Universidade de São Paulo. Instituto de Ciências Matemáticas e de Computação SCC0640 - Bases de Dados

Projeto De Monitoramento Urbano Iniciativa Pegue o Pombo

Profa. Dra. Elaine Parros M. de Sousa
Estagiário PAE: André Moreira Souza

Nome:	NUSP:
Che Fan Pan	11200421
Eduardo Cavalari Valença	11234381
Murilo Mussatto	11234245

São Carlos
2021

Conteúdo

1 Parte 1

1.1 Descrição do Problema e dos Requisitos de Dados

A cidade fictícia de Pombópolis é famosa pela grande quantidade de pombos espalhados por sua região. Infelizmente, com o passar dos anos e descaso dos prefeitos, a quantidade de pombos na cidade passou a se tornar um problema para a população pomboliana. Além de serem famosos transmissores de doenças como salmonelose, histoplasmose e criptococose, esses pássaros são conhecidos por deixarem suas fezes por onde quer que passem.

Tendo em vista esse problema, o prefeito recém-eleito, Pombal Columbi, decidiu implementar um sistema de Monitoramento de Pombos para a cidade. Esse sistema teria como principal objetivo controlar a população de pombos e automatizar a limpeza das fezes deixadas por eles. Além disso, o sistema auxiliaria na verificação de possíveis doenças transmitidas pelas aves.

O sistema a ser implementado pelo prefeito teria o auxílio da população da cidade por meio de um sistema distribuído. Essa seria responsável por informar locais onde fezes foram encontradas por meio de um aplicativo de celular. Além disso, o sistema contaria com sensores espalhados pelas regiões da cidade, capazes de identificar a densidade de pombos em um local. Com esses dados em mãos, a prefeitura tomaria as medidas necessárias, contratando zeladores para limpar as fezes, pesquisadores para coletar amostras e analisá-las e Matadores de Pombos para controlar o tamanho da população de pombos em uma determinada região.

1.1.1 Pessoas

O sistema de banco de dados conta com 4 perfis de pessoas físicas. São eles: **Matador de Pombos**, **Zelador**, **Informante** e **Pesquisador**. Essas **Pessoas** são um componente essencial para a resolução do problema. Elas são identificadas por meio de seu atributo chave *CPF*, além de possuírem *nome, data de nascimento* e *endereço*, composto de *CEP*, *Rua,* e *Número*. Utilizando a data de nascimento, é possível calcular também a *idade* da pessoa, como atributo derivado. Ademais, elas são diferenciadas pelo atributo multi-valorado *tipo de pessoa*. Vale ressaltar que uma pessoa pode se encaixar em mais de um perfil, porém, para o sistema, todas as pessoas devem pertencer a pelo menos um desses 4 perfis.

1.1.2 Matador de Pombos

O **Matador de Pombos** é uma especificação de pessoa física, possuindo assim, os mesmos atributos comuns a todas elas. Além disso, ele também possui uma *licença*, dada por um código numérico, para poder exercer sua profissão. O **Matador de Pombos** possui também uma entidade fraca relacionada a ele chamada **Arma**, que possui um *modelo* específico.

Sua principal função é diminuir a população de pombos de um local quando for notificado pela Central de Pombos ao qual está submetido.

1.1.3 Zelador

O **Zelador**, por sua vez, tem como atributo numérico próprio, sua *Carteira de Trabalho*, que pode identificá-lo, além de possuir os atributos herdados por ser uma pessoa física. Sua principal função é limpar um cocô de pombo que foi identificado por um Informante. Os zeladores são avisados por meio de notificações, indicando o local onde se encontra o cocô que deve ser retirado.

1.1.4 Informante

No que diz respeito ao **Informante**, todos possuem pelo menos um **Celular**, porém cada celular possui apenas um dono. O **Celular** tem, em seus atributos, o *Número do Celular* e seu *Modelo*. Utilizando o aparelho, o **Informante** pode informar quando algum Cocô de Pombo é encontrado, guardando a *Data/Hora* de encontro.

O sistema verifica se já existe um cocô, ainda não retirado, no mesmo local. Caso exista, só é armazenada a notificação gerada pelo celular. Caso contrário, um cocô é cadastrado no sistema juntamente com as informações de *Data e Hora* de quando foi encontrado, assim como as suas *Coordenadas*, adquiridas automaticamente por meio do GPS do celular e que podem ser utilizadas para verificar o endereço do *Local*.

1.1.5 Pesquisador

Já o **Pesquisador**, o qual possui uma determinada *formação*, é o responsável por realizar a coleta do cocô de pombo após esse ser identificado. Assim, são geradas **Amostras**, que possuem uma *Data e Hora de Coleta*, um *Peso* e uma *Coloração* específica. O **Pesquisador** é responsável por enviar essas amostras para um **Laboratório** especializado, identificado pelo seu *CNPJ* além de possuir um *Endereço*, composto de *CEP*, *Rua* e *Número*. Após a realização do estudo por parte do **Laboratório** é gerado um **Relatório** para a central de pombos da região, identificado pela chave da amostra, já que essa possui apenas um relatório correspondente. Além disso, o relatório possui as informações da *Data e Hora* de quando foi gerado e se a amostra em estudo caracteriza um *risco iminente* para a população.

1.1.6 Cocô de Pombo

Uma parte central do sistema é a identificação dos **Cocôs de Pombo** pela cidade. Estes são identificados por meio das informações passadas à Central de Pombos por meio de um Informante. Essas informações são: as **coordenadas** de **latitude** e **longitude** de onde foram encontrados, que podem ser obtidas pelo sistema de GPS do celular que os registrou e a *Data e Hora* do encontro. O endereço do local pode ser calculado a partir das coordenadas obtidas.

Além disso, é armazenada a *data e hora de retirada*. Caso este valor seja nulo, o sistema entende que o cocô ainda não foi retirado do local.

1.1.7 Regiões e Sensores

A cidade de Pombópolis é dividida em diversas **Regiões**, cada uma cobrindo um ou mais *Bairros*. Além disso estão dispostos pela região vários **Sensores de Densidade de Pombos**, cada um com seu próprio *Número de Série*. Esses sensores são câmeras capazes de identificar pombos em uma região e contabilizar a sua densidade, criando assim **Dados de Densidades**. Esses dados são entidades fracas, identificados por meio do **Sensor de Densidade de Pombos** de onde foram gerados, assim como a *Data e Hora* de sua criação. Por fim, os dados registram a *densidade* de pombos em número de pombos por metro quadrado.

1.1.8 Centrais de Pombos

As **Centrais de Pombos** são instalações dispostas em cada região para auxiliar no monitoramento dos pombos da cidade. Estas são identificadas por meio do *CNPJ*, único a cada uma das instalações. Além disso, cada uma delas possui seu próprio *endereço*, contendo *CEP, Rua e Número*. Essas centrais são responsáveis por notificar os matadores de pombos, zeladores e pesquisadores para agirem na região.

1.1.9 Notificações

Os **Matadores de Pombos** são notificados por uma **Central de Pombos** de uma região em duas ocasiões. A primeira ocorre quando o valor do atributo da *Densidade de Pombos* de um sensor ultrapassa um certo valor, estipulado pela prefeitura. A segunda ocorre quando um relatório enviado a central mostra que existe risco iminente de doenças transmitidas por pombos naquela região. Assim, é gerada uma **Notificação do Matador de Pombos**, identificada por meio da *Data e Hora de Emissão* de quando foi criada, para que um matador de pombos seja convocado para diminuir a população de pombos dessa região.

Um **Zelador** é notificado no momento em que a **Central de Pombos** de uma região recebe a localização de um **Cocô de Pombo**. É gerado então uma **Notificação Zelador**, única para cada cocô e identificada pela chave do mesmo, que armazena a *Data e Hora de Emissão* de quando foi criada.

Além disso, a localização do cocô é informada junto com essa notificação. Sendo notificado, um Zelador vai a caminho do local para limpar o excremento deixado pela ave e após finalizar, o atributo *Data e Hora Retirado*, da entidade cocô, é modificado.

Um **Pesquisador**, por sua vez, também é notificado no momento em que a **Central de Pombos** de uma região recebe a localização de um **Cocô de Pombo**. É gerado então uma **Notificação Pesquisador**, única para cada cocô e identificada pela chave do mesmo, que armazena a *Data e Hora de Emissão* de quando foi criada. Ademais, a localização do cocô é informada junto com essa notificação. Sendo notificado, um Pesquisador vai a caminho do local para coletar o excremento deixado pela ave, gerando assim uma **Amostra**, que será enviada para um laboratório posteriormente.

1.2 Funcionalidades

1.2.1 Matador de Pombos

- Consulta notificações sobre regiões onde há necessidade de trabalho.
- Consulta histórico de notificações e regiões já atendidas.
- Registra ou remove que irá atender uma notificação.
- Registra notificações já atendidas.
- Registra, modifica ou remove código da licença para trabalho.
- Registra, modifica ou remove modelo da arma.

1.2.2 Zelador

- Consulta notificações sobre cocôs com necessidade de limpeza.
- Consulta histórico dos cocôs limpos.
- Registra ou remove que irá atender um notificação.
- Registra notificações já atendidas.

1.2.3 Informante

- Cadastra um cocô encontrado com data e local.
- Consulta histórico de cocôs registrados.
- Consulta situação (removido ou não) de um cocô registrado.
- Registra, modifica ou remove modelo e número de celular.

1.2.4 Pesquisador

- Consulta notificações sobre cocôs com necessidade de coleta.
- Consulta histórico dos cocôs coletados.
- Registra ou remove que irá atender uma notificação.
- Registra notificações já atendidas
- Registras amostras enviadas para laboratório.
- Consulta situação de envio amostras (concluído ou não)
- Consulta estado de amostras enviadas e seus relatórios (em espera ou concluído).

1.2.5 Laboratório

- Consulta detalhes sobre a amostra recebida.
- Registra a situação da amostra (recebido, em análise, concluído).
- Registra situação do relatório (em espera, concluído, enviado)
- Registra, modifica ou remove cadastro próprio.

1.2.6 Sensor de Densidade de Pombos

- Registra densidade de Pombos da região em determinada hora.

1.2.7 Central de Pombos

- Registra, modifica ou remove notificação para matador de pombos.
- Registra, modifica ou remove notificação para zelador.
- Registra, modifica ou remove notificação para pesquisador.
- Consulta de cocôs da região.
- Consulta situação de remoção de cocôs da região. (removido ou não)
- Consulta situação de notificação para matador de pombos (atendida ou não)
- Consulta dados dos sensores da região.
- Consulta relatórios recebidos.
- Registra, modifica ou remove cadastro próprio.

1.3 Fragilidades do Modelo e Restrições

1. **Possível cadastro duplicado de um cocô em um mesmo local em data/hora diferentes.**

   Caso duas pessoa decidam cadastrar o mesmo cocô em momentos diferentes, haveriam duas instâncias que representam o mesmo cocô no sistema. Isso pode ser resolvido com a filtragem de novas inserções no nível de aplicação. Assim, o segundo cadastro de um cocô que possui a mesma localização de outro já cadastrado, antes desse ser limpo, é impedido pelo sistema.

2. **Uma mesma pessoa não pode cadastrar o mesmo cocô mais de uma vez.**

   O modelo não permite colocar essa restrição, então será necessário colocá-la durante a implementação do banco de dados.

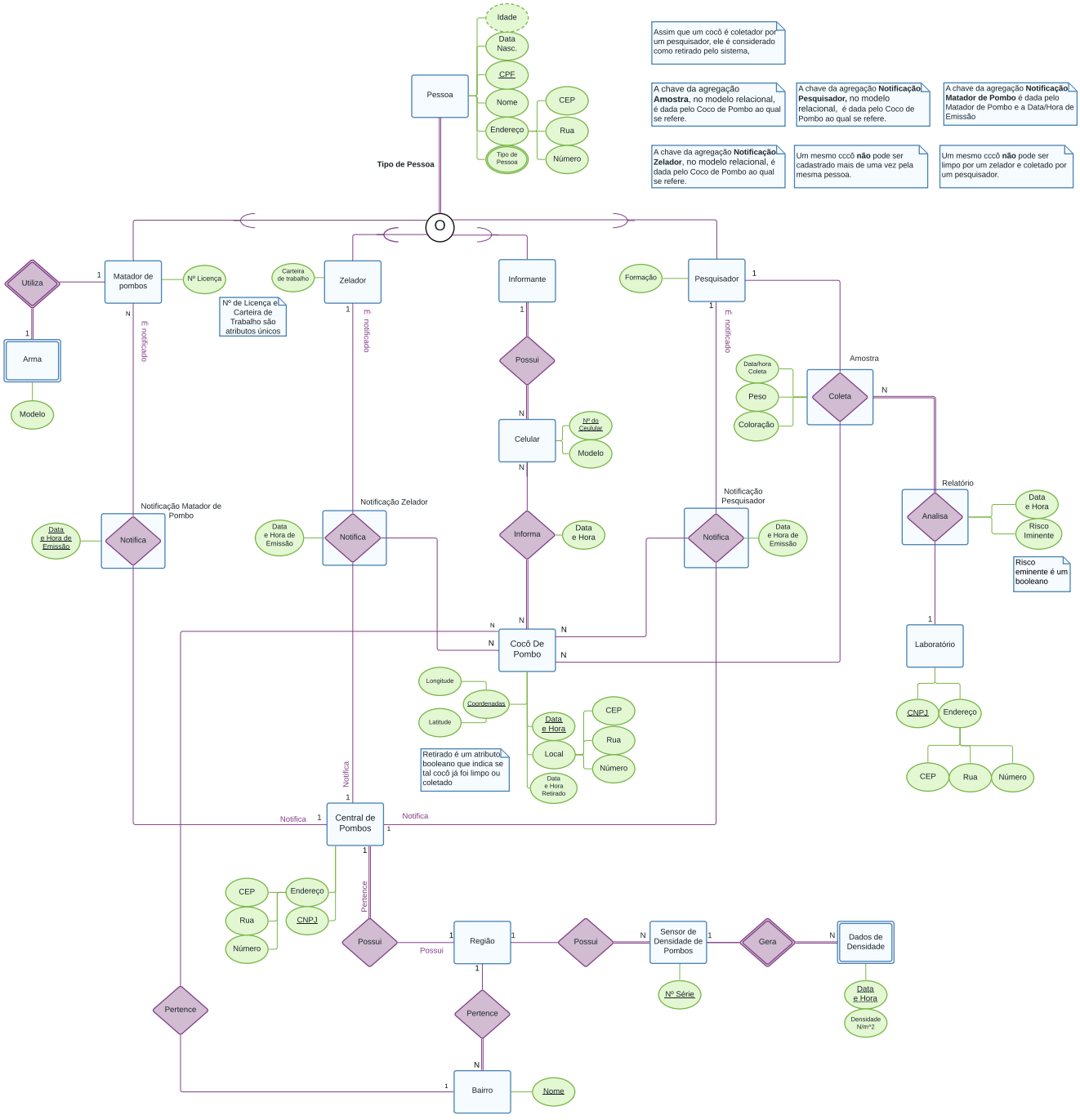
3. **Um mesmo cocô não pode ser limpo por um zelador e coletado por um pesquisador.**

   O modelo também não permite impor essa restrição, assim, será necessário implementá-la no banco de dados.

4. **Ciclo: Cocô de Pombo → Notificação Pesquisador → Pesquisador → Amosta → Cocô de Pombo**

   Apesar de o relacionamento Notifica e Coleta possuírem os mesmos dados em suas tabelas, ambos representam semânticas diferentes. A notificação chega ao pesquisador primeiro, e após isso, ele vai até o local para realizar a coleta do cocô.

1.4 Projeto Conceitual: MER-X



2 Parte 2

2.1 Mudanças no MER

1. **Adição do atributo composto "Coordenadas"no Cocô de Pombo** 1.1.6

Analisando o projeto inicial, ficou claro para o grupo que guardar o endereço completo como chave dessa entidade seria impraticável. Além disso, seria mais condizente com uma situação real, salvar as coordenadas do local desejado e, se preciso, verificar o endereço correspondente.

Assim, o grupo optou por criar esse atributo composto por *"Latitude"* e *"Longitude"*, que são valores obtidos das coordenadas de GPS do celular que informou sobre o cocô. Tal atributo passou a ser chave da entidade, juntamente com a *Data e Hora*. Portanto, o local tornou-se apenas um atributo normal.

2. **Alteração do Atributo Retirado do Cocô de Pombo**

O atributo *"Retirado"* do cocô de pombo foi inicialmente modelado como um booleano que indicaria se o cocô já tivesse sido retirado. Porém, como forma de incluir informações mais pertinentes ao projeto, o grupo optou por modificar esse atributo booleano para a *"Data e Hora Retirado*. Assim, caso este atributo possua valor nulo, o sistema interpreta que o cocô ainda não foi retirado.

Portanto, as informações iniciais foram mantidas e novas informações pertinentes foram incluídas com essa modificação.

3. **Alteração na Multiplicidade da Notificação do Matador** 1.1.9

Anteriormente, a cardinalidade da relação *"Notifica"* entre as entidades *"Matador de Pombos"* e *"Central de Pombos"* era 1:1. Porém, essa cardinalidade implicaria que a central de pombos poderia notificar apenas um matador de pombos, o que contradiz a proposta do trabalho.

Dessa forma, a cardinalidade foi alterada para 1:N, permitindo que uma central notifique vários matadores. Além disso, cada matador continuou podendo ser notificado por apenas uma central, o que está de acordo com o projeto já que cada um deles trabalha para apenas uma das centrais de pombo da cidade.

4. **Alteração na chave da Amostra** 1.1.5

Anteriormente, a agregação *"Amostra"* possuía, além das chaves do pesquisador e do cocô de pombo, um número identificador. Como não era pertinente colocar um número identificador abstrato como chave para a agregação, o grupo decidiu deixá-la sem um atributo chave. Dessa forma, cada par Cocô/Pesquisador formaria apenas uma amostra, que teria como atributos: *"Data e Hora de Coleta"*, *"Peso"* e *"Coloração"*. Sua chave seria formada apenas pelas chave do Cocô de Pombo, já que para o modelo proposto não é possível haver mais de uma coleta para o mesmo cocô.

5. **Criação da Agregação Relatório** 1.1.5

Antes modelado como uma entidade, o grupo optou por transformar o relatório em uma agregação sobre a relação *"Analisa"* entre a agregação *"Amostra"* e a entidade *"Laboratório"*. A chave dessa nova agregação é composta apenas pela chave da amostra. Assim, um par Amostra/Laboratório pode gerar apenas um relatório, o que está condizente com o proposto no projeto.

Além disso, manteve-se a cardinalidade da relação *"Analisa"* de forma que um laboratório pode analisar diversas amostras, mas estas só podem ser analisadas por apenas um laboratório.

6. **Adição do atributo Data/Hora no relacionamento Informa** 1.1.4

Como era de interesse do grupo guardar todas as vezes que um cocô era informado ao sistema por um informante, foi adicionado o atributo *"Data/Hora"* ao relacionamento *"Informa"* entre as entidades *"Celular"* e *"Cocô de Pombo"*.

Assim, gerou-se uma tabela no banco de dados para guardar essas informações, que podem ser usadas para calcular, por exemplo, locais que são repetidamente notificados.

7. **Alteração das chaves das Notificações Zelador e Pesquisador** 1.1.9

Durante a construção do modelo relacional, o grupo percebeu que seria mais conveniente e oportuno, que cada cocô de pombo gerasse apenas uma notificação, tanto para os casos do zelador, quanto para do pesquisador. Assim, o atributo *"Data e Hora de Emissão"*, que antes era chave das duas agregações (*Notificação Zelador* e *Notificação Pesquisador*), deixou de ser chave. Sendo assim, as agregações são identificadas apenas pela chave do cocô, considerando que um cocô não pode ser limpo/coletado mais de uma vez.

As cardinalidades dos relacionamentos forma mantidos.

8. **Adição da Entidade Bairro** 1.1.7

O atributo multivalorado *"Bairro"*, da entidade *"Região"* foi remodelado como um Conjunto de Entidades para melhor atender os requisitos do modelo relacional. Como a ideia inicial do grupo era que uma região poderia possuir vários bairros, porém estes pertenceriam a apenas a uma região, essa restrição não podia ser modelada como atributo multivalorado.

Dessa forma, foi criado um relacionamento *"Pertence"* entre as entidades *"Região"* e *"Bairro"*, com cardinalidade 1:N. Além disso, o atributo *"Nome"* foi criado para identificar os bairros. Por fim, esse relacionamento possui participação total por parte do bairro, impedindo que uma instância exista sem pertencer a uma região.

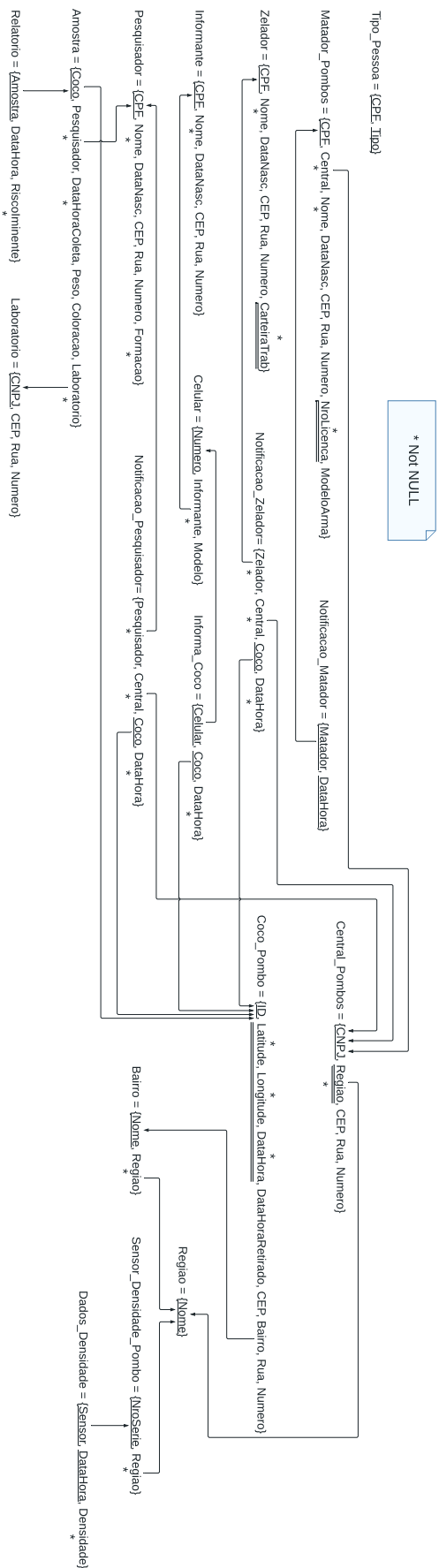
9. **Notação da Generalização da Entidade Pessoa**

Foi incluído o "Tipo de Pessoa"que estava faltando na notação da generalização

10. **Remoção dos Produtos de Limpeza**

Devido a complexidade que o modelo relacional estava tomando durante sua modelagem, o grupo optou por simplificar o MER. A parte do sistema que poderia ser facilmente retirada sem comprometer o restante, era a parte dos *"Produtos de Limpeza"*, relacionados com a entidade *"Central de Pombos"*. Assim, essa parte for retirada do trabalho.

2.2 Projeto Lógico



2.3 Justificativas do Mapeamento

1. **Generalização Pessoa**

Para a escolha do método utilizado no mapeamento da generalização *Pessoa* foram considerados os critérios de relacionamento, quantidade de atributos nulos e a forma como se irá realizar as buscas no banco de dados.

Para começar, temos que no modelo MER a entidade *Pessoa* não realiza qualquer tipo de relacionamento, sendo apenas os seus descendentes que os realizam. Apesar de os descendentes possuírem cada um apenas um atributo específico, com exceção do informante que não o tem, se mapeássemos todas as entidades em uma única relação, teríamos que verificar para cada um a qual tipo de pessoa ele pertence, de acordo com os atributos que estariam nulos ou não. Isso se torna inviável uma vez que cada descendente possui relacionamentos específicos com outras entidades, o que torna a busca com um tempo custoso para encontrar um determinado tipo de pessoa. Outra desvantagem é que a redundância dos atributos genéricos poderia ocasionar possíveis inconsistência nos dados de uma pessoa.

Assim, analisando os critérios, foi decidido mapear cada um dos subtipos junto com seus atributos, pois cada descendente faz diferentes relações. Por fim, foi criado uma tabela para o *Tipo_Pessoa*, a qual permite que uma pessoa seja de mais de um tipo, fazendo coerência à restrição de *Overlap*.

Com esse método, a busca se torna mais eficaz e reduz a quantidade de valores nulos, caso mapeássemos em uma única relação. A desvantagem é a não utilização da chave estrangeira para garantir a consistência dos dados, já que a chave de *Tipo_Pessoa* é composta por CPF e Tipo, devendo ser resolvido no nível de aplicação. Uma outra alternativa de mapeamento seria justamente mapear todos em uma relação, porém com todas as desvantagens já mencionadas.

2. **Entidade Cocô de Pombo**

A entidade Cocô de Pombo tem como chave composta o atributo composto *Coordenadas*, formado por *Latitude* e *Longitude*, e o atributo *DataHora* do registro do cocô. Essa entidade realiza muitos relacionamentos, 4 no total, e, por essa razão, se mantivéssemos no mapeamento esses atributos como chave primária, teríamos que repassar para cada um dos relacionamentos as coordenadas e a data/hora, o que adicionaria para cada tabela dos relacionamentos três dados a mais. Isso faria com que o banco de dados tivesse um espaço de armazenamento ocupado maior.

Assim, foi decidido criar um *ID* sintético para o *Coco_Pombo*, utilizado como chave primária para facilitar a chave estrangeira passada para os outros relacionamentos. A *Latitude*, *Longitude* e *DataHora* foram mantidas como chave composta secundária.

Além disso, na tabela há o atributo *DataRetirado*. Ele representa a data em que um cocô foi limpo e recebe o valor nulo caso ainda não tenha sido retirado. A desvantagem desse mapeamento é a necessidade de verificar o dado de *DataRetirado* a cada inserção de um cadastro de um cocô que está na mesma coordenada de um cocô já registrado. Caso o atributo seja nulo, o cocô não é registrado, caso contrário é realizado o registro.

Uma outra alternativa seria substituir os atributos *DataHora* e *DataRetirado* por um único atributo *Data/Hora_Retirado*, que se tornaria chave composta junto com *Latitude* e *Longitude*. Dessa maneira, quando um cocô fosse registrado num primeiro momento, o atributo receberia o valor nulo. Porém, já que nem todos os bancos de dados aceitam uma chave secundária com valor nulo, isso poderia ocasionar um problema, dependendo do banco utilizado.

3. **Entidade Central de Pombos**

A *Central de Pombos* também possui quatro relacionamentos, porém como sua chave é simples, não houve necessidade de criar um id sintético para diminuir o número de atributos nas tabelas dos relacionamentos. Essa entidade faz relacionamento 1:1 com *Região*, e, portanto, no mapeamento, o *CNPJ* foi utilizado como chave primária e a *Região* como secundária, não podendo ser nulo, já que uma central de pombos deve obrigatoriamente pertencer a uma região. A escolha da *Região* para ser chave secundária foi feita para manter a unicidade do relacionamento, uma vez que cada central de pombos pertence a uma única região, além de manter a participação total da central, já que *Região* não pode ser nulo.

Caso mapeássemos a *Central de Pombos* como chave secundária dentro da tabela Região, por se tratar de um relacionamento parcial por parte de *Região*, haveriam valores nulos para a central, e como nem todo o SGBD aceita valor nulo em chave secundária, necessitaria de uma notação para ser resolvido na aplicação. Assim, o mapeamento escolhido foi o que melhor atendeu aos critérios do modelo e nenhuma outra maneira pareceu mais eficaz.

4. **Entidade Bairro**

Anteriormente, a entidade *Bairro* estava sendo modelada como um atributo multivalorado, pertencente a entidade *Região*. Porém, devido à restrição de que um bairro só pode pertencer a apenas uma região, esse atributo foi remodelado como uma entidade no MER.

Já no modelo relacional, o grupo optou por criar uma tabela para armazenar os bairros, que possuem o nome como chave primária. Além disso, a tabela possui um atributo chamado *"Região"*, que é uma chave estrangeira e não pode ter valor nulo, já que cada bairro deve pertencer a uma região (participação total).

5. **Entidade Fraca Dados de Densidade**

A entidade *Dados de Densidade* é fraca de *Sensor de Densidade de Pombos*, com um relacionamento N:1, respectivamente. A forma de mapeamento foi feita criando-se uma tabela para D*Dados de Densidade* e uma outra para *Sensor de Densidade de Pombos*. Por ser fraca, na tabela de *Dados_Densidade* é inserida o atributo que é chave no *Sensor_Densidade_Pombo*, ou seja, o número de série do sensor. Desse modo, é formado a chave composta de Dados que são: o número de série e a data/hora do registro, fazendo com que a cardinalidade seja consistente ao modelo.

Não foi identificado nenhuma desvantagem desse mapeamento ou uma outra maneira de mapear que seja pertinente.

6. **Entidade Fraca Arma**

Por se tratar de uma entidade fraca com relacionamento 1:1 com o *Matador de Pombos*, o mapeamento foi realizado adicionando o atributo da entidade *Arma* na tabela do *Matador_Pombos*, o qual tem o *CPF* como chave. Assim, é garantido que cada matador tenha apenas uma arma e cada arma pertença a um matador. Por ser entidade fraca com chave simples e de cardinalidade 1:1, não houve necessidade de criação de uma tabela para Arma.

A desvantagem é a impossibilidade de se realizar uma busca para uma arma específica de forma direta, já que não há uma tabela própria, necessitando consultar cada matador para encontrar a arma buscada. Ademais, não foi encontrado outra forma de mapear que atendesse ao modelo.

7. **Agregação Notificação Zelador/Pesquisador**

Para ambas as agregações, temos um relacionamento ternário constituído da *Central de Pombos*, um ator (Zelador/Pesquisador), e um *Coco de Pombo*, possuindo cardinalidade 1:1:N respectivamente. São possíveis duas abordagens para a agregação, primeiramente gerar uma tabela para o relacionamento e posteriormente outra para agregação, entretanto, como nossa agregação não possui atributo próprio, não haveria elementos que diferenciariam as duas tabelas. Além disso, o atributo *DataHora* não tende a se repetir, logo, naturalmente, não é necessário se preocupar com a duplicação dos dados. Assim foi decidido apenas manter uma tabela para a agregação.

Agora para a estruturação da tabela que representaria nossa agregação *Notificação Zelador/-Pesquisador* foi feita de forma que melhor se encaixa no modelo do projeto. Como é previsto que haja apenas uma notificação por Coco de Pombo, foi definido que esse seria utilizado como chave primária da tabela, o qual seria uma chave estrangeira do próprio Coco ao que se refere. Além disso, será guardado a *Central de Pombos* que emitiu a notificação, e o *Zelador/Pesquisador* responsável, ambos sendo chaves estrangeiras e com a marcação *Not NULL* para garantir consistência. Também é armazenado o atributo de *DataHora* da emissão da notificação.

8. **Agregação Notificação Matador de Pombos**

Para a agregação *Notificação do Matador de Pombos*, o grupo optou por criar uma tabela para a notificação, utilizando a chave estrangeira do *CPF do matador* e a *Data/Hora* como chaves.

Como cada Matador de Pombos trabalha obrigatoriamente em uma *Central de Pombos* da cidade, foi incluído um atributo, na tabela do *Matador de Pombos*, que não pode ser nulo. Esse atributo é chamado de *"Central"*.

No MER, não existe uma participação total por parte do matador de pombos no relacionamento notifica, entre ele e a Central de Pombos. Porém, a proposta do grupo era que, apesar de sempre trabalhar para uma central de pombos, poderiam ser cadastrados matadores que ainda não foram notificados. Caso existisse a participação total em questão, isso não seria possível.

9. **Agregação Amostra**

Temos que a agregação *Amostra* é derivada da relação de coleta entre um *Pesquisador* e um *Coco de Pombo* com cardinalidade 1:N. Uma alternativa de modelagem da relação seria o *Coco de Pombo* ter o atributo pesquisador e os atributos únicos da agregação, entretanto, como temos que apenas uma minoria dos cocôs serão utilizados como amostra, acabaria por gerar muito valores nulos na tabela. Assim, concluímos que a melhor alternativa seria gerar uma tabela separada para a *Agregação Amostra*.

Essa tabela contém uma chave estrangeira para o *Coco de Pombo* a qual será usada como chave primária da agregação, já que podemos ter apenas uma coleta por cocô. Além disso armazenamos o *Pesquisador* por chave estrangeira (*Not NULL*), a coloração e o peso do cocô retirado.

Vale comentar que existe mais um atributo na tabela que é uma chave estrangeira para o *Laboratório* (marcado como *Not NULL* devido a participação total), isso foi feito devido ao relacionamento *Analisa* entre *Amostra* e *Laboratório* o qual é N:1 respectivamente. Como cada amostra só possui um laboratório relacionado, achamos válido esse mapeamento.

10. **Agregação Relatório**

Para o mapeamento da agregação *Relatório* é utilizado como chave primária a chave estrangeira de *Amostra*, já que iremos possuir apenas um relatório acerca de uma amostra específica. Além disso temos armazenado a *DataHora* do relatório e um atributo *RiscoIminente* marcado como *Not NULL*, pensando que não haveria motivo para ter um relatório caso ele não apresentasse um resultado.

11. **Relacionamento entre Informante e Celular**

Esse é um relacionamento 1:N com participação total de ambas as entidades. Infelizmente esse é um caso o qual não pode ser completamente mapeado no modelo Relacional, o grupo optou por inserir um atributo informante ,*Not NULL*, por chave estrangeira , dentro da tabela da entidade *Celular*. Vale ressaltar que não seria possível adicionar o celular por chave estrangeira dentro do informante devido ao fato de que o mesmo pode ter vários celulares atrelados.

O ponto negativo da modelagem adotada é que não é possível garantir a participação total do informante, apenas do celular, e esse é um problema que terá de ser tratado em nível de aplicação.

12. **Relacionamento Celular e Coco de Pombo**

Como o relacionamento informa possui cardinalidade N:N optamos por modelá-lo em uma tabela a parte, que contém tanto *Celular* e *Cocô de Pombo* como chaves primárias, além do atributo *DataHora Not NULL*. Vale ressaltar que poderíamos optar por não criar outra tabela e sim levar o cocô como chave estrangeira para a tabela do celular com o objetivo de garantir a participação total, no entanto, esse método acaba por gerar a duplicação de dados, o que é muito negativo para o banco, então não será utilizado.

O ponto negativo do método escolhido é que não é possível garantir a participação total do *Coco de Pombo*, portanto terá que ser tratado em nível de aplicação.

13. **Atributo Derivado Idade**

O atributo derivado *"Idade"*, pertencente ao conjunto de entidades *"Pessoa"*, não é armazenado no banco de dados. Assim, ele não está presente em nenhuma tabela do modelo relacional e deve ser calculado, utilizando a informação de data de nascimento, sempre que necessário. Essa informação não é essencial para o sistema e, por isso, pode ser deixada de fora do banco de dados.

3 Parte 3

3.1 Mudanças nas Partes Anteriores

- O Modelo Relacional (2.2) foi refeito para incluir uma referência (seta) que estava faltando entre o atributo *ID* da tabela **Coco_Pombo** e o atributo *Coco* da tabela **Notificacao_Pesquisador**;
- O item 1 da seção 2.3 foi reescrito para incluir a desvantagem da redundância dos atributos genéricos, que não tinha sido mencionada na Parte 2;
- O nome de uma das tabelas do Modelo Relacional foi alterado de *Tipo_de_Pessoa* para *Tipo_Pessoa*, apenas por ser mais simples e intuitivo.
- Os atributos *NroLicenca* e *CarteiraTrab* passaram a ser chaves secundárias das tabelas **Matador_Pombos** e **Zelador**, respectivamente. Como esses atributos identificam inequivocamente seus respectivos tipos de pessoa, o grupo optou por torná-los chave secundária para refletir essa propriedade.
- O atributo *Nome* da tabela **Bairro** torno-se uma chave estrangeira para o atributo *Bairro* da tabela **Coco_Pombo**. Além disso, a maioria dos atributos *Bairro* presentes em outras tabelas do modelo relacional foram removidos por dois motivos. O primeiro foi a redundância de ter o CEP e ainda armazenar o bairro. O segundo, e mais importante, foi que, para manter a consistência, seria preciso que todos os atributos desse tipo tivessem uma chave estrangeria para o atributo *Nome* da tabela **Bairro**. Além disso, o grupo teria que incluir diversos relacionamentos no MER, o que poluiria ainda mais a visualização do modelo. Portanto, o grupo resolveu deixar o atributo *Bairro* apenas na tabela **Coco_Pombo**, fazendo a devida alteração no MER, e retirá-lo de todas as outras tabelas.

3.2 Implementação do Banco de Dados

O banco de dados foi implementado inteiramente usando o SGBD da "Oracle", juntamente com a plataforma "SQLDeveloper"e utilizando a linguagem "SQL". Todo o banco de dados está dividido em três arquivos: *esquema.sql*, *dados.sql*, *consultas.sql* e *delete.sql*.

3.2.1 Arquivo *esquema.sql*

No arquivo *esquema.sql* estão definidos os comandos de criação das tabelas do banco. É nele onde são criadas todas as tabelas presentes no Modelo Relacional, juntamente com seus atributos, chaves primárias, chaves secundárias e chaves estrangeiras. Uma decisão do grupo durante a construção das tabelas foi utilizar o tipo de dado **"VARCHAR2"** para atributos numéricos que possuem tamanho fixo, como por exemplo o CPF (11 dígitos), CNPJ (14 dígitos), CEP (8 dígitos) e Numero de Celular (11 dígitos). Como o tipo de dado **"NUMBER"** permitia que o valor inserido fosse menor do que o necessário, o grupo decidiu utilizar o **"VARCHAR2"** juntamente com um operador **"CHECK"** para garantir que os dados teriam o tamanho necessário.

Além desse caso, o operador **"CHECK"** foi utilizado em outras três situações:

- Garantir que os tipo de pessoa inseridos na tabela **Tipo_Pessoa** pertencessem a um dos quatro tipos definidos durante o trabalho: matador, informante, zelador ou pesquisador;
- Garantir que o atributo *DataHoraRetirado* da tabela **Coco_Pombo** não fosse menor que o valor do atributo *DataHora* da mesma tabela. Este último idica o momento em que determinado cocô foi reportado ao sistema;
- Garantir que atributos como *Latitude*, *Longitude*, *Peso* e *Densidade* estivessem dentro de uma faixa de valores pré-determinada.

Também estão incluídos nesse arquivo alguns **"TRIGGERS"** que garantem a inserção correta de dados nas tabelas que possuem condição de verificação em outra tabela. Por exemplo, um dos triggers foi criado para verificar que o valor do atributo *CPF* em uma inserção na tabela **Informante** está presente na tabela **Tipo_Pessoa** e com o tipo adequado. Uma breve descrição dos triggers pode ser encontradas nos comentários do arquivo.

Por fim, ainda no arquivo *esquema.sql* estão definidos alguns **"TYPEs"** e **"FUNCITONS"** que são utilizadas para fazer consultas com valores variáveis. Por exemplo, em uma das funções, informa-se o CPF de um Zelador cadastrado e é retornado um histórico com todos as notificações atendidas por aquele zelador em específico. Outras funções e seus devidos comentários podem ser encontrados no conteúdo do arquivo, na seção **"Type Objects"** e **"Functions"** e na descrição das consultas na seção 3.4 deste documento.

3.2.2 Arquivo *dados.sql*

O arquivo *dados.sql* foi criado para popular o banco de dados de forma a permitir a execução das consultas pedidas. Nele estão contidos vários comandos do tipo **"INSERT INTO"**.

Vale ressaltar que esse arquivo foi criado apenas para popular inicialmente o banco e não necessariamente atende algumas restrições da aplicação. Por exemplo, várias entradas na tabela **Coco_Pombo** foram inseridas sem necessariamente terem uma entrada correspondente na tabela **Informa_Coco**. Essa situação não aconteceria num ambiente de simulação mais realista já que a única forma de um cocô ser inserido em sua tabela é se um informante o notificasse, criando uma entrada na tabela **Informa_Coco** juntamente com a entrada da tabela **Coco_Pombo**. Outro exemplo é que a o valor do atributo *DataHoraRetirado* da tabela **Coco_Pombo** não necessariamente corresponde ao valor do atributo *DataHoraColeta* da tabela **Amostra**. O que, novamente, seria impossível de acontecer em um ambiente real já que, no momento em que um pesquisador coletasse o cocô e modificasse o atributo *DataHoraRetirado*, seria criado, necessariamente, uma entrada na tabela **Amostra** com o valor do atributo *DataHoraColeta* correspondente. Dessa forma, não foram criados "checks"ou "triggers"para impedir esses tipos de dados inconsistentes.

Por fim, a inserção dos dados no arquivo está separada por tabelas, para melhor visualização.

3.2.3 Arquivo *consultas.sql*

No arquivo *consultas.sql* encontram-se todas as consultas criadas para exibir dados relevantes sobre o banco de dados. As consultas estão descritas mais detalhadamente na seção 3.4.

É nesse arquivo, também, que se encontram as consultas que utilizam as funções definidas no arquivo *esquema.sql*, juntamente com uma breve descrição sobre a funcionalidade de cada uma e o trecho da função que efetivamente faz a consulta no banco.

3.2.4 Arquivo *delete.sql*

Este arquivo foi adicionado para facilitar a remoção das tabelas. Nele está contido todos os **"DROPS"** para as tabelas, *types* e funções criadas.

3.3 Descrição da Aplicação

Para o desenvolvimento da aplicação foi utilizada a linguagem de programação "Python", juntamente com as bibliotecas: cx_Oracle e python-dotenv. A primeira biblioteca é responsável por fazer a conexão com o banco de dados utilizado, para que fosse possível executar os comandos da aplicação. Já a segunda biblioteca, foi utilizada somente para que os dados referentes a conexão, como login e senha, pudessem ser colocados em um arquivo separado.

O código referente à aplicação encontra-se dentro dos arquivos: *connections.py*, *consulta.py*, *insercao.py*, *main.py* e *printfunctions.py*, sendo necessário executar apenas o arquivo *main.py* para inicio do programa. Como mencionado anteriormente, o SGBD utilizado foi o Oracle, disponibilizado pela professora, com servidores no ICMC.

A Aplicação em si é feita inteiramente no terminal, sem possuir uma interface gráfica. Seu funcionamento ocorre da seguinte maneira. Ao abrir a aplicação, é apresentado ao usuário um menu, onde ele pode escolher as seguintes opções:

1. **Logar/Cadastrar**;
2. **Consultar**;
3. **Finalizar o programa**

3.3.1 Inserção

Caso o usuário opte por fazer uma **Inserção**, ele deverá escolher a opção de **Logar/Cadastrar**. Antes de poder registrar um cocô, o usuário deve estar cadastrado no sistema e após esse passo ele poderá realizar o login, informando apenas o seu CPF.

Na opção **Logar/Cadastrar** aparecerá um novo menu com duas opções:

1. **Logar**

   Essa opção só será realizada com sucesso caso o usuário já tenha cadastrado seu CPF no banco de dados. Se o CPF não for encontrado, uma mensagem de erro será informada ao usuário. O login será realizado assim que for informado um CPF previamente cadastrado.

   Após o login, aparecerá o(s) tipo(s) de pessoa aos quais o CPF informado pertence. Além disso, caso o CPF seja de um informante, será mostrado uma mensagem perguntando se ele deseja cadastrar um cocô. Caso o usuário escolha "sim", novos campos aparecerão para serem preenchidos de acordo com a localização do cocô. Entretanto, se o informante não informou o número do celular no momento do cadastro, ele não conseguirá reportar um cocô.

   Os campos a serem preenchidos ao se reportar um cocô são:

   - *Celular*;
   - *Latitude*, de -90 a 90;
   - *Longitude*, de -180 a 180;
   - *CEP*;
   - *Rua*;
   - *Número*, devendo ser maior que zero;
   - *Bairro*

   Em seguida, será inserida uma entrada na tabela **Coco_Pombo**, com todos os dados fornecidos.

   Por fim, é mostrado uma mensagem informando que o registro do cocô foi realizado com sucesso. Em seguida o usuário volta ao menu principal e poderá ver os dados do registro do cocô na opção de consulta do menu principal.

2. **Cadastrar**

   Ao selecionar essa opção, será pedido ao usuário que insira um valor para os atributos:

   - *CPF* do Informante, contendo 11 dígitos (obrigatório);
   - *Nome* do Informante (obrigatório);
   - *Data de Nascimento*, no modelo DD/MM/YYYY;
   - *CEP*, contendo 8 dígitos;
   - *Rua*;
   - *Número*, devendo ser maior que zero;
   - *Número do celular*;
   - *Modelo do celular*;

   Note que na aplicação, o número do celular está como opcional, mas, em uma situação real, o número seria reconhecido no aplicativo automaticamente.

   Após preencher todos os dados, será inserida uma entrada na tabela **Tipo_Pessoa**, contendo o CPF fornecido e o tipo "informante".

   Em seguida, será inserida uma entrada na tabela **Informante**, com todos os dados fornecidos.

   Caso um número de celular e modelo tenham sido informados, será inserida uma entrada na tabela **Celular** contendo o número e o modelo informados.

   Por fim, encerra-se a operação e o usuário volta ao menu principal.

3.3.2 Consulta

Caso o usuário opte por fazer uma **Consulta**, um novo menu aparecerá com as consultas disponíveis e implementadas na aplicação. O usuário pode então, escolher uma das consultas e o seu resultado será exibido na tela. Há duas categorias de consultas: a geral e a específica.

A **consulta geral** mostra todas as tuplas de uma tabela, com os dados de todos os atributos. São 6 opções de tabelas a serem mostradas, a saber:

1. **Pesquisador**

   Será mostrado o *CPF*, *Nome* do pesquisador, *Data* de Nascimento, *CEP*, *Rua*, *Número* de onde mora e *Formação*.

2. **Informante**

   Será mostrado o *CPF*, *Nome* do informante, *Data* de nascimento, *CEP*, *Rua* e *Número* de onde mora.

3. **Zelador**

   Será mostrado o *CPF*, *Nome* do zelador, *Data* de nascimento, *CEP*, *Rua*, *Número* de onde mora e o *Número* de sua carteira de trabalho.

4. **Matador_pombos**

   Será mostrado o *CPF*, *Nome* do matador de pombos, *Data* de nascimento, *CEP*, *Rua*, *Número* de onde mora, o *Número* de sua licença e o *Modelo* de sua arma.

5. **Central_Pombos**

   Será mostrado o *CNPJ*, *Região*, *CEP*, *Rua* e *Número* de onde se localiza.

6. **Coco_Pombo**

   Será mostrado o *ID* do cocô, *Latitude*, *Longitude*, *Data* do registrado, *Data* de retirada, *CEP*, *Bairro*, *Rua* e *Número* do local de onde foi registrado.

A **consulta específica** mostra algumas das consultas que foram implementadas pelo grupo e necessitam que um valor adicional seja informado antes que o resultado possa ser mostrado. São 5 opções que podem ser escolhidas:

1. **Regiões com Densidade acima de X**

   Será mostrado o *Nome* da região e a *Densidade*, de acordo com o valor estabelecido pelo usuário.

2. **Histórico de Zelador X**

   Será mostrado o *Nome* do zelador, *ID* do cocô referente à notificação, *Data* da notificação e *Data* em que o cocô foi retirado. Caso o valor seja **"None"** para a coluna *Data Retirado*, deduz-se que o cocô ainda não foi retirado.

3. **Histórico de Pesquisador X**

   Será mostrado o *Nome* do pesquisador, *ID* do cocô referente à notificação, *Data* da notificação e *Data* em que o cocô foi coletado. Caso o valor seja **"None"** para a coluna *Data Retirado*, deduz-se que o cocô ainda não foi coletado.

4. **Histórico completo de Pesquisador X**

   Será mostrado o *Nome* do pesquisador, *ID* do cocô referente à notificação, se há *Risco* iminente relacionado ao cocô, *Peso*, *Coloração*, *Laboratório* em que o cocô foi examinado, *Data* da notificação e *Data* em que o cocô foi coletado. Caso o valor seja **"None"** na coluna *Data Retirado*, deduz-se que o cocô ainda não foi coletado.

5. **Histórico de Matador de Pombo X**

   Será mostrado o *Nome* do matador de pombo e a *Data* de suas notificações.

Para a opção 1, o campo pede um valor arbitrário, devendo ser um número real e positivo. Paras as demais opções, o campo deve ser preenchido por um CPF válido, ou seja, que esteja cadastrado no banco de dados e que pertença ao tipo de pessoa da opção de consulta selecionada. Por exemplo, a busca não será bem sucedida se o usuário selecionar a consulta do histórico de um zelador e preencher o campo com um CPF cadastrado de uma pessoa que é apenas do tipo pesquisador.

3.3.3 Finalizar programa

Nesta opção encerra-se a conexão com o servidor da Oracle e a aplicação é finalizada.

3.4 Apresentação das Consultas

1. Essa consulta mostra a média de densidade de pombos por região. São informados o nome da região e a média de densidade de cada região.

Com a tabela **Sensor_Densidade_Pombo** é feito o **"JOIN"** com **Dados_Densidade**. A informação da *região* é retirada da tabela **Sensor_Densidade_Pombo** e a *densidade* da tabela **Dados_Densidade**.

Para calcular a média por região foi utilizada a função agregada **"AVG"** em Densidade, aplicada a cada região, utilizando o operador **"GROUP BY"**.

O resultado da consulta é mostrada em ordem decrescente da média, utilizando o operador **"DESC"**.

```
-- Consulta 1
SELECT SPD.Regiao, AVG(DD.Densidade) AS Densidade_Media
FROM Sensor_Densidade_Pombo SPD
    JOIN Dados_Densidade DD on SPD.NroSerie = DD.Sensor
GROUP BY SPD.Regiao
ORDER BY AVG(DD.Densidade) DESC;
```

2. De maneira similar à **Consulta 1**, a **Consulta 2** mostra o nome da região e a média de densidade de pombos, para regiões que possuam um valor de densidade maior que 1.

Os operadores **"JOIN"** e **"GROUP BY"** são utilizados da mesma maneira que na **Consulta 1**. A diferença é que, para mostrar as regiões que atendem à condição, os grupos de regiões formados pelo **GROUP BY** precisam ter a média de densidade acima de 1. Para verificar essa condição, foi adicionado o comando **"HAVING"**.

O resultado da consulta é mostrada em ordem decrescente da média, utilizando o operador **"DESC"**.

```
-- Consulta 2
SELECT SPD.Regiao, AVG(DD.Densidade) AS Densidade_Media
FROM Sensor_Densidade_Pombo SPD
    JOIN Dados_Densidade DD on SPD.NroSerie = DD.Sensor
GROUP BY SPD.Regiao HAVING AVG(DD.Densidade) > 1
ORDER BY AVG(DD.Densidade) DESC;
```

3. Com o objetivo de conhecer a situação de cada região em relação a pombos que apresentam riscos à saúde da população de Pombópolis, é realizado uma consulta contendo os nomes das regiões e a quantidade de cocôs com risco iminente em cada uma delas.

Primeiramente é feito um **"JOIN"** da tabela **Bairro** com a tablea **Coco_Pombo**, selecionando todos os bairros que tiveram cocôs de pombo registrados. Dessa junção é feito um **"LEFT JOIN"** com a tabela **Relatorio**, pois também é desejado contabilizar os cocôs de uma região que ainda não tiveram o relatório finalizado. O valor vazio, em quantidade de cocô contaminado, retornado pela **Consulta 3**, representa que não temos informações relacionadas àquela região, já que os cocôs dessa região ainda não tiveram os relatórios finalizados.

O resultado da consulta é mostrado em ordem decrescente da quantidade de cocôs com risco iminente, utilizando o operador **"DESC"**.

```
-- Consulta 3
SELECT B.Regiao, SUM(R.RiscoIminente) AS Amostras_Contaminadas
FROM Bairro B
    JOIN Coco_Pombo CP on B.Nome = CP.Bairro
    LEFT JOIN Relatorio R on R.Amostra = CP.ID
GROUP BY B.Regiao
ORDER BY SUM(R.RiscoIminente) DESC;
```

4. Nessa consulta são mostradas as informações sobre o mais recente cocô limpo de um zelador. As informações mostradas são: o nome do zelador, seu CPF, o ID do cocô e a data e hora em que foi retirado.

Com a tabela **Zelador**, foi feito um **"JOIN"** com a tabela **Notificacao_Zelador**, resultando em uma tabela de todos os zeladores que receberam alguma notificação, uma vez que, antes de limpar um cocô, ele deve ter recebido uma notificação . Desse resultado é feito outro **"JOIN"** com a tabela **Coco_Pombo** para que seja possível recuperar a *DataHoraRetirado* do cocô retirado. São utilizadas duas condições, a primeira é de que a *DataHoraRetirado* seja **"NOT NULL"**, pois queremos os dados de um cocô que já foi limpo. Já a segunda condição, é que a *DataHoraRetirado* pertença ao conjunto da data e hora mais recente encontrado na tabela **Coco_Pombo**. Essa última condição foi feita percorrendo a tabela **Coco_Pombo** e procurando a maior *DataHoraRetirado*, utilizando a função agregada **"MAX"**.

Como não temos um atributo na tabela **Coco_Pombo** que indique se esta data e hora foi registrada por um zelador ou pesquisador, é feito um **"JOIN"** com a tabela **Notificacao_Zelador**, garantido que a *DataHoraRetirado* seja referente a um ID de cocô retirado por um zelador.

O resultado da consulta é ordenada pelo nome dos zeladores, em ordem alfabética, utilizando o operador **"ORDER BY"**.

```
-- Consulta 4
SELECT  Z.Nome, NZ.Zelador AS CPF, C.ID AS ID_coco_limpo,
    TO_CHAR(C.DataHoraRetirado, 'DD/MM/YYYY HH24:MI:SS') AS DataHora_retirado
FROM Zelador Z
    JOIN Notificacao_Zelador NZ ON Z.CPF = NZ.Zelador
    JOIN Coco_Pombo C ON NZ.Coco = C.ID
WHERE C.DataHoraRetirado IS NOT NULL AND C.DataHoraRetirado IN (
    SELECT MAX(DataHoraRetirado) FROM Coco_Pombo
        JOIN Notificacao_Zelador ON ID = Coco)
ORDER BY Z.Nome;
```

5. Essa consulta tem a mesma ideia que a **Consulta 4**, porém aplicado aos pesquisadores. Ela mostra o nome do pesquisador, seu CPF, o ID do cocô mais recente coletado por ele e a data e hora da coleta.

A explicação da consulta é exatamente a mesma que da **Consulta 4**. As junções seguem a mesma lógica, trocando-se as tabelas **Zelador** e **Notificacao_Zelador** pelas tabelas **Pesquisador** e **Notificacao_Pesquisador**, respectivamente.

O resultado da consulta é ordenada por nome dos pesquisadores, em ordem alfabética, utilizando o operador **"ORDER BY"**.

```
-- Consulta 5
SELECT  P.Nome, NP.Pesquisador AS CPF, C.ID AS ID_coco_limpo,
    TO_CHAR(C.DataHoraRetirado, 'DD/MM/YYYY HH24:MI:SS') AS DataHora_retirado
FROM Pesquisador P
    JOIN Notificacao_Pesquisador NP ON P.CPF = NP.Pesquisador
    JOIN Coco_Pombo C ON NP.Coco = C.ID
WHERE C.DataHoraRetirado IS NOT NULL AND C.DataHoraRetirado IN (
    SELECT MAX(DataHoraRetirado) FROM Coco_Pombo
        JOIN Notificacao_Pesquisador ON ID = Coco)
ORDER BY P.Nome;
```

6. Para informações detalhadas sobre cada cocô registrado de uma região, a **Consulta 6** mostra o nome da região, o ID do cocô, a data e hora de quando foi retirado e se existe risco iminente ou não. Caso o atributo *Data_Retirado* esteja vazio, significa que o cocô ainda não foi retirado.

A primeira junção é feita com a tabela **Bairro** e a tabela **Coco_Pombo**, para recuperar os bairros dos cocôs registrados. Em seguida, é feito um **"LEFT JOIN"** com a tabela **Relatorio**, para considerar também os cocôs que ainda não tiveram o relatório finalizado. Por último, são organizados grupos de acordo com a região, utilizando o operador **"ORDER BY"**.

```sql
-- Consulta 6
SELECT B.Regiao, CP.ID,
    TO_CHAR(CP.DataHoraRetirado, 'DD/MM/YYYY HH24:MI:SS') AS Data_Retirado,
    R.RiscoIminente
FROM Bairro B
    JOIN Coco_Pombo CP on B.Nome = CP.Bairro
    LEFT JOIN Relatorio R on R.Amostra = CP.ID
ORDER BY B.Regiao;
```

7. Para verificar os serviços que um zelador já realizou, a **Consulta 7** mostra todo o histórico do trabalhador. Mostra-se o seu nome, o ID do cocô, o CNPJ da central, a data e hora da notificação e a data e hora da limpeza do cocô. Caso o valor mostrado no campo *Data_Retirado* esteja vazio, significa que o zelador recebeu a notificação mas ainda não limpou aquele determinado cocô.

Para que as informações do histórico fossem recuperadas, foi feito um **"JOIN"** da tabela **Zelador** com a tabela **Notificacao_Zelador**, pois em **Zelador**, encontra-se o nome da pessoa e, em **Notificacao_Zelador**, o ID do cocô e a data e hora da notificação. Como não é possível saber a data e hora de retirada com apenas essas duas tabelas, é feito outro **"JOIN"** com a tabela **Coco_Pombo** par que fosse possível recuperar essa informação.

O resultado da consulta é mostrado em ordem alfabética dos nomes e por ordem crescente de data e hora, utilizando o comando **"ORDER BY"**.

```sql
-- Consulta 7
SELECT  Z.Nome,NF.Coco,NF.Central,
        TO_CHAR(NF.DataHora, 'DD/MM/YYYY HH24:MI:SS') AS Data_Notificacao,
        TO_CHAR(CP.DataHoraRetirado, 'DD/MM/YYYY HH24:MI:SS') AS Data_Retirado
FROM Zelador Z
    JOIN Notificacao_Zelador NF on Z.CPF = NF.Zelador
    JOIN Coco_Pombo CP  on NF.Coco = CP.ID
ORDER BY Z.Nome,NF.DataHora;
```

Vale ressaltar que foram criadas consultas por CPF para todos os tipos de trabalhadores, no entanto, essas apresentam a mesma estrutura da busca mostrada acima, e por isso serão omitidas do relatório.

8. Essa consulta tem como objetivo identificar a maior parte das características pertinentes aos **Pesquisadores**. Ela mostra, para cada pesquisador, o número de notificações recebidas, o numero de notificações atendidas, o número de amostras coletadas, o número de amostras enviadas para um laboratório e o número de relatórios finalizados.

Vale lembrar que o número de notificações atendidas é o mesmo que de amostras coletadas pois, toda vez que um pesquisador limpa um cocô que foi notificado, ele automaticamente cria uma amostra.

Nessa consulta, foi utilizado o operador **"COUNT"** para contabilizar os números de notificações, amostras e relatórios, da seguinte forma:

- Para as notificações recebidas, contabilizou-se a quantidade de notificações por meio da chave primária da tabela **Notificacao_Pesquisador**
- Para as notificações atendidas, contabilizou-se o número de cocôs que possuem o atributo *DataHoraRetirado*, da tabela **Coco_Pombo**, não nulo.
- Para as amostras coletadas, contabilizou-se simplesmente a quantidade de amostras, utilizando a chave primária da tabela **Amostras**.
- Para as amostras enviadas, contabilizou-se a quantidade de relatórios criados, utilizando a chave primária da tabela **Relatorio**. Isso porque, quando uma amostra é recebida por um laboratório, automaticamente cria-se um relatório para ela.
- Para a quantidade de relatórios finalizados, contabilizou-se a quantidade de atributos *DataHora*, não nulos, da tabela **Relatorio**.

Para a seleção das tabelas, foi necessário utilizar uma combinação dos operadores **"JOIN"**, **"LEFT JOIN"** e **"RIGHT JOIN"**. Apesar de custoso, essa foi a melhor solução achada pelo grupo devido a necessidade de apresentar todo os pesquisadores cadastrados e os casos de exceção. Primeiramente, utilizou-se um **"JOIN"** entre as tabelas **Coco_Pombo** e **Notificacao_Pesquisador** para selecionar apenas os cocôs que estão relacionados com um pesquisador. Em seguida, utilizou-se o operador **"LEFT JOIN"** nas tabela **Amostra**, para manter as notificações que não possuem uma amostra correspondente, e na tabela **Relatorio** para mostrar os casos de exceção onde as amostras foram coletadas mas não foram enviadas a um laboratório e, logo, não tinham um relatório atrelado a elas. Por fim, utilizou-se um **"RIGHT JOIN"** com a tabela **Pesquisador** para mostrar todos os pesquisadores cadastrados, até os que não tinham nenhuma notificação.

Por fim, utilizou-se o operador **"GROUP BY"** para que as contagens fossem agrupadas pelo *Nome* e *CPF* de cada pesquisador. O operador **"ORDER BY"** foi usado apenas para deixar a apresentação dos dados mais amigável ao usuário, mostrando os dados dos pesquisadores em ordem alfabética.

```sql
-- Consulta 8
SELECT  P.Nome AS Pesquisador, P.CPF,
        COUNT (N.Coco) AS N_Notif_Recebidas,
        COUNT (C.DataHoraRetirado) AS N_Notif_Atendidas,
        COUNT (A.Coco) AS N_Amostras_Coletadas,
        COUNT (R.Amostra) AS N_Amostras_Enviadas,
        COUNT (R.DataHora) AS N_Relatorios_Finalizados
FROM Coco_Pombo C
    JOIN Notificacao_Pesquisador N ON C.ID = N.Coco
    LEFT JOIN Amostra A ON A.Coco = C.ID
    LEFT JOIN Relatorio R ON R.Amostra = A.Coco
    RIGHT JOIN Pesquisador P ON P.CPF = N.Pesquisador
GROUP BY P.CPF, P.Nome
ORDER BY P.Nome;
```

9. O objetivo da dessa consulta é verificar os valores obtidos por meio da **Consulta 8**, com respeito as notificações. Ela mostra todas as notificações dos pesquisadores, qual pesquisador está atrelado a cada notificação, a data e hora em que o cocô da notificação foi reportado no sistema, a data e hora de quando a notificação foi criada e a data e hora de quando o cocô foi retirado, ou seja, quando a notificação foi atendida.

Para a seleção das tabelas, foi utilizado um **"JOIN"** da tabela **Coco_Pombo** com a tabela **Notificao_Pesquisador**, para mostrar apenas os cocôs que possuem uma notificação de pesquisador, excluindo assim, aqueles que foram notificados a um zelador. Além disso, utilizou-se um **"RIGHT JOIN"** com a tabela **Pesquisador**, para que pudesse ser mostrado todos os pesquisadores, até aqueles que não possuem nenhuma notificação.

Por fim, o operador **"ORDER BY"** foi usado apenas para deixar a apresentação dos dados mais amigável ao usuário, mostrando os dados dos pesquisadores em ordem alfabética e ordenando, para cada pesquisador, em ordem crescente dos *ID*s dos cocôs.

```sql
-- Consulta 9
SELECT  P.Nome as Pesquisador, C.ID as ID_COCO,
        TO_CHAR(C.DataHora, 'DD/MM/YYYY HH24:MI:SS') as Data_Reportado,
        TO_CHAR(N.DataHora, 'DD/MM/YYYY HH24:MI:SS') as Data_Notificacao,
        TO_CHAR(C.DataHoraRetirado, 'DD/MM/YYYY HH24:MI:SS')
FROM COCO_POMBO C
    JOIN NOTIFICACAO_PESQUISADOR N on N.Coco = C.ID
    RIGHT JOIN PESQUISADOR P on N.Pesquisador = P.CPF
ORDER BY P.Nome, C.ID;
```

10. De maneira similar à **Consulta 9**, essa consulta tem o objetivo de verificar as informações mostradas pela **Consulta 8**, porém ao que se refere às amostras e relatórios. Ela mostra, para cada pesquisador, as notificações que recebeu, juntamente com a sua data e hora, além da data de quando a notificação foi atendida e de quando a amostra foi coletada. O ID da amostra e a data em que o relatório foi finalizado também são mostrados.

Utilizando as informações dessa tabela é possível deduzir diversas afirmações verificando se existe um valor nulo nos campos. Esas dedução é feita da seguinte forma:

- Caso os campos *ID_Coco* e *Data_Notificacao* estiverem com valor nulo, significa que aquele pesquisador não possui nenhuma notificação.
- Se o campo *Data_Coleta* estiver com valor nulo, mas o campo *Data_Notificacao* não estiver, deduz-se que o pesquisador recebeu uma notificação porém ainda não coletou o cocô.
- Caso o campo *Data_Coleta* estiver preenchido, mas o campo *ID_Relatorio* estiver nulo, deduz-se que o pesquisador coletou a amostra mas não a enviou ao laboratório.
- Se o campo *ID_Relatorio* estiver preenchido, mas o campo *Data_Relatório* estiver nulo, deduz-se que o pesquisador enviou a amostra para o laboratório mas o relatório não foi finalizado.

Para selecionar as tabelas, utilizou-se o operador **"LEFT JOIN"** e a justificativa é similar a da **Consulta 8**. Para que fosse possível mostrar todos os casos de exceção, como por exemplo pesquisadores sem notificação ou amostras sem relatório, foi preciso juntar todas as tabelas utilizando o operador em questão.

Por fim, o operador **"ORDER BY"** foi usado apenas para deixar a apresentação dos dados mais amigável ao usuário, mostrando os dados dos pesquisadores em ordem alfabética.

```
-- Consulta 10
SELECT  P.Nome AS Pesquisador, N.Coco AS ID_COCO,
        TO_CHAR(N.DataHora, 'DD/MM/YYYY HH24:MI:SS') AS Data_Notificacao,
        TO_CHAR(A.DataHoraColeta, 'DD/MM/YYYY HH24:MI:SS') AS Data_Coleta,
        R.Amostra AS ID_Relatorio,
        TO_CHAR(R.DataHora, 'DD/MM/YYYY HH24:MI:SS') AS DATA_RELATORIO
FROM Pesquisador P
    LEFT JOIN Notificacao_Pesquisador N ON N.Pesquisador = P.CPF
    LEFT JOIN Amostra A ON A.Pesquisador = P.CPF AND N.COCO = A.COCO
    LEFT JOIN Relatorio R ON R.Amostra = A.Coco
ORDER BY P.Nome;
```

11. Essa consulta mostra valores pertinentes para os Informantes, como o número de cocôs reportados e quantos deles já foram retirados.

O operador **"COUNT"** foi utilizado para contabilizar esses números. Para os cocôs reportados, contabilizou-se o número de entradas na tabela **Informa_Coco**, utilizando-se sua chave primária. Para o número de cocôs retirados, contabilizou-se a quantidade de campos do atributo *DataHoraRetirado*, não nulos, da tabela **Coco_Pombo**.

Para a seleção das tabelas utilizou-se o operador **"JOIN"** para juntar as entradas da tabela **Informa_Coco** com a tabela **Celular** já que cada notificação foi feita por pelo menos um celular. Em seguida, utilizou-se o operador **"RIGHT JOIN"** para que fosse possível selecionar todos os informantes, até aqueles que não possuem um celular cadastrado ou que nunca criaram uma notificação. Por fim, utilizou-se o operador **"LEFT JOIN"** para relacionar as tabelas com os cocôs cadastrados na tabela **Coco_Pombo**, mantendo os casos de exceção mencionados.

O operador **"GROUP BY"** foi utilizado para fazer com que cada contagem fosse referente a um informante em particular, assim podemos pegar os números pertinentes a cada informante.

Por fim, o operador **"ORDER BY"** foi usado apenas para deixar a apresentação dos dados mais amigável ao usuário, mostrando os dados dos informantes em ordem alfabética.

```
-- Consulta 11
SELECT  I.Nome AS Informante,I.CPF,
        COUNT (IC.Coco) AS Cocos_Reportadors,
        COUNT (C.DataHoraRetirado) AS Cocos_Retirados
FROM INFORMA_COCO IC
    JOIN CELULAR Cel ON IC.Celular = Cel.Numero
    RIGHT JOIN Informante I ON I.CPF = Cel.Informante
    LEFT JOIN Coco_Pombo C ON C.ID = IC.Coco
GROUP BY I.CPF, I.Nome
ORDER By I.Nome;
```

12. Essa consulta tem o objetivo de verificar as informações mostradas pela **Consulta 11**. Ela mostra todos os informantes cadastrados, seus números de celular, seus cocôs notificados, a data de quando a notificação foi criada e a data em que o cocô foi retirado.

Utilizando as informações dessa tabela é possível deduzir diversas afirmações verificando se existe um valor nulo nos campos. Essa dedução é feita da seguinte maneira:

- Caso o campo *Celular* esteja com valor nulo, deduz-se que aquele informante não possui nenhum celular cadastrado no sistema;
- Caso o campo *Celular* estiver preenchido, mas o campo *ID_Coco* estiver nulo, deduz-se que o informante não cadastrou nenhum cocô utilizando aquele número de celular.
- Caso os campos *ID_Coco* e *Data_Notificacao* estiverem preenchidos, mas o campo *Data_Retirado* estiver com valor nulo, deduz-se que o aquele cocô foi notificado, porém ainda não foi retirado por um zelador ou pesquisador.

```
-- Consulta 12
SELECT  I.Nome AS Informante, Cel.Numero , IC.Coco AS ID_COCO,
        TO_CHAR(IC.DataHora, 'DD/MM/YYYY HH24:MI:SS') AS Data_Notificacao,
        TO_CHAR(C.DataHoraRetirado, 'DD/MM/YYYY HH24:MI:SS') AS Data_Retirado
FROM Informante I
    LEFT JOIN CELULAR Cel ON I.CPF = Cel.Informante
    LEFT JOIN Informa_Coco IC ON Cel.Numero = IC.Celular
    LEFT JOIN Coco_Pombo C ON C.ID = IC.Coco
ORDER BY I.Nome;
```

13. Essa consulta implementa a **DIVISÃO** como pedido nas especificação do projeto. Ela mostra para o usuário o nome e o CPF de todas as pessoas cadastradas no banco de dados que pertencem ao tipo Informante, Matador de Pombos, Zelador e Pesquisador simultaneamente.

Como a consulta retorna informações de uma pessoa que pertence aos 4 tipos, então poderíamos utilizar o operador **"SELECT"** em qualquer uma das tabelas referentes a esses quatro (**Informante**, **Matador_Pombos**, **Zelador** e **Pesquisador**), uma vez que é garantido que o nome da pessoa estará em todas elas. Escolheu-se a tabela **Pesquisador** por arbitrariedade e por considerar que menos pessoas pertençam a esse tipo, logo, o número de tuplas da tabela é menor, fazendo com que o tempo de execução seja menor.

A divisão foi implementada com consultas aninhadas, consultando externamente o nome e CPF em **Pesquisador**. Na condição foi feita a consulta **"DISTINCT"** na tabela **Tipo_Pessoa** para que retornasse todos os 4 tipos possíveis que uma pessoa pode ser. Com o retorno desta consulta é feito a operação do tipo **"MINUS"** com o retorno da consulta de tipo na tabela **Tipo_Pessoa** novamente, onde é consultado os tipos pertencentes a um CPF de pesquisador. Esta última consulta varre na tabela **Tipo_Pessoa** procurando todos os tipos que pertencem a um mesmo CPF para cada CPF de Pesquisador. O resultado dessa operação no conjunto é vazio caso o CPF pertença aos quatro tipos, fazendo com que a operação **"NOT EXISTS"** seja verdadeira e assim seja mostrado o nome e CPF da pessoa que pertence a todos os tipos.

```
-- Consulta 13
SELECT P.Nome, CPF FROM Pesquisador P WHERE
    NOT EXISTS ( (SELECT DISTINCT UPPER(Tipo) FROM Tipo_Pessoa)
                  MINUS
                 (SELECT UPPER(Tipo) FROM Tipo_Pessoa WHERE P.CPF = CPF)
               );
```

14. Essa consulta também implementa a **DIVISÃO** pedida. Ela mostra o nome e o CPF dos zeladores que trabalharam para todas as centrais registradas no banco de dados.

A lógica de implementação é a mesma que a da **Consulta 13**. É feito o aninhamento de consultas, sendo a consulta externa referente ao Nome e CPF da tabela **Zelador**. Na condição é feito primeiramente a consulta de todos os CNPJ's das centrais de pombos. Depois com o resultado desta consulta, é feito a operação **"MINUS"** com o retorno da consulta de *Central* em **Notificacao_Zelador**, procurando todas as *Centrais* que um mesmo CPF de zelador atendeu.

Dessa operação, o retorno é vazio para o caso onde o zelador tenha trabalhado para todas as centrais, fazendo o operador **"NOT EXISTS"** retornar verdadeiro. Assim, é mostrado o nome e CPF do zelador que trabalhou em todas as centrais cadastradas.

```
-- Consulta 14
SELECT Z.Nome, CPF FROM Zelador Z WHERE
    NOT EXISTS ((SELECT CNPJ FROM Central_Pombos)
                MINUS
                (SELECT Central FROM Notificacao_Zelador WHERE Zelador = Z.CPF)
                );
```

3.5 Conclusão

O projeto em si foi um grande desafio para nosso grupo ao decorrer do semestre. A começar pelo seu tema que foi bem amplo e dava a possibilidade de muitas implementações. Nosso grupo decidiu utilizar a criatividade e criar uma aplicação que não necessariamente seria útil no mundo real, mas que ainda assim mostrasse a importância de se ter um banco de dados bem construído e de se usar um SGBD para gerenciá-lo.

Após termos definido o nosso tema, o próximo desafio, proposto pela Parte 1, foi gerar as definições do nosso banco de dados. Tivemos diversas ideias durante nossas reuniões, sempre tentando manter o projeto criativo, mas possível. Desenhar um Modelo Entidade Relacionamento que atendesse a maior parte de nossas ideias, mas que também fosse possível de ser implementado com o tempo que tínhamos, foi outro grande desafio dessa parte. O modelo foi modificado várias vezes durante o trabalho para melhor acomodar as mudanças e problemas que fomos descobrindo durante sua implementação.

Já na parte 2, o grande desafio foi transformar nosso MER em um Modelo Relacional que fizesse sentido e que não tivesse um número muito grande de tabelas. Novamente, tivemos que modificar nosso MER para se encaixar no Modelo Relacional que construímos. As aulas da disciplina foram de grande ajuda nessa parte já que elas continham um passo-a-passo bem detalhado de como transformar as Entidades e os Relacionamentos do MER para as tabelas do Modelo Relacional.

A parte 3 foi de longe a mais divertida de se fazer. Finalmente implementar as tabelas que criamos e populá-las com dados, trouxe uma enorme satisfação. As mudanças no Modelo ER não escaparam dessa parte e tivemos que adicionar alguns relacionamentos que não tínhamos previsto. Além disso, escrever uma aplicação que implementasse uma pequena parte dessa situação problema irreal que criamos, também foi muito satisfatório.

Por fim, concluímos que o trabalho foi um ótimo meio de solidificar os diferentes conhecimentos que aprendemos durante as aulas da disciplina, já que ele cobriu a maior parte do conteúdo. Os diversos problemas encontrados durante sua implementação e as discussões que tivemos como grupo, realmente ajudaram a entender o que estávamos fazendo e a importância que os bancos de dados tem em um mundo tão conectado quanto o nosso.