolCPF}\ (\delta_{Descricao \rightarrow PartDescr}\ Participacao)$$

$$T3 \leftarrow \sigma_{CodAtiv = null}\ T2$$

$$Resp \leftarrow \vartheta_{count(CPF)\ as\ Qtd}\ (T3)$$

SQL

```sql
SELECT COUNT(*)
FROM Voluntario AS  v
LEFT OUTER JOIN Participacao AS  p
ON v.cpf = p.cpf
WHERE v.sexo = 'M' AND p.cod_ativ IS NULL
```

4. Exibir o voluntário que trabalhou mais horas ao total

Álgebra Relacional

$$T1 \leftarrow Voluntário \bowtie_{CPF = VolCPF} Participacao$$

$$T2 \leftarrow {}_{CPF,\ Nome}\vartheta_{sum(HorasTrab)\ as\ TotalHoras}(T1)$$

$$T3 \leftarrow {}_{CPF,\ Nome}\vartheta_{max(TotalHoras)\ as\ TotalHoras}(T2)$$

$$Resp \leftarrow \pi_{CPF,\ Nome,\ TotalHoras}\ T3$$

5. Mostrar o Nome e CPF de voluntários que não falam inglês

Álgebra Relacional

$$T1 \leftarrow \sigma_{Nome = 'Ingles'} \ Lingua$$

$$T2 \leftarrow T1 \bowtie_{Codigo = CodLing} (Fala)$$

$$T3 \leftarrow Voluntário \bowtie_{CPF = VolCPF} (\pi_{VolCPF} \ T2)$$

$$T4 \leftarrow \pi_{Nome,\ CPF} \ (Voluntário)$$

$$Resp \leftarrow T4 - T3$$

SQL

```sql
SELECT DISTINCT v.cpf, v.nome
FROM Voluntario v
WHERE NOT EXISTS (
    SELECT *
    FROM Lingua l, Fala f
    WHERE f.cpf = v.cpf
      AND l.codigo = f.cod_ling
      AND l.Nome = 'Inglês'
)
```

6. Mostrar o CPF de voluntários que falam inglês e mandarim

```sql
WITH FalaLingua(CPFVol, NomeVol, NomeLing) AS (
  SELECT v.cpf, v.nome, l.nome
  FROM Voluntario AS  v
  INNER JOIN Lingua AS  l
  INNER JOIN Fala AS  f
  ON l.codigo = f.cod_ling
  ON v.cpf = f.cpf
)
((SELECT CPFVol, NomeVol
FROM FalaLingua
WHERE NomeLing = 'Mandarim')
 INTERSECT
(SELECT CPFVol, NomeVol
FROM FalaLingua
WHERE NomeLing = 'Inglês'))
```

7. Mostrar o CPF de voluntários que falam inglês *ou* mandarim

```
WITH FalaLingua(CPFVol, NomeVol, NomeLing) AS (
  SELECT v.cpf, v.nome, l.nome
  FROM Voluntario AS  v
  INNER JOIN Lingua AS  l
  INNER JOIN Fala AS  f
  ON l.codigo = f.cod_ling
  ON v.cpf = f.cpf
)
((SELECT CPFVol, NomeVol
FROM FalaLingua
WHERE NomeLing = 'Mandarim')
  UNION
(SELECT CPFVol, NomeVol
FROM FalaLingua
WHERE NomeLing = 'Inglês'))
```

8. Mostrar o CPF de voluntários que falam inglês, mas não falam mandarim

```
WITH FalaLingua(CPFVol, NomeVol, NomeLing) AS (
  SELECT v.cpf, v.nome, l.nome
  FROM Voluntario AS  v
  INNER JOIN Lingua AS  l
  INNER JOIN Fala AS  f
  ON l.codigo = f.cod_ling
  ON v.cpf = f.cpf
)
((SELECT CPFVol, NomeVol
FROM FalaLingua
WHERE NomeLing = 'Inglês')
  EXCEPT
(SELECT CPFVol, NomeVol
FROM FalaLingua
WHERE NomeLing = 'Mandarim'))
```

9. Mostrar as atividades que contenham a palavra ‘Arrecadação’ em sua descrição

```
SELECT codigo, nome
FROM atividade
WHERE descricao LIKE '%Arrecadação%'
```

10.1 Exibir os voluntários que não participaram de *nenhuma* atividade

```
SELECT v.CPF, v.Nome
```

```
FROM Voluntario AS  v
LEFT OUTER JOIN Participacao AS  p
ON v.CPF = p.cpf
WHERE p.cod_ativ IS NULL
```

10.2 Exibir os voluntários que participaram de *alguma* atividade

```
SELECT v.CPF, v.Nome
FROM Voluntario AS  v
LEFT OUTER JOIN Participacao AS  p
ON v.CPF = p.cpf
WHERE p.cod_ativ IS NOT NULL
```

11. Listar os funcionários que falem inglês de nível médio ou avançado (fluente), por ordem ascendente de domínio da língua

```
WITH FalaLingua(CPFVol, NomeVol, NomeLing, Nivel) AS (
  SELECT v.CPF, v.Nome, l.Nome, f.Nivel
  FROM  Lingua AS  l
  RIGHT OUTER JOIN Fala AS  f
  RIGHT OUTER JOIN Voluntario AS  v
  ON v.CPF = f.cpf
  ON l.Codigo = f.cod_ling
  WHERE v.CPF IS NOT NULL AND  f.Nivel IS NOT NULL
)
SELECT CPFVol, NomeVol
FROM FalaLingua
WHERE NomeLing = 'Inglês'
AND Nivel IN ('medio', 'fluente')
ORDER BY Nivel
```

OU

```
WITH FalaLingua(CPFVol, NomeVol, NomeLing, Nivel) AS (
  SELECT v.cpf, v.nome, l.nome, f.nivel
  FROM Voluntario AS  v
  INNER JOIN Lingua AS  l
  INNER JOIN Fala AS  f
  ON l.codigo = f.cod_ling
  ON v.cpf = f.cpf
)
SELECT CPFVol, NomeVol
FROM FalaLingua
WHERE NomeLing = 'Inglês'
```

```
  AND Nivel = SOME ('medio', 'fluente')
  ORDER BY Nivel
```

12. Listar a média de horas trabalhadas por cada voluntário em ordem decrescente, desde que tenha sido superior a 5 horas

```
SELECT t.CPFVol, t.nome, t.MediaHoras
FROM (
  SELECT v.cpf AS CPFVol, v.nome, AVG(horas_trab) AS MediaHoras
  FROM Participacao AS p
  NATURAL JOIN Voluntario AS v
  GROUP BY v.cpf, v.nome
  HAVING AVG(p.horas_trab) > 5
) AS t
ORDER BY t.MediaHoras DESC
```

OU

```
SELECT t.CPFVol, t.nome, t.MediaHoras
FROM (
  SELECT v.cpf AS CPFVol, v.nome, AVG(horas_trab) AS MediaHoras
  FROM Participacao AS p
  NATURAL JOIN Voluntario AS v
  GROUP BY v.cpf, v.nome
) AS t
WHERE t.MediaHoras > 5
ORDER BY t.MediaHoras DESC
```

13. Listar funcionários que realizaram alguma atividade entre 20 de julho de 2013 e 25 de julho de 2013 ou entre 20 de agosto de 2013 ou 25 de agosto de 2013

```
SELECT v.CPF, v.Nome
FROM Voluntario AS  v
INNER JOIN Participacao AS  p
INNER JOIN Atividade AS  a
ON a.Codigo = p.cod_ativ
ON v.CPF = p.CPF
WHERE a.Data BETWEEN '2013-07-20' AND '2013-07-25'
OR a.Data BETWEEN '2013-08-20' AND '2013-08-25'
```

14. Listar total de atividades organizada por cada filial em cada um dos últimos 12 meses

```
SELECT (lpad(CAST(mes AS char(2)), 2, '0')||'/'||ano) AS periodo,
estado, COUNT(a.codigo) AS totalAtiv
        FROM (
          SELECT
            extract(month from ultimoano.mesano) as mes, extract(year
from ultimoano.mesano) as ano, f.codigo, f.estado
          FROM (
            SELECT CURRENT_DATE - (cast( s.m||' month' as interval))
as mesano FROM generate_series(0,11) as s(m) ORDER BY mesano ASC
          ) as ultimoano FULL OUTER JOIN filial f ON 1=1
        ) AS filialmes LEFT OUTER JOIN atividade a ON
          extract(month from a.data) = filialmes.mes AND
          extract(year from a.data) = filialmes.ano AND
          filialmes.codigo = a.cod_filial
        GROUP BY mes, ano, estado
        ORDER BY estado ASC;
```

15. Conta a quantidade de voluntários em cada faixa de idade (<18, [18, 24], [25, 34], [35, 44], [45, 54], [55, 64], >65)

```
SELECT
  SUM(CASE WHEN idades.idade < 18 THEN 1 ELSE 0 END) AS i18,
  SUM(CASE WHEN idades.idade BETWEEN 18 AND 24 THEN 1 ELSE 0 END) AS i1824,
  SUM(CASE WHEN idades.idade BETWEEN 25 AND 34 THEN 1 ELSE 0 END) AS i2534,
  SUM(CASE WHEN idades.idade BETWEEN 35 AND 44 THEN 1 ELSE 0 END) AS i3544,
  SUM(CASE WHEN idades.idade BETWEEN 45 AND 54 THEN 1 ELSE 0 END) AS i4554,
  SUM(CASE WHEN idades.idade BETWEEN 55 AND 64 THEN 1 ELSE 0 END) AS i5564,
  SUM(CASE WHEN idades.idade > 64 THEN 1 ELSE 0 END) AS i65
FROM (
  SELECT DATE_PART('year', AGE(data_nasc)) as idade FROM voluntario
 ) as idades;
```

16. Obtém voluntários cujo nome tenha 10 caracteres

```
SELECT * FROM voluntario
WHERE nome LIKE '__________'
```

17. Lista os 10 voluntarios com mais horas de participação acumuladas:

```sql
SELECT p1.cpf, SUM(p1.horas_trab) AS totalHoras
FROM participacao p1
GROUP BY p1.cpf
HAVING (
  SELECT COUNT(*)
  FROM (
        SELECT SUM(p2.horas_trab)
        FROM participacao p2
        GROUP BY p2.cpf
        HAVING SUM(p2.horas_trab) > SUM(p1.horas_trab)
  ) as total_maiores
) < 10
ORDER BY totalHoras DESC
```

18. Mostrar os funcionários que podem doar sangue para A+

Álgebra Relacional

T1

TipoSanguíneo
A+
A-
O+
O-

$$Resp \leftarrow Voluntário \div T1$$

Modelagem Física e Scripts DDL

Observações

A criação da tabela **emissores_rg** permite a realização de um controle mais rigoroso do registro de órgãos de emissão de RG de um possível candidato. Ela apresenta-se apenas no modelo físico, por ser apenas uma forma de tornar o sistema mais robusto à entrada de dados inválidos por parte de usuários.

Os índices de pessoa (email) e atividade (data) visam agilizar queries que utilizem estes campos, e o índice unique de funcionário (username) visa, além disso, garantir que não existam nomes de usuários duplicados.

Tipos

```
CREATE TYPE est_civil_type AS
ENUM('solteiro','casado','viuvo','divorciado');
CREATE TYPE tipo_sangue_type AS
ENUM('A+','A-','B+','B-','AB+','AB-','O+','O-');
CREATE TYPE tipo_cnh_type AS ENUM('A','B','C','D','E','AB','AC','AD','AE');
CREATE TYPE sexo_type AS ENUM('M', 'F');
CREATE TYPE formacao_type AS ENUM('fundamental incompleto', 'fundamental
completo', 'médio incompleto', 'médio completo', 'superior
incompleto','superior completo');
CREATE TYPE nivel_ling_type AS ENUM('iniciante', 'medio', 'fluente');
```

Tabelas e Restrições

```
CREATE TABLE emissor_rg (
      sigla char(4) NOT NULL,
      nome varchar(64) NOT NULL,
      CONSTRAINT PK_Emissores_RG PRIMARY KEY (sigla)
);

CREATE TABLE pessoa
(
      cpf numeric(11) NOT NULL,
      nome varchar(128) NOT NULL,
      data_nasc DATE NOT NULL,
      nome_pai varchar(128),
      nome_mae varchar(128),
      endereco varchar(256) NOT NULL,
      num_rg numeric(12) NOT NULL,
      sigla_emissor char(4) NOT NULL,
      naturalidade varchar(32) NOT NULL,
      est_civil est_civil_type NOT NULL,
      sexo sexo_type NOT NULL,
      profissao varchar(32) NOT NULL,
      email varchar(254) NOT NULL,
```

```
	tipo_sangue tipo_sangue_type NOT NULL,
	tipo_cnh tipo_cnh_type,
	tel1 numeric(15) NOT NULL,
	tel2 numeric(15),
	formacao formacao_type NOT NULL,
	obs varchar(1024),
	CONSTRAINT PK_Pessoa_CPF PRIMARY KEY(CPF),
	CONSTRAINT FK_Pessoa_Sigla_Emissor FOREIGN KEY(sigla_emissor)
REFERENCES emissor_rg(sigla) ON DELETE RESTRICT ON UPDATE RESTRICT
);

CREATE TABLE filial
(
  codigo SERIAL NOT NULL,
  estado char(40) NOT NULL CONSTRAINT UNIQ_Filial_Estado UNIQUE,
  endereco varchar(200) NOT NULL,
  CONSTRAINT PK_Filial_Codigo PRIMARY KEY(codigo)
);

CREATE TABLE funcionario
(
	cpf numeric(11) NOT NULL,
	cod_filial INTEGER NOT NULL,
	username char(20) NOT NULL,
	password char(20) NOT NULL,
	CONSTRAINT PK_Funcionario_Matr PRIMARY KEY(cpf),
	CONSTRAINT FK_Funcionario_CPF FOREIGN KEY (cpf) REFERENCES pessoa(cpf)
ON DELETE CASCADE ON UPDATE CASCADE,
	CONSTRAINT FK_Funcionario_Cod_Filial FOREIGN KEY(cod_filial)
REFERENCES filial(codigo) ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE atividade (
	codigo SERIAL NOT NULL,
	cod_filial INTEGER NOT NULL,
	cpf_responsavel numeric(11),
	nome varchar(128) NOT NULL,
	data date,
	endereco varchar(256),
	descricao varchar(256),
	CONSTRAINT PK_Atividade_Codigo PRIMARY KEY(codigo),
	CONSTRAINT FK_Atividade_Cod_Filial FOREIGN KEY(cod_filial) REFERENCES
filial(codigo) ON DELETE RESTRICT ON UPDATE CASCADE,
	CONSTRAINT FK_Atividade_Cod_Responsavel FOREIGN KEY(cpf_responsavel)
REFERENCES pessoa(cpf) ON DELETE SET NULL ON UPDATE CASCADE
);

CREATE TABLE lingua (
	codigo SERIAL NOT NULL,
```

```
	nome varchar(128) NOT NULL,
	dialeto varchar(128) UNIQUE,
	CONSTRAINT PK_Lingua_Codigo PRIMARY KEY(codigo)
);

CREATE TABLE fala (
	cpf NUMERIC(11) NOT NULL,
	cod_ling INTEGER NOT NULL,
	nivel nivel_ling_type NOT NULL,
	CONSTRAINT PK_Fala_PK PRIMARY KEY (cpf, cod_ling),
	CONSTRAINT Fala_CPF_FK FOREIGN KEY (cpf) REFERENCES pessoa(cpf) ON
DELETE CASCADE ON UPDATE CASCADE,
	CONSTRAINT Fala_CodLing_FK FOREIGN KEY(cod_ling) REFERENCES
lingua(codigo) ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE participacao (
	cpf  NUMERIC(11) NOT NULL,
	cod_ativ INTEGER NOT NULL,
	horas_trab SMALLINT NOT NULL,
	descricao varchar(256) NOT NULL,
	CONSTRAINT PK_Participacao_CPF_CodLing PRIMARY KEY(cpf, cod_ativ),
	CONSTRAINT FK_Participacao_CPF FOREIGN KEY(cpf) REFERENCES pessoa(cpf)
ON DELETE CASCADE ON UPDATE CASCADE,
	CONSTRAINT FK_Participacao_CodAtiv FOREIGN KEY(cod_ativ) REFERENCES
Atividade(codigo) ON DELETE RESTRICT ON UPDATE CASCADE,
	CONSTRAINT CK_Participacao_Horas check( horas_trab between 1 and 8 )
);
```

Views

```
CREATE VIEW voluntario AS
SELECT * FROM pessoa p WHERE NOT EXISTS(
	SELECT * FROM funcionario f WHERE f.cpf = p.cpf
)


-- View que facilita a inserção de um funcionário e obtenção de todas as
suas
-- informações cadastrais, e garante a restrição inserção de que este seja
-- maior de idade.
CREATE VIEW funcionario_completo AS
SELECT * FROM pessoa p NATURAL JOIN funcionario f
WHERE age(p.data_nasc) >= '18 years'
WITH CHECK OPTION;
```

Índices

```
CREATE UNIQUE INDEX index_un_username ON funcionario USING btree(username);
CREATE INDEX index_email ON pessoa USING btree(email);
CREATE INDEX index_ativ_data ON atividade USING btree(data);
```

Triggers e Functions

```
CREATE OR REPLACE FUNCTION cpf_validar( ) RETURNS trigger AS $$
  -- ROTINA DE VALIDAÇÃO DE CPF
  DECLARE
  x real;
  y real; --Variável temporária
  soma integer;
  dig1 integer; --Primeiro dígito do CPF
  dig2 integer; --Segundo dígito do CPF
  len integer; -- Tamanho do CPF
  contloop integer; --Contador para loop
  val_par_cpf varchar(11); --Valor do parâmetro
  BEGIN
  -- Teste do tamanho da string de entrada
  val_par_cpf := CAST(NEW.cpf AS varchar(11));
  IF char_length(val_par_cpf) = 11 THEN
  ELSE
  RAISE NOTICE 'Formato inválido: %',$1;
  RETURN NULL;
  END IF;
  -- Inicialização
  x := 0;
  soma := 0;
  dig1 := 0;
  dig2 := 0;
  contloop := 0;
  len := char_length(val_par_cpf);
  x := len -1;
  --Loop de multiplicação - dígito 1
  contloop :=1;
  WHILE contloop <= (len -2) LOOP
  y := CAST(substring(val_par_cpf from contloop for 1) AS NUMERIC);
  soma := soma + ( y * x);
  x := x - 1;
  contloop := contloop +1;
  END LOOP;
  dig1 := 11 - CAST((soma % 11) AS INTEGER);
```

```
  if (dig1 = 10) THEN dig1 :=0 ; END IF;
  if (dig1 = 11) THEN dig1 :=0 ; END IF;

  -- Dígito 2
  x := 11; soma :=0;
  contloop :=1;
  WHILE contloop <= (len -1) LOOP
  soma := soma + CAST((substring(val_par_cpf FROM contloop FOR 1)) AS REAL) *
x;
  x := x - 1;
  contloop := contloop +1;
  END LOOP;
  dig2 := 11 - CAST ((soma % 11) AS INTEGER);
  IF (dig2 = 10) THEN dig2 := 0; END IF;
  IF (dig2 = 11) THEN dig2 := 0; END IF;
  --Teste do CPF
  IF ((dig1 || '' || dig2) = substring(val_par_cpf FROM len-1 FOR 2)) THEN
  RETURN NEW;
  ELSE
  RAISE EXCEPTION 'DV do CPF Inválido: %',NEW.cpf;
  RETURN NULL;
  END IF;
  END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER validacao_CPF BEFORE INSERT OR UPDATE ON pessoa
FOR EACH ROW EXECUTE PROCEDURE cpf_validar( );

CREATE OR REPLACE FUNCTION tels_diferentes( ) RETURNS trigger AS $$
DECLARE
BEGIN
  IF new.tel1 = new.tel2 THEN
    RAISE EXCEPTION 'telefones devem ser diferentes';
    RETURN NULL;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER telefones_diferentes BEFORE INSERT OR UPDATE ON pessoa
FOR EACH ROW EXECUTE PROCEDURE tels_diferentes( );
```

Atualização de Dados

As queries de atualização de dados são parametrizadas segundo os dados recebidos pela aplicação PHP. Assim, mostraremos aqui exemplos de possíveis queries montadas pela aplicação.

As queries nunca são complexas pois os parâmetros são sempre bem definidos. Por exemplo, caso o usuário funcionário queira deletar todos os voluntários naturais da Argentina, ele teria que realizar uma busca por este atributo no web app e, então, deletar um por um.

Exemplo de SQL para atualizar nome de uma atividade.

```
UPDATE atividade
SET nome='Festa Beneficente'
WHERE codigo=22;
```

Exemplo de SQL para inserir uma tupla na tabela Fala.

```
INSERT INTO fala VALUES (
'12345678909',
5,
'fluente');
```

Exemplo de SQL para deletar a participação de um voluntário em uma atividade.

```
DELETE FROM participacao WHERE cod_ativ=3 AND cpf='46476479431';
```

Testes de Restrição

A seguir, mostramos queries que violam restrições do banco e, portanto, mostram erros quando executadas.

Essa query falhará pois o CPF é inválido. Existe uma trigger que valida o CPF de uma pessoa antes da inserção da tupla na tabela.

```
INSERT INTO pessoa VALUES (
11122233300,
'William Bonner',
'1970-05-20',
'Willy',
'Bonna',
'R. Rede Globo',
123456,
'PC',
'Rio de Janeiro',
'casado',
'M',
'Jornalista',
'bonner@globo.com',
'A+',
'B',
```

```
'59998888',
NULL,
'superior completo',
NULL)
```

Já essa inserção vai falhar pois não é possível participar de uma única atividade por mais de 8h, condição que é checada com uma restrição do tipo *check*.

```
INSERT INTO participacao VALUES ('19534157848', 2, 12, 'Socorrista');
```

Este é um exemplo das várias restrições de integridade referencial implementadas no banco. Não há o CPF informado na tabela Pessoa, logo o erro acusado será de violaçao de foreign key constraint.

```
INSERT INTO funcionario VALUES ('99999999900', 1, 'sergio', 'puc123');
```

Também encontraremos erros se tentarmos inserir uma nova tupla que tenha um valor para um atributo chave primária igual ao de alguma outra tupla. O mesmo acontecerá com atributos que tenham restrição de *unique*. No exemplo, tentamos inserir duas filais com atributo 'estado' igual, mas um erro será gerado pois este atributo tem um *unique constraint*.

```
INSERT INTO filial VALUES
(DEFAULT, 'Acre', 'Av. Rio Branco, 222'),
(DEFAULT, 'Acre', 'Rua das Laranjeiras, 50');
```