# Assignment Brief Database Design

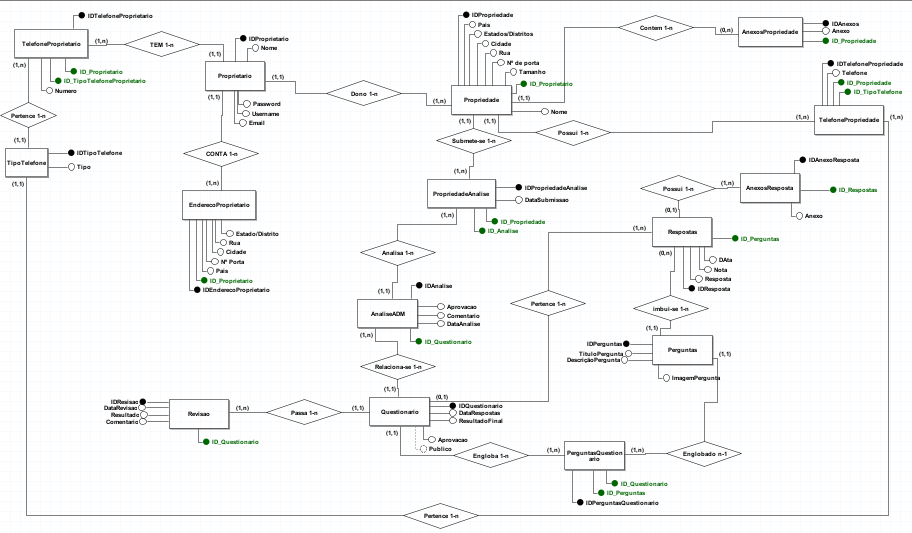
## Trabalho Realizado por Daniel Brendo Franco Turma: Computer Science

## Introdução

No âmbito da disciplina de Database Design, foi-me proposto um cenário em que sou escolhido pela “CAS” (CertAuth Services) para a implementação de uma plataforma que pretende disponibilizar uma lista pública de propriedades e a sua avaliação em termos de “pegada ecológica”.

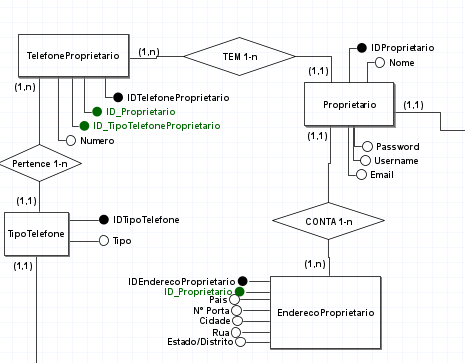
## TASK 1

Para a primeira parte deste projeto é necessário criar em UML um diagrama Entidade-Relação da base de dados que pretendemos construir. Desta forma decidi utilizar o programa BrModelo.jar, um programa java, gratuito e voltado para a criação de deste tipo de diagramas. O Resultado foi o seguinte:



Neste modelo os quadrados representam as entidades ou tabelas, os losangos são as relações entre as entidades e as esferas são os atributos das entidades. Temos ainda duas entidades (esferas) preenchidas, as pretas representam as chaves primarias de cada entidade, já as verdes são das chaves estrangeiras.

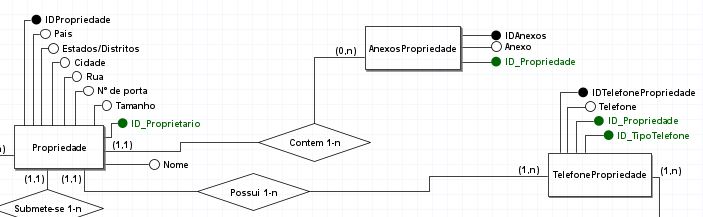
### Proprietário:



A razão pela qual a tabela “telefoneProprietario” não está “dentro” da tabela “proprietario” é porque um proprietário pode possuir mais do que um número de telefone. Desta forma separei a entidade “proprietário” de “telefoneProprietario” criando uma relação de “1-n” entre os dois (um proprietário pode possuir “n” número de telefones).

O mesmo se aplica á tabela “endereçoProprietario”, pois, um proprietário pode possuir mais do que um endereço. Desta forma separamos a entidade “proprietário” de “endereçoProprietario” criando uma relação de “1-n” entre os dois (um proprietário pode possuir “n” endereços).

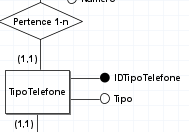
### Propriedade:



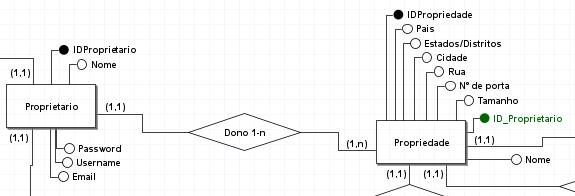
Para a propriedade aplica-se os mesmos princípios que aplicamos para o proprietário no que diz respeito à entidade “TelefonePropriedade”, o motivo de eu ter adicionado um “TelefonePropriedade” foi porque é possível que seja necessário ligar diretamente para uma determinada propriedade onde o proprietário está ausente, ou até mesmo ligar para um determinado sector específico da propriedade. Adicionei também uma tabela “AnexosPropriedade” onde o proprietário poderá acrescentar anexos da sua propriedade para a CAS. Porém como podemos ver na imagem acima, a obrigatoriedade deste anexo é 0 e a cardinalidade é n, isto significa que os anexos não são obrigatórios e que uma propriedade poderá ter múltiplos anexos.

### Tipo Telefone:

Também foi criado a tabela “tipoTelefone” pois, um telefone pode ser: comercial, fixo, móvel, etc...  
Esta tabela relaciona-se tanto com a tabela “TelefonePropriedade”, quando com a tabela “TelefoneProprietario”.



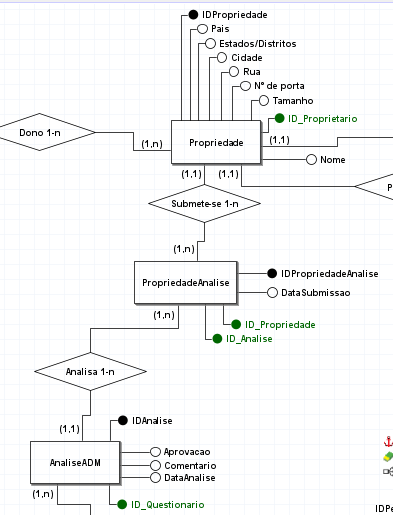
### Propriedade – Proprietário:



Também podemos observar que cada losango possui uma palavra diferente que caracteriza a relação entre as entidades, isto foi-nos ensinado como sendo uma boa pratica na construção de modelos entidade-relação, e para complementar decidi ainda colocar o tipo de relação que o losango representa (1-1/1-n/n-n).

No caso acima podemos ver a relação entre Proprietário e Propriedade, onde 1 único proprietário pode registrar várias propriedades, sendo assim uma relação 1-n.

### Analise:



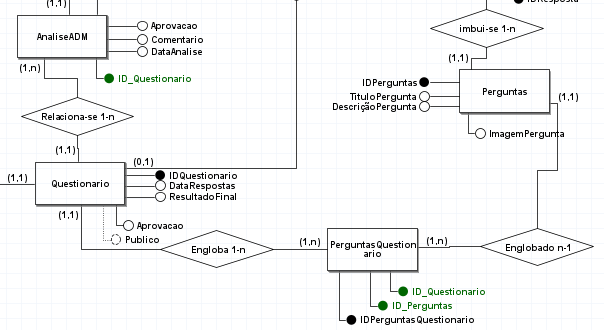
Quando o proprietário submete a sua propriedade na plataforma da CAS, esta passa por uma análise de um administrador, que pode aprovar ou reprovar a propriedade.

Deste modo, foi criada uma entidade chamada “AnaliseADM” (ou analise na base de dados final), que possui uma relação de “n-n” com a Entidade “Propriedade”, pois uma propriedade pode passar por várias analises, assim como também são analisadas diversas propriedades. Para evitar esse problema, foi criado uma terceira entidade chamada “PropriedadeAnalise”, que Além de solucionar o problema mencionado anteriormente, ainda regista as datas em que a propriedade foi submetida para analise.

Quando a propriedade é analisada guardamos a data da análise na tabela “AnaliseADM”, e é através da comparação entre a data de submissão e a data de analise que podemos filtrar quais propriedades falta analisar.

Vale realçar que assim que a propriedade é registrada ela automaticamente é submetida para analise pois é gravado a data de registro da propriedade no atributo “DataSubmissao” e é criado um ID nas tabelas “AnaliseADM” e “Questionario”, sendo que o perito e o Admin apenas terão que preencher os campos em falta de cada uma destas tabelas utilizando Stored Procedures que irão facilitar e tornar tudo mais “user friendly”.

### Questionário:

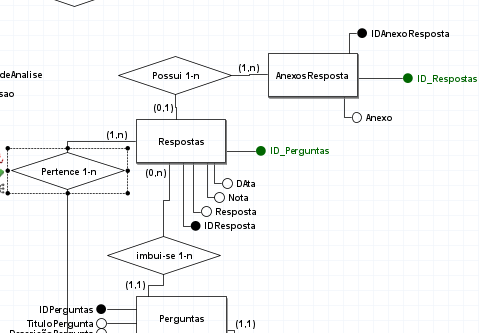


Após a aprovação da análise, é enviado um questionário para o proprietário com diversas perguntas. Cada pergunta pode ter vários anexos, e da mesma forma as respostas do utilizador também poderão ter vários anexos.

Sabemos que um questionário possui várias perguntas, porém o perito também poderá utilizar as mesmas perguntas em questionários diferentes, por esse motivo foi criado a tabela “PerguntasQuestionario”, que permite esta relação acontecer.

A tabela “Perguntas” é a única que possui os anexos como uma entidade e não numa tabela a parte. Isto porque no problema apresentado diz “Como perito, quero ser capaz de adicionar **uma** imagem a **uma** questão”, ou seja, seria desnecessário criar uma nova entidade para ser usada numa relação de 1-1. Desta forma a própria base de dados fica mais limpa.

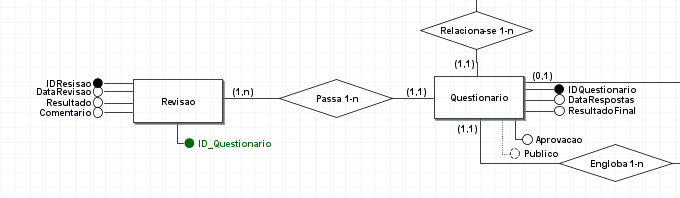
### Respostas:



Na tabela respostas é onde ficam armazenadas as notas, a data de cada resposta e a resposta em si. Uma questão que surgiu durante o desenvolvimento da Base de Dados é como saberíamos a qual questionário e a qual propriedade uma determinada resposta se refere, visto que uma mesma pergunta pode ser utilizada em vários questionários. Para resolver este problema decidi criar uma relação entre a entidade respostas e o Questionário. O que nos permite identificar a qual questionário a resposta se refere.

Um ultimo detalhe é que todos os anexos são LONGBLOB o que permite ao utilizador carregar arquivos de até 4GB.

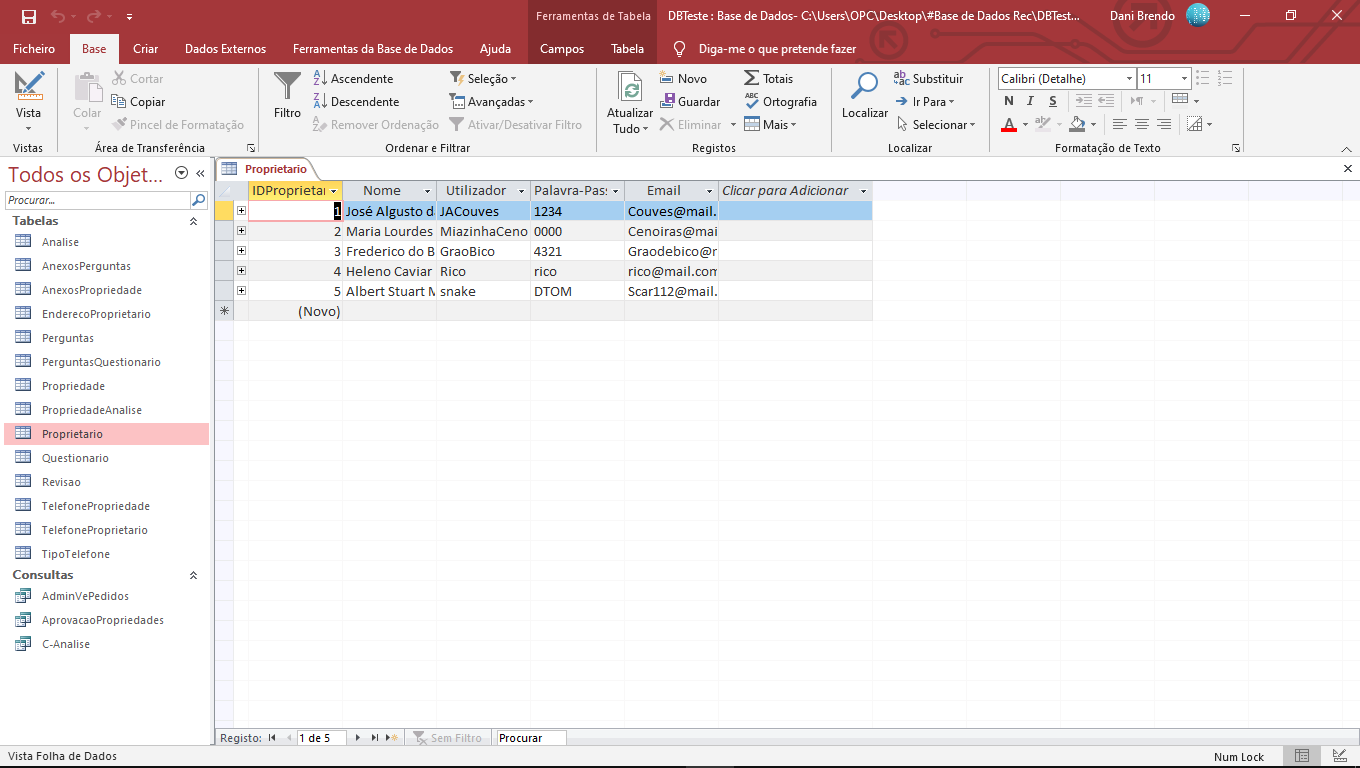
### Revisão:



O questionário pode passar por “n” revisões, para perceber quais questionários faltam ser revistos é utilizado novamente a comparação de datas entre a DataRevisão e DataRespostas, caso a Revisão seja mais antiga que a última resposta o questionário reaparecera para ser avaliado. Cada vez que o questionário for reavaliado uma nova revisão será feita, desta forma será possível consultar um histórico de todas as revisões feitas previamente, o resultado final guardado na tabela questionário será sempre o da última revisão. Após a aprovação final, o proprietário poderá escolher colocar a propriedade pública ou privada. Isso ficará registado no atributo “público” da entidade “questionário”.

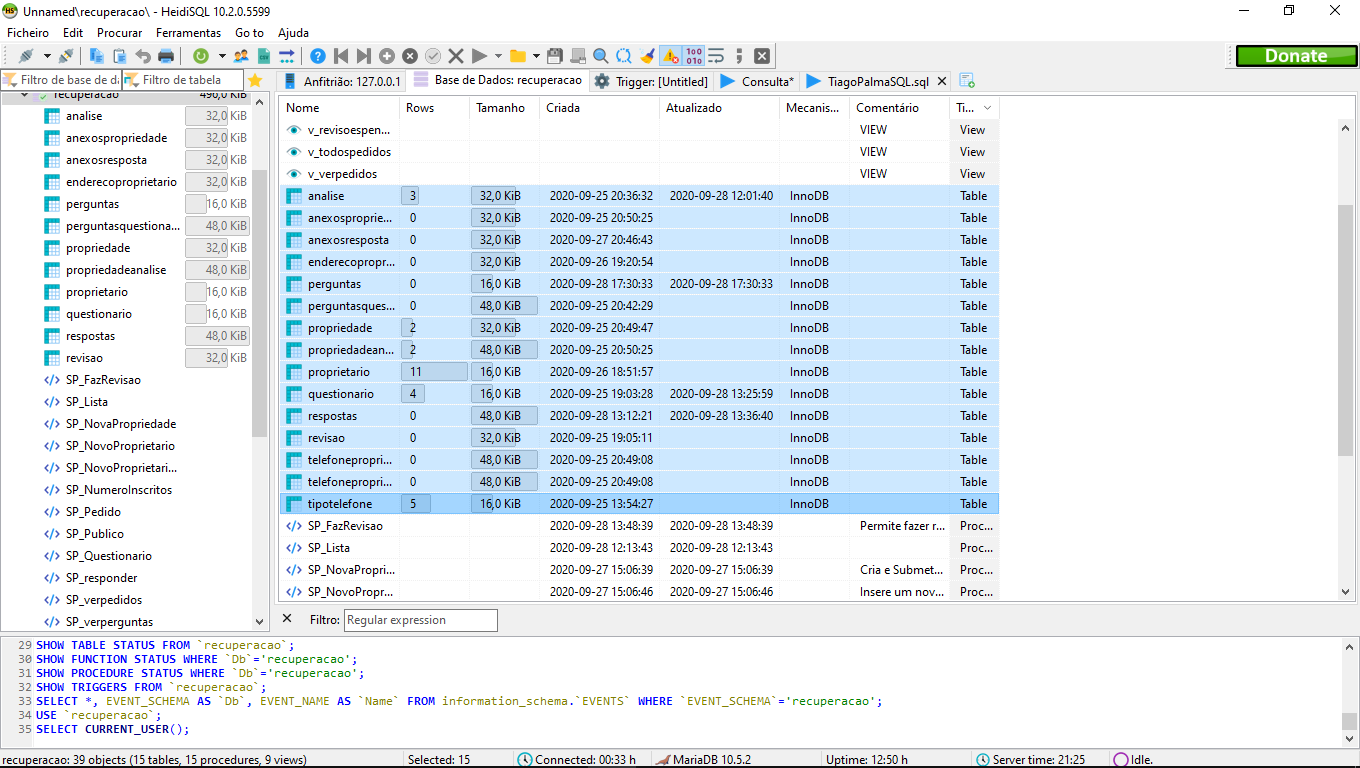
## TAKS 2

Antes de iniciar a implementação da Base de Dados em MySql, decidi criar primeiramente uma base de dados de testes no Access, pois mesmo com as suas limitações é um programa bastante visual que nos permite criar de forma rápida e simples uma base de dados local.

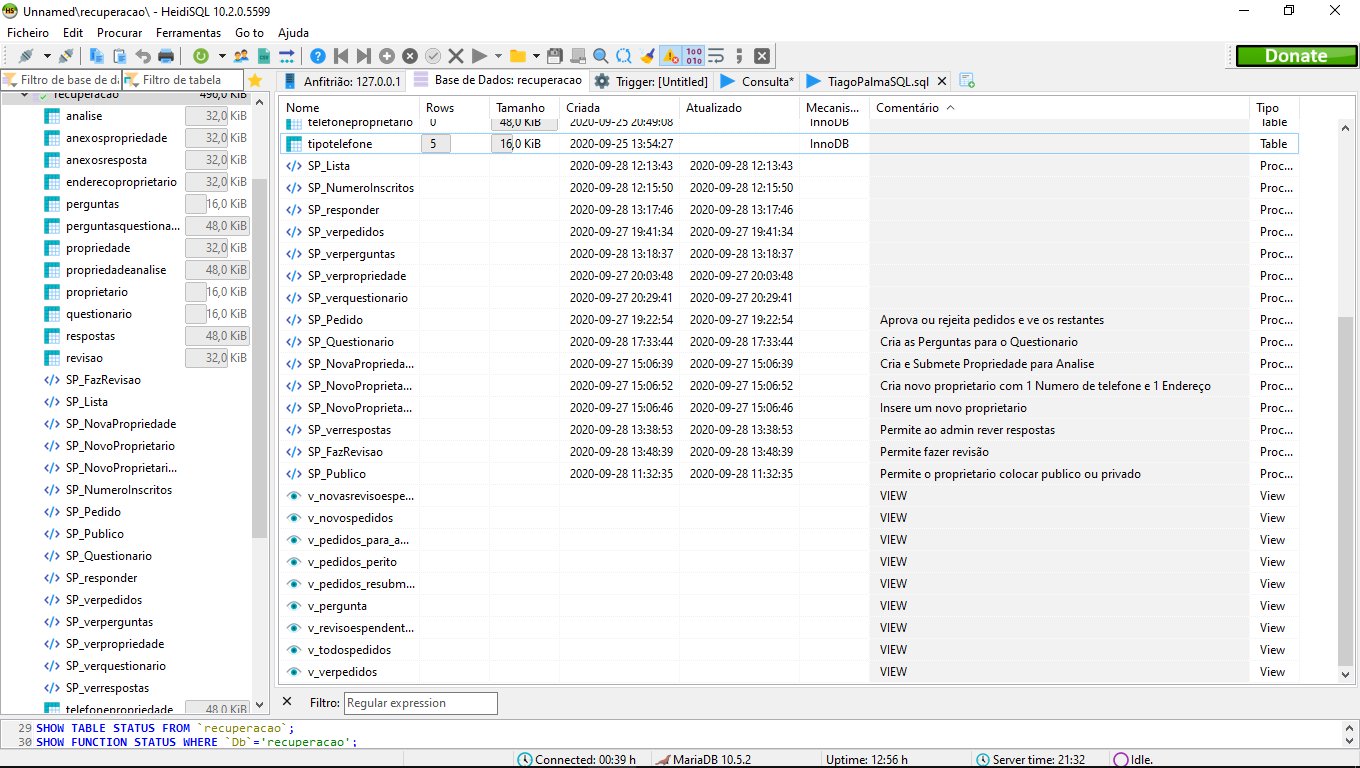


Depois dos teste e pequenas correções pude então implementar a base de dados incluindo todas tabelas e realizando pequenas alterações em algumas entidades e atributos. Para isso utilizei ferramentas como o terminal do windows, o Workbench, Heidi e Excel.

O resultado foram as seguintes tabelas:

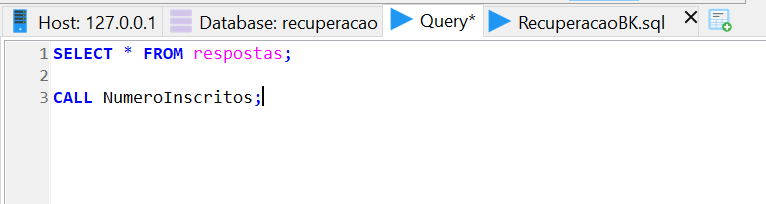


Criei também algumas Views e Stored Procedures que contêm informações uteis os utilizadores da base de dados e que permitem também inserir e editar dados conforme pedido no assigment brief:

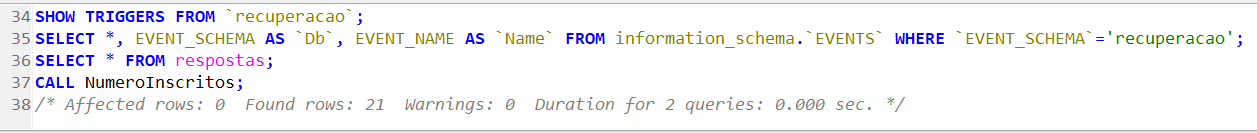


Boa parte das Stored Procedures foram devidamente comentadas nos seus respetivos códigos para permitir um melhor entendimento de cada instrução.

Para executar as queries, views e stored procedures podemos utilizar tanto o terminal quanto interfaces gráficas como o Heidi, na imagem seguinte poderemos ver na linha 1 um comando que nos permite chamar uma View e consequentemente ver a querie que esta possui. Já na linha 3 podemos ver o comando que chama e executa uma Stored Procedure



É possível ainda ver o tempo gasto pelo computador para executar a querie. Esta é uma forma que temos de analisar a performance da nossa base de dados:



Para otimizar esta base de dados em termos de performance, eficiência e espaço ocupado é necessário de aplicar algumas medidas já na fase da criação dos atributos. Atributos de texto como o char e o varchar possuem propriedades que impactam tanto a performance como o espaço ocupado.

O Char é mais adequado para texto com um número fixo de caracteres (EX: ‘M’ e ‘F’ ou ‘A’ e ‘B’), isto diminui o processamento necessário e consequentemente aumenta a performance da Base de dados.  
Alem disso quando simplificamos dados de “Masculino” para ‘M’ também ocupamos menos espaço de memoria. Por outro lado, se eu definir um char de 4 caracteres quando podem haver palavras com menos de 4 carateres, eu estou a ocupar espaço desnecessário na minha base de dados, isto ocorre porque as variáveis char são fixas, ou seja, caso eu diga que um atributo deve ocupar 4 caracteres, esse atributo irá ocupar obrigatoriamente 4 caracteres de espaço mesmo que eu introduza uma palavra com menos caracteres.

O Varchar por outro lado é um tipo de dado dinâmico, ou seja, se eu disser que o meu varchar é de 20 caracteres, porem introduzir na base de dados somente 10, os espaços que seria ocupado pelos outros 10 caracteres vão ficar livres para outros dados, ou seja, ocupamos menos espaço em disco. Por outro lado, o varchar acaba por pedir mais processamento, o que leva a uma menor performance.

Outra medida a ser tomada para melhorar na performance é definir somente o tamanho realmente necessário para um determinado atributo. Por exemplo, não vale a pena definir um Varchar de 250 caracteres quando os utilizadores irão utilizar apenas 30, nem vale a pena definir atributos como BigInt quando não vão ser utlizados números acima de 32,767 (valor suportado pelo tipo de dados SmallInt).

Outra medida utilizada nesta Base de dados foi a utilização de TINYINT(1) para dados que consistiam em “SIM/NÃO”, no caso do MySql não existe um tipo Booleano, em vez disso utiliza-se o TINYINT(1) onde 0 é FALSE e qualquer valor diferente de 0 é TRUE.

Finalmente uma das medidas utilizadas pelos profissionais de tuning de bases de dados é na utilização de operadores lógicos como o AND e o OR. No caso do OR, a condição com mais chances de ser verdadeira deve vir em primeiro lugar. Já no caso do AND a logica é inversa ao OR, ou seja, primeiro ver o statement com menores chances de ver verdadeira e só depois é que vem a com maiores chances. Isto irã poupar bastante o processamento da base de dados e melhorar a performance.

EX:

‘F’ -> 70% verdadeira

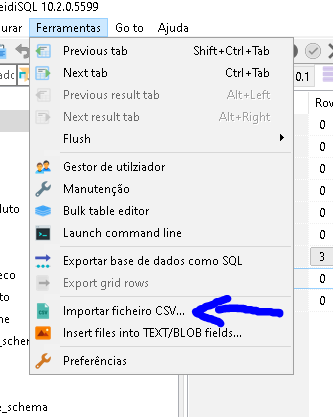
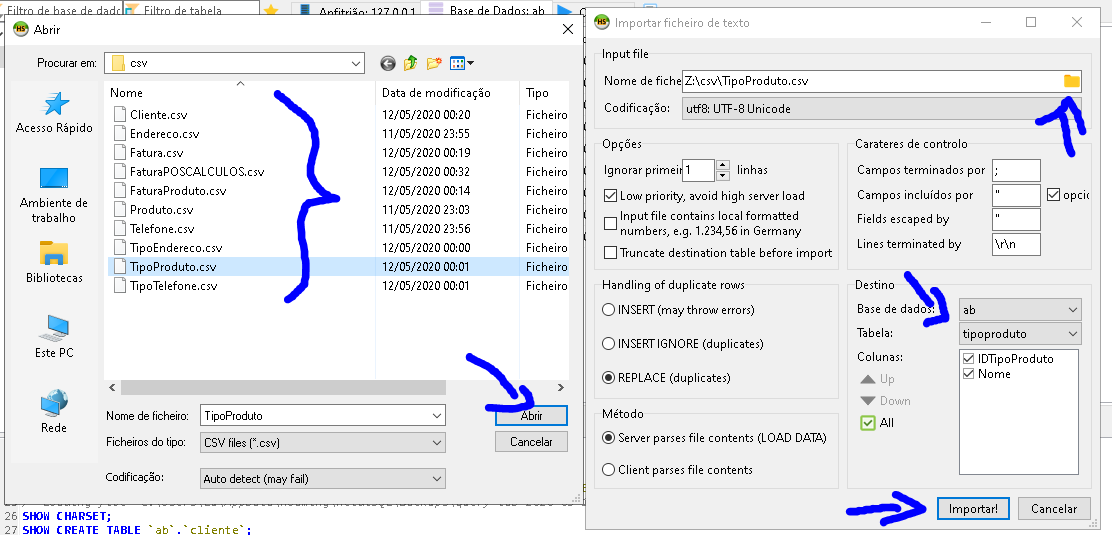
‘Lisboa’ -> 30% verdadeira

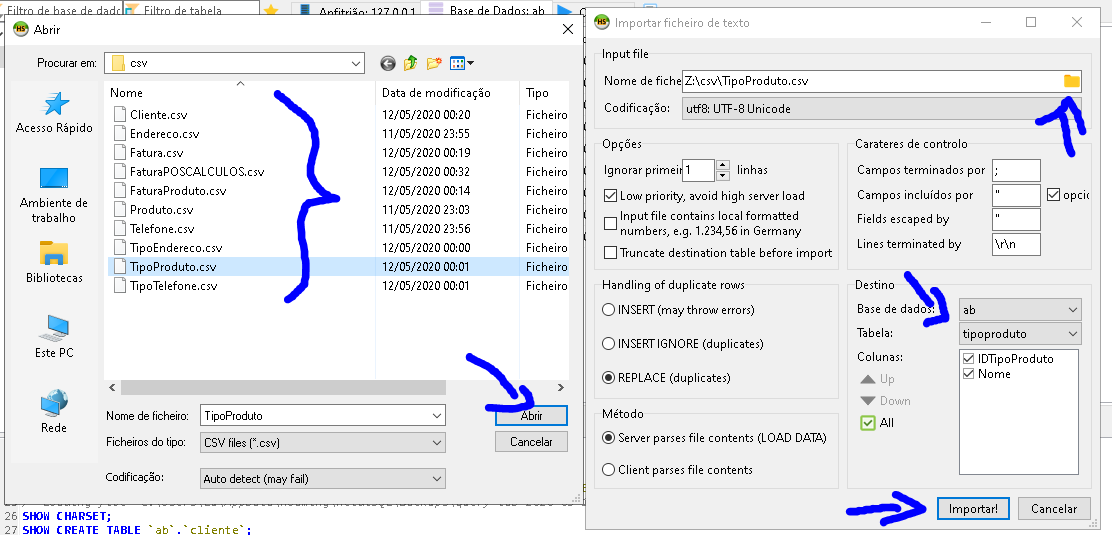
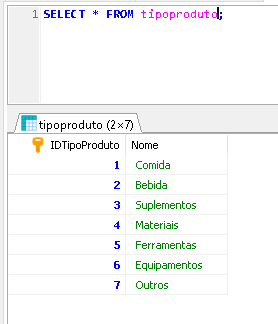
* No caso do OR
  + Forma correta:
    - Where sexo = ‘F’ OR cidade = ‘Lisboa’
  + Forma incorreta:
    - Where cidade = ‘Lisboa’ OR sexo = ‘F’
* No caso do AND
  + Forma correta:
    - Where cidade = ‘Lisboa’ AND sexo = ‘F’
  + Forma incorreta:
    - Where sexo = ‘F’ AND cidade = ‘Lisboa’

Outra estratégia básica e essencial para melhorar a eficiência da base de dados é a simples aplicação das Formas Normais, pois quando as aplicamos, estamos a evitar redundâncias (repetição dos mesmos dados), ocupando assim menos espaço em disco e facilitando até mesmo na busca de informação e numa melhor performance, pois uma base de dados sem redundâncias significa que ela é mais limpa, o que torna os dados mais acessíveis. Por isso é tão importante aplicar todas as 3 formas normais.

### Popular a base de dados:

Para popular as tabelas utilizei o excel, que nos permite não só criar os dados de forma mais organizada e rápida como também nos permite guardar esses dados em um ficheiro .csv, onde os dados são delimitados por virgulas, que depois são encontrados pelos programas de gestão de bases de dados assim que importamos tais ficheiros. Vejamos o procedimento nas imagens que se seguem:

**HEIDI:**

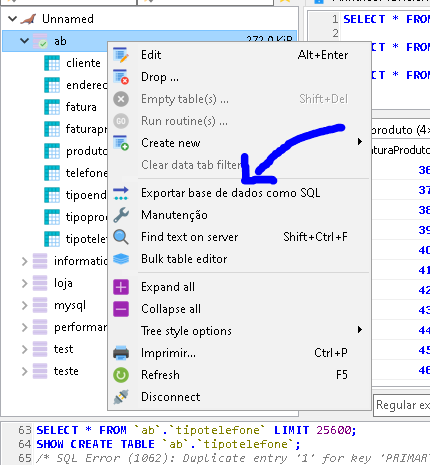


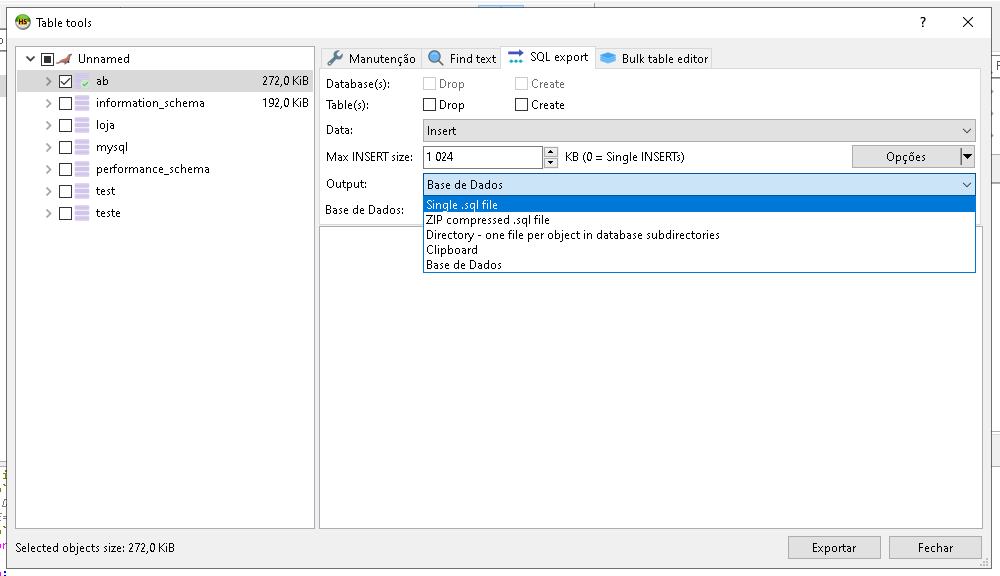
### Backup

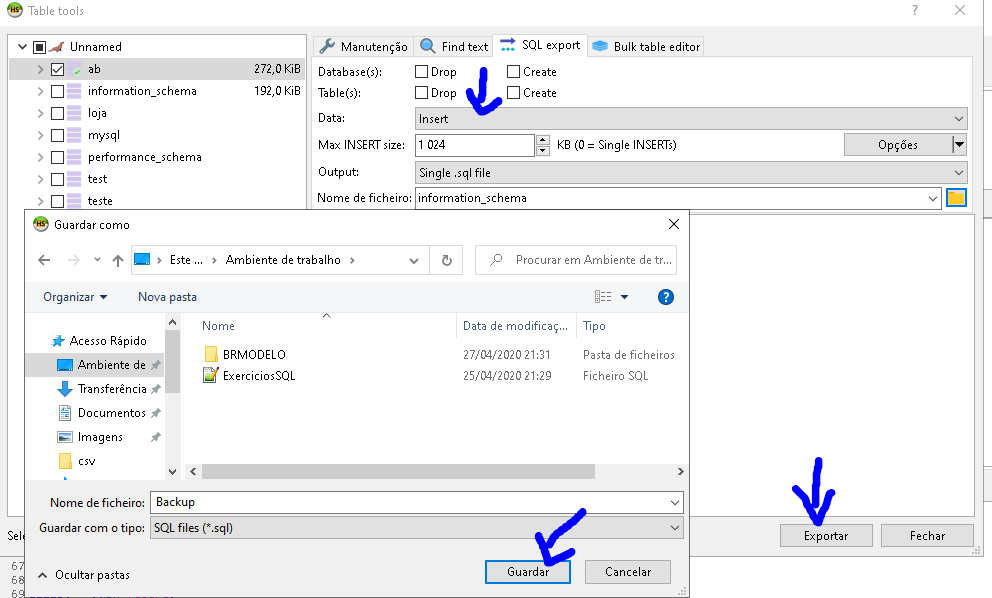
Com todos os dados armazenados na nossa base de dados devemos sempre nos preocupar com a segurança destes dados, para tal podemos utilizar a ferramenta de Backup, que não só permite guardar os dados como também pode ser utilizado para migrar uma base de dados de um servidor para outro.

Podemos realizar o Backup da seguinte forma:

**Heidi:**

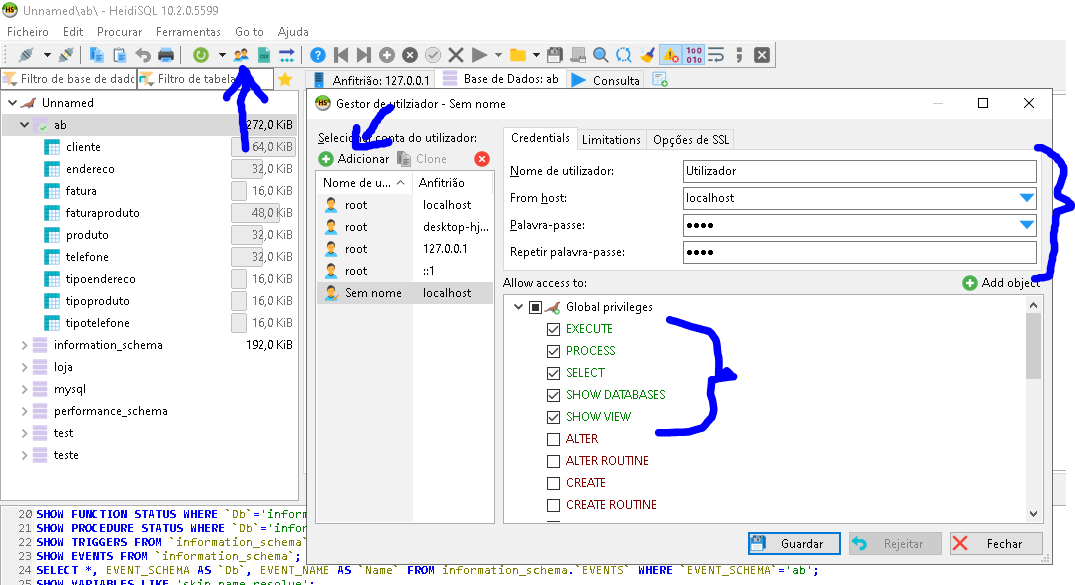






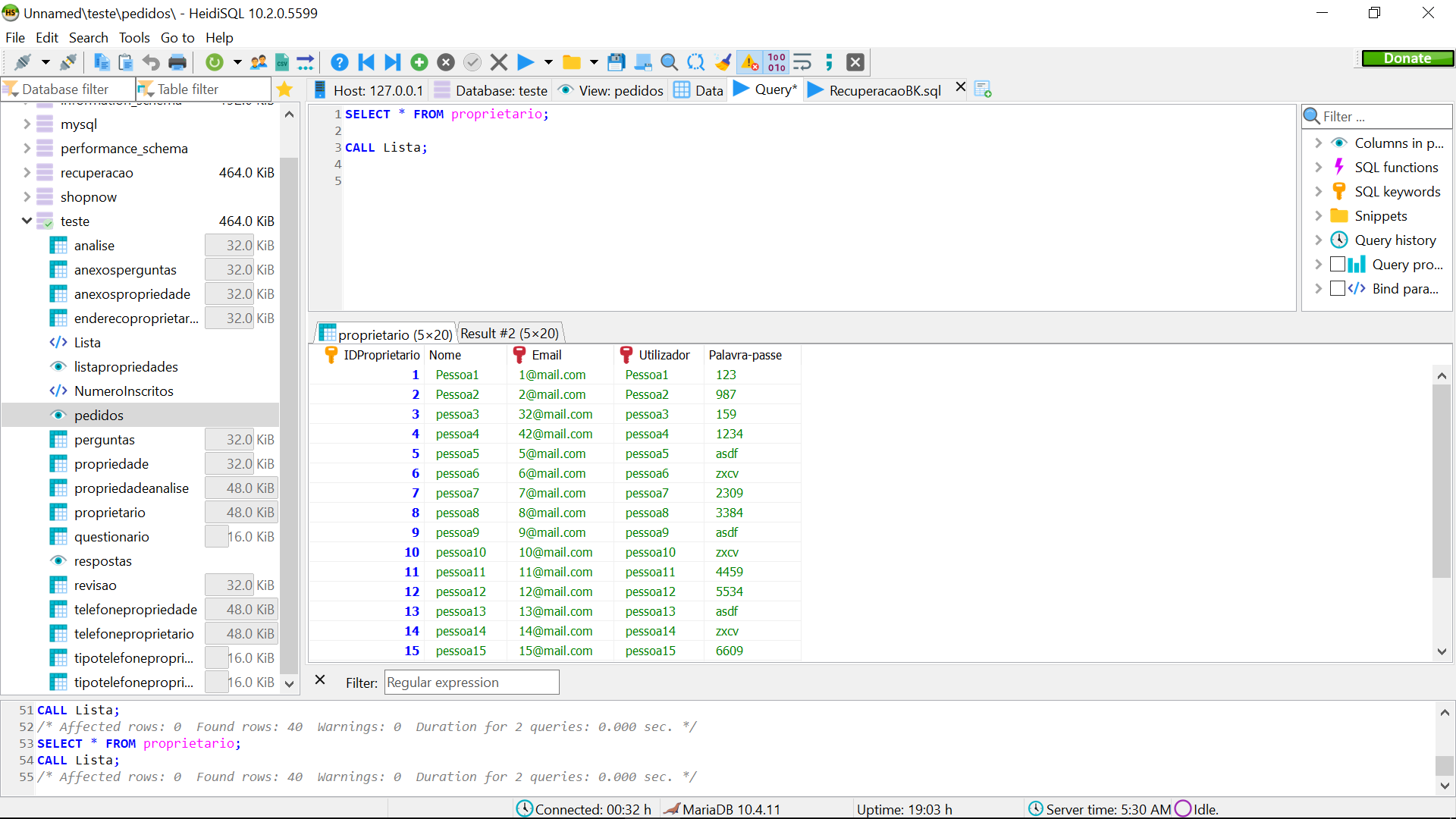
**Controles de Acesso e Privilégios:**

No Heidi, ao entrarmos na janela de gestão de utilizador, podemos definir o nome do utilizador, a sua password, as bases de dados que ele poderá ter acesso e ainda que tipo de acessos ele pode ter (Global privileges). Desta forma podemos dizer para a nossa base de dados que o utilizador “pessoa1” só pode ver a base de dados, enquanto que o utilizador “pessoa2”, pode ler, alterar e até fazer outros procedimentos, bastando apenas marcar as checkboxes como no exemplo da imagem abaixo. Isto permite uma maior segurança das bases de dados, pois caso ocorra algum problema, nós já sabemos exatamente quais são os utilizadores que têm privilégios mais avançados como alterar e apagar elementos da base de dados.

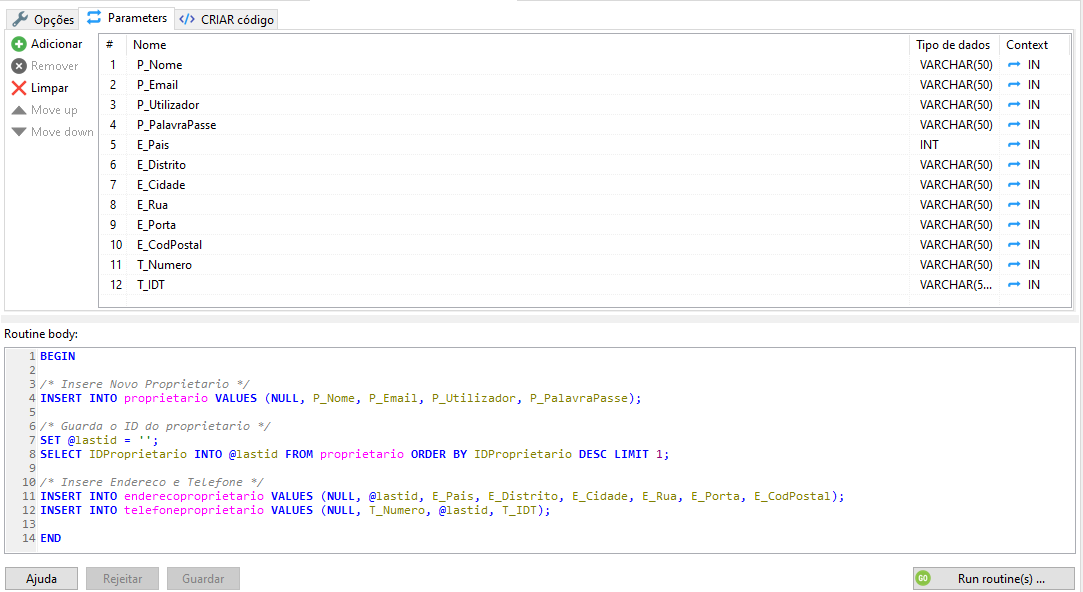


## Task 3

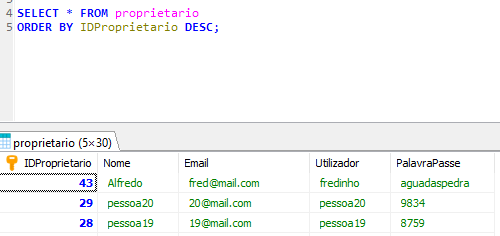
Finalmente pude testar as queries da base de dados e realizar os ajustes e correções finais, permitindo uma base de dados funcional, vejamos alguns exemplos:



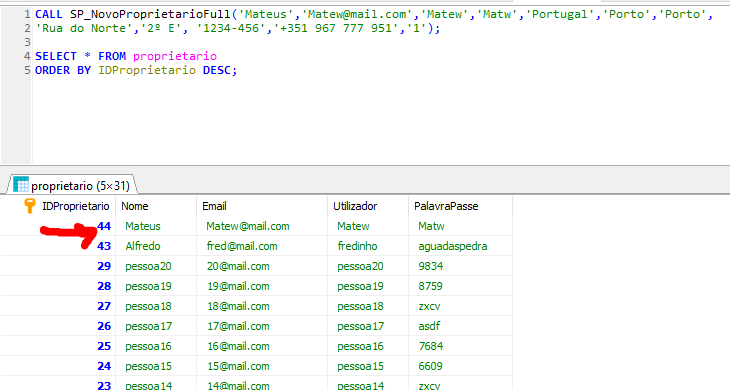
### Inserir Proprietário:



A Primeira Stored Procedure que iremos testar é a de inserir proprietário, acima podemos ver os parâmetros e o código SQL responsável por introduzir o Proprietário juntamente com seu endereço e numero de telefone.



Ao executar esta query podemos notar que o último proprietário registado é o de ID 43.



Ao utilizar-mos a Stored Procedure “SP\_NovoProprietarioFull” com todos os dados necessários, conseguimos criar por completo um novo registo de um proprietário. Existe também a Stored Procedure “SP\_NovoProprietario” onde escrevemos apenas os dados referentes a tabela “proprietário”.  
Vejamos os seguintes testes que seguem a mesma logica:

### Registar / Submeter propriedade:

Antes da storage procedure:



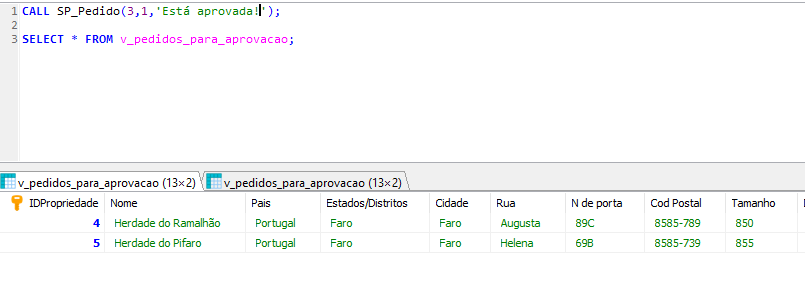
Depois da storage procedure:



### Ver e aprovar pedidos:

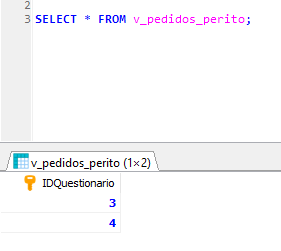
### 

Para que o Admin possa ver os pedidos que estão pendentes para aprovação ele poderá utilizar a View “v\_pedidos\_para\_aprovacao”, esta View filtra todos os registos onde a data de submissão é superior a data de analise ou onde a data de analise é igual a “NULL”, desta forma o Admin consegue ver com precisão todos os pedidos que faltam.



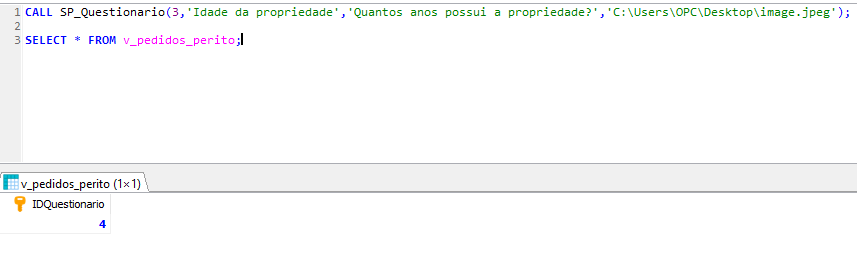
Quando a Stored Procedure é executada a propriedade editada desaparece da lista, além de que a própria storage procedure possui a view “v\_pedidos\_para\_aprovacao” no final do seu código, o que permite ao Admin ver a versão mais recente das propriedades em falta sem ter que digitar uma query nova.

### Questionário:

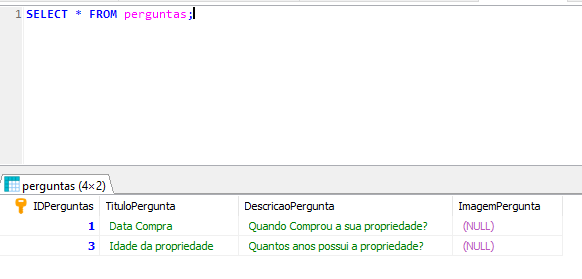
Para ver os questionários que precisa fazer, o perito apenas necessita de utilizar a View “v\_pedidos\_perito”.

Para Criar as questões basta utilizar a Storage Procedure “SP\_Questionario” e inserir os campos:

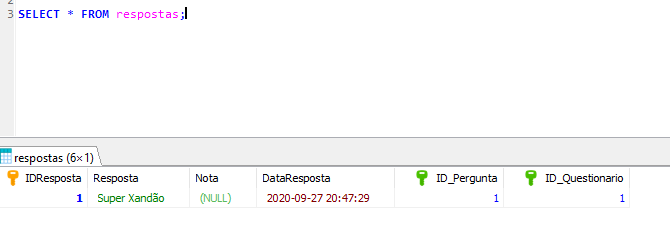
* ID do Questuinario
* Título da pergunta
* Descrição da pergunta
* Localização da imagem que pretende anexar



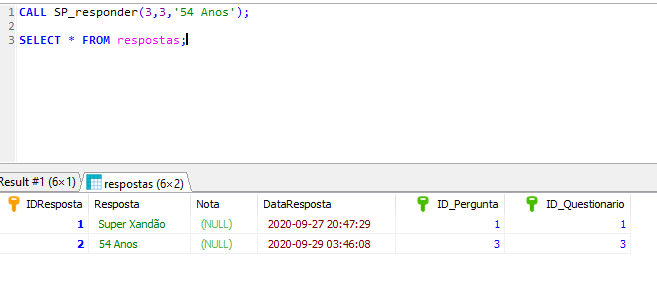
Assim que a procedure for executada a view não irá mostrar mais o questionário em questão, porém o perito poderá continuar a inserir mais perguntas para o mesmo questionário, bastando apenas manter o ID, podemos ver o resultado abaixo:



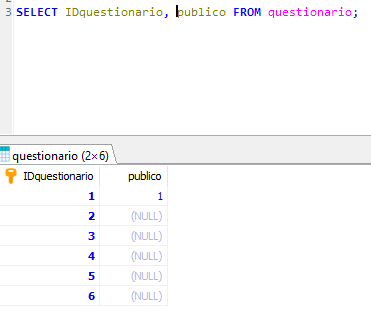
### Responder perguntas:



Para responder as questões basta utilizar a procedure “SP\_responder”:



### Publico/Privado:



Para finalizar os exemplos podemos ver que através da procedure “SP\_Publico” é possível que o proprietário mude os seus resultados para publico ou privado bastando inserir os campos ID do questionário e a opções 1 para publico ou 0 para privado:

