08

Introdução a certificação Oracle

Introdução

Bem-vindo ao curso da certificação Oracle. Nele vamos aprender os principais conceitos sobre um banco de dados, como ele funciona, como é implementado pela Oracle e como pode rodar dentro de uma máquina.

Para começar a discutir um pouco sobre isto, vamos pensar em uma situação real. Vamos imaginar que queremos ir ao mercado fazer a "compra do mês" e comprar diversos itens essenciais. Por exemplo, vou comprar refrigerante, lasanha e chocolate. Minha casa também vai precisar estar limpa, então, comprarei sabão em pó, detergente, shampoo... Para o jardim, vou precisar de alguns utensílios, como uma tesoura de jardinagem.

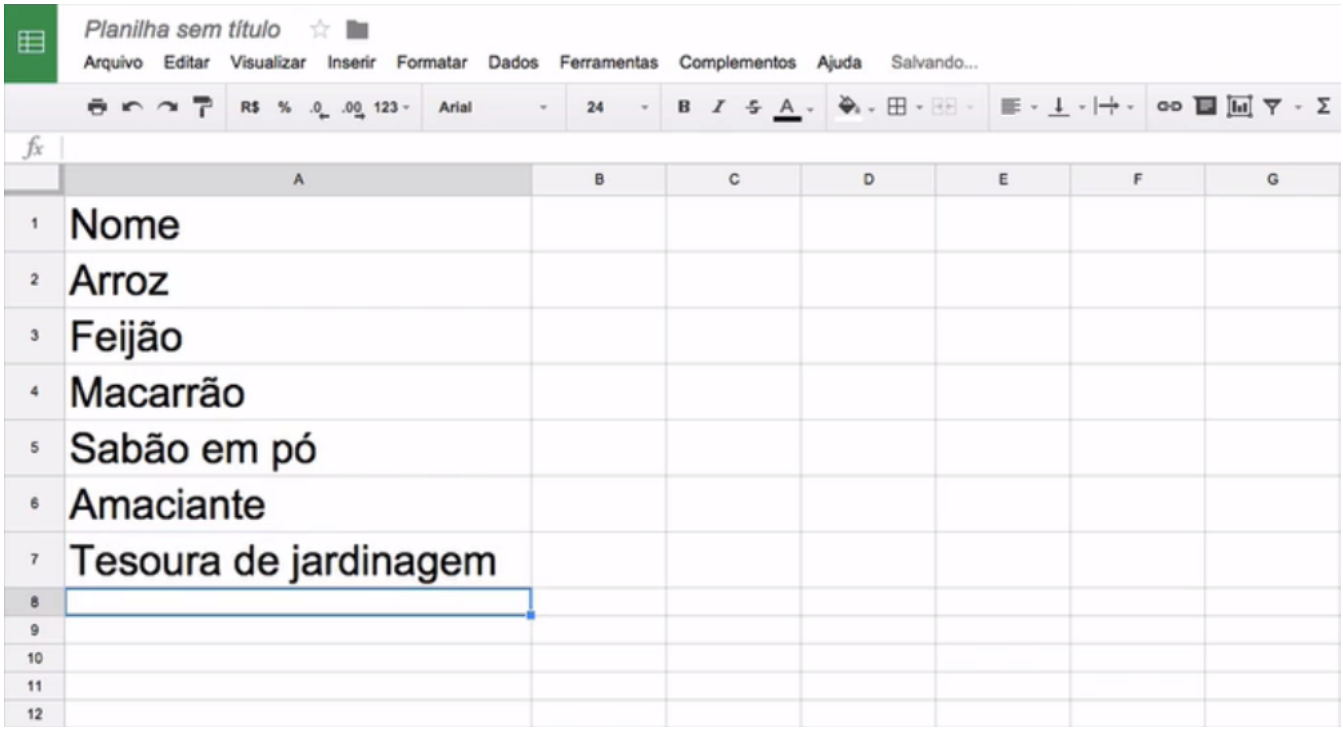
O grande problemas será lembrar de tudo isto, quando eu chegar ao mercado.

Como não esquecer?

A primeira ideia que me vem a mente é uma que costumava ser adotada pelas pessoas: fazer uma lista de compras.

> Lista de compras
>
> Arroz
>
> Feijão
>
> Macarrão
>
> Sabão em pó
>
> Amaciante
>
> Tesoura de jardinagem

Para começar, vou fazer uma lista no Excel. Vou colocar o nome do que vou comprar.



Agora, eu posso consultar a minha lista de compras e verificar tudo o que preciso comprar. Na minha lista, eu consigo armazenar todas as informações que eu preciso para ir ao mercado e saber quais itens comprar.

Nós podemos dizer que **a nossa lista é um banco de dados**.

Mas, o que é um banco de dados? Quais são as características da nossa lista de compras? Na lista de compras conseguimos adicionar e consultar informações, que estão dispostas de uma maneira organizada. O conceito de um banco de dados é parecido: trata-se de uma estrutura usada para organizar e salvar informações.

A grande pergunta que nos fazemos é sobre quem cuidará do banco de dados. No meu caso, eu abri o Excel e criei a minha lista. Sendo assim, o responsável pela lista, sou eu. Nós somos os responsáveis pelas nossas listas de compras.

No entanto, a verdade é que eu não sei tudo o que preciso comprar para minha casa... Na prática, quem irá cuidar da minha lista?

A minha mãe ou alguém que saiba melhor o que a casa precisa. A minha mãe é responsável por adicionar, remover e definir como irá organizar a minha lista. Para mim, é irrelevante se ela criará a lista no celular, no papel ou no Excel... Mas toda vez que eu quiser que algo seja adicionado, eu irei avisar a minha mãe, porque é ela quem gerencia a lista. Minha mãe é o sistema gerenciador de banco de dados, responsável por cuidar de todas as operações e pela forma como as informações serão salvas.

No banco de dados Oracle não é muito diferente. Nele temos os ***Database Management System*** **(DBMS)**. São sistemas responsáveis por um banco de dados.

Porém, quando peço algo para minha mãe, qual é a linguagem utilizada por mim? A minha mãe fala português. Então, eu posso pedir "mãe, adiciona um chocolate na lista" e ela irá adicionar. Ou se eu quiser que ela tire algum item, eu também posso pedir.

E no banco de dados Oracle, como iremos fazer isto? Como eu peço para o meu sistema gerenciador de dados, adicionar ou remover algo? Nós precisamos falar na linguagem que ele entende. Enquanto a minha mãe fala português, o banco de dados entende **SQL**, que significa *Structured Query Language (Linguagem Estruturada de Consultas, traduzido para o português).

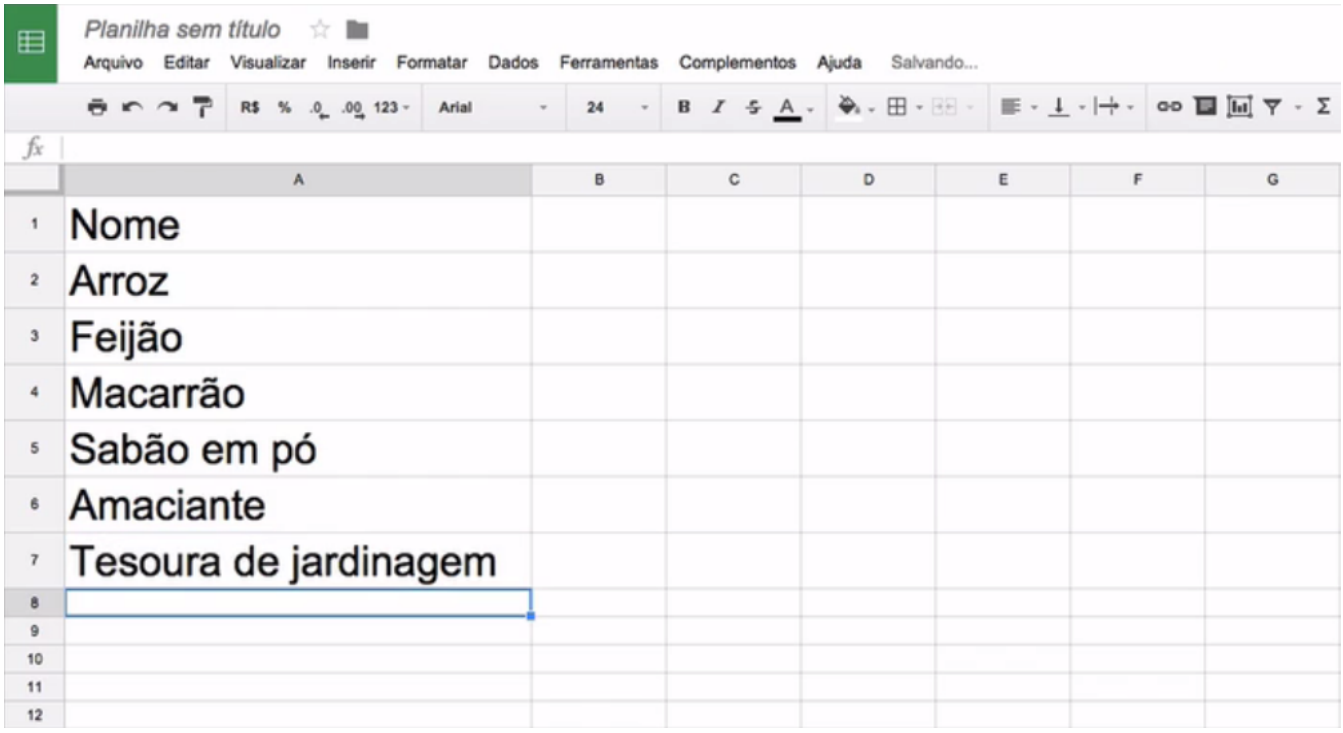
Toda vez que precisarmos mexer no banco de dados, iremos usar SQL, uma linguagem que iremos começar a conhecer agora e iremos nos focar ainda mais. Afinal, a primeira prova da certificação se chama "SQL Fundamentals" ou seja, uma avaliação sobre os fundamentos do SQL.

Nestes primeiros cursos falaremos bastante sobre SQL e as características da linguagem e como a Oracle implementa isto. Até os próximos cursos!

Entidades e relacionamentos

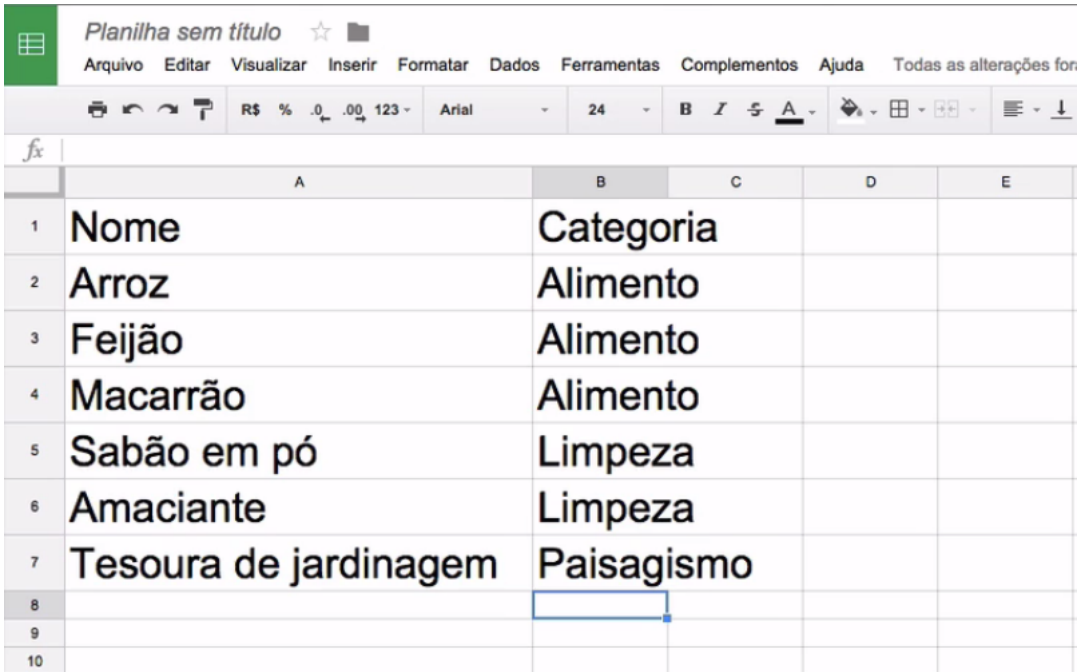
Vamos continuar com o exemplo da minha lista de compras, vamos imaginar que este mês, eu fui para o mercado e vi que as coisas estavam mais caras. Decidi então, comprar apenas o que para mim era mais importante, ou seja, comida. De todos os itens que tenho anotado na minha lista, irei aproveitar apenas os **Alimentos**. Como faço para analisar todos os elementos e selecionar apenas os alimentos?

Vamos observar a lista novamente:



Como eu sei quais itens são alimentos? Ao ler o nome "arroz", eu sei que é alimento. O mesmo acontece com "feijão" e "macarrão", também são comida. Mas na nossa lista, onde está especificado que eles são alimentos? Nós identificamos quais são, porque sabemos quais são comestíveis. Mas na minha lista precisa aparecer quais produtos são alimentos.

Por isso, vou criar uma nova coluna na minha planilha que será a "Categoria". Nela, irei especificar quais produtos são alimentos: "Arroz", "Feijão" e "Macarrão". Também vou informar que o "Sabão em pó" e "Amaciante" estão na categorias de "Limpeza". E a "Tesoura de jardinagem", vou categorizar como "Paisagismo".

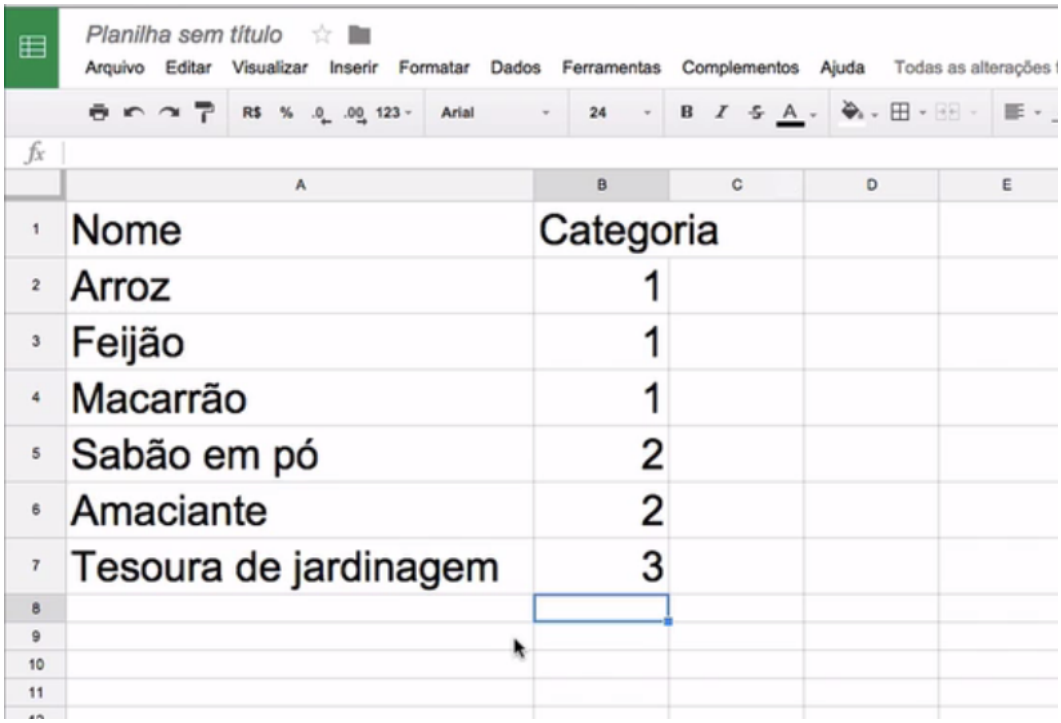


Agora, além de salvar na lista o "nome" do produto que eu quero comprar, precisarei salvar as categorias. Se eu quero saber o que é alimento, não preciso olhar o nome. Posso selecionar os itens pela coluna de "Alimento", por exemplo.

Porém, observe que repeti três vezes a palavra "Alimento". A palavra "Limpeza", eu também repeti. Mas vamos imaginar que a lista fosse maior. É provável que repetiríamos as categorias inúmeras vezes. Imagina se eu quisesse anotar a lista na minha

mão?

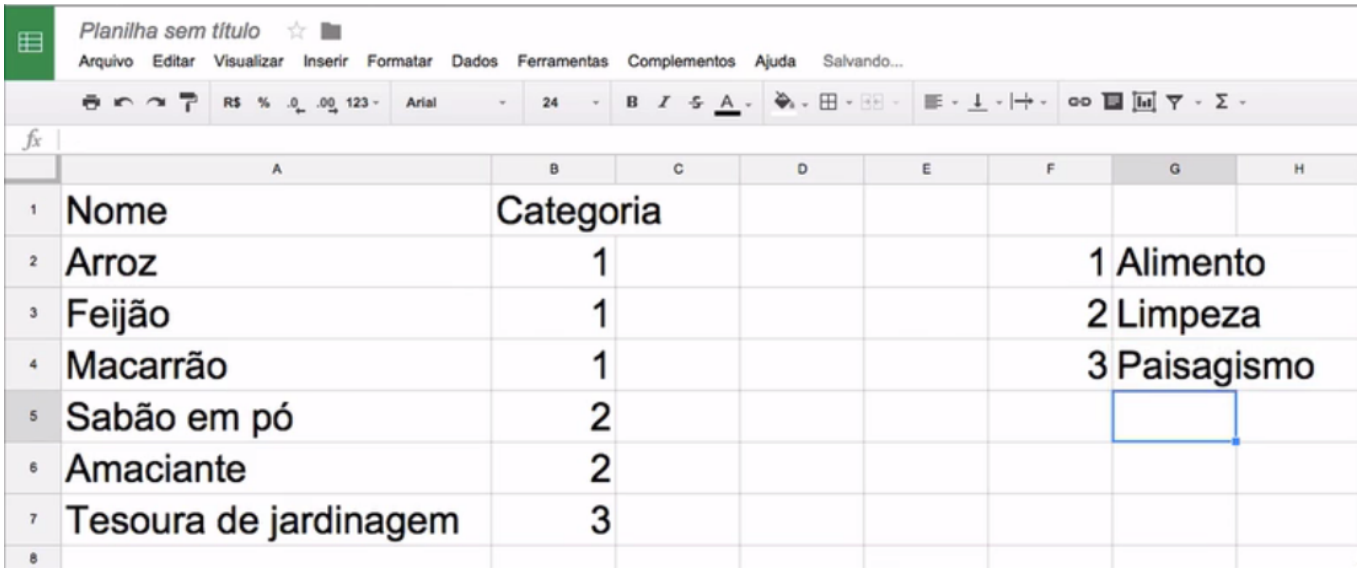
Minha sugestão é criarmos uma forma mais simples de anotar os produtos. Podemos gerar uma categoria mental e o que for alimento, será categorizado como "1". Todos os itens que forem da **categoria 1**, eu saberei que são alimentos. A categoria "Limpeza", eu posso usar o número "2", e "Paisagismo", substituirei pelo "3".



O objetivo é facilitar a busca, agilizando o processo. Verificar quais são os produtos "1-1-1" é mais simples do que "alimento-alimento-alimento". Usar números é mais rápido.

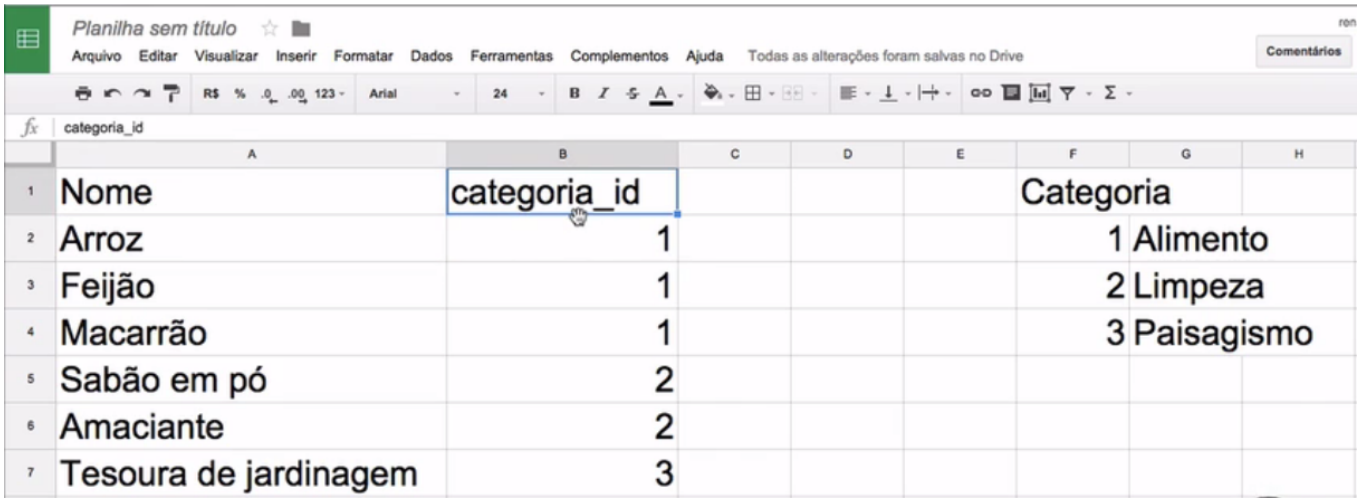
Mas, vamos imaginar que alguém abra a minha lista amanhã. Como outra pessoa irá saber o que é o número 1? Eu posso criar um tabelinha do lado explicando que:

- **1** representa "Alimento".
- **2** representa "Limpeza".
- **3** representa "Paisagismo".

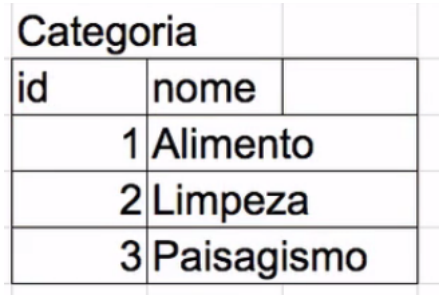


Agora, qualquer pessoa irá conseguir fazer o link entre os número e as categorias. Então, para organizar melhor minha planilha, vou utilizar o termo "Categoria" para fazer referência à nova estrutura que efetivamente fala sobre as três categorias. E irei nomear o antigo campo como algo que identifica uma categoria. Se eu preenchi com "1" a coluna ao lado do "Arroz", é possível que ele não seja alimento? Não. Na tabela de "Categorias", vemos que apenas "Alimento" é representado pelo número "1". A coluna à esquerda, informa os número correspondentes e registra as categorias de forma **única**. Nós a chamamos de **chave primária**.

Na estrutura de produtos, a coluna que se chama "Categoria" passará a se chamar de "categoria_id", em referência à chave de outra tabela. Por ela ser de outra tabela, iremos dizer que se trata de uma **chave estrangeira**.



Vou fazer alterações na tabela da esquerda, especificando que ela é referente a "Produtos".



E outras referente a "Categorias", que será composta pelas colunas: "id" e "nome".

Agora temos duas tabelas.

O que é importante salvar

Vimos que era importante salvar duas informações: o **Produto** e as **Categorias**. Por isso, acabei criando tabelas exclusivas para salvar as informações. Isto é o que chamamos de **Entidades**.

Como definir o que são entidades? São estruturas que irão salvar as informações.

Observe que na nossa planilha, temos duas estruturas: à esquerda, temos a estrutura "Produto", com os campos "Nome" e "categoria_id", e à direita, temos a estrutura "Categoria", com os campos "id" e "nome".

As colunas "Nome" e "categoria_id", da primeira tabela, falamos que são **atributos** da minha entidade "Produto". Observe que o item "Arroz" **não** é um atributo, ele é um dado dentro da entidade, que recebe o nome de **registro**.

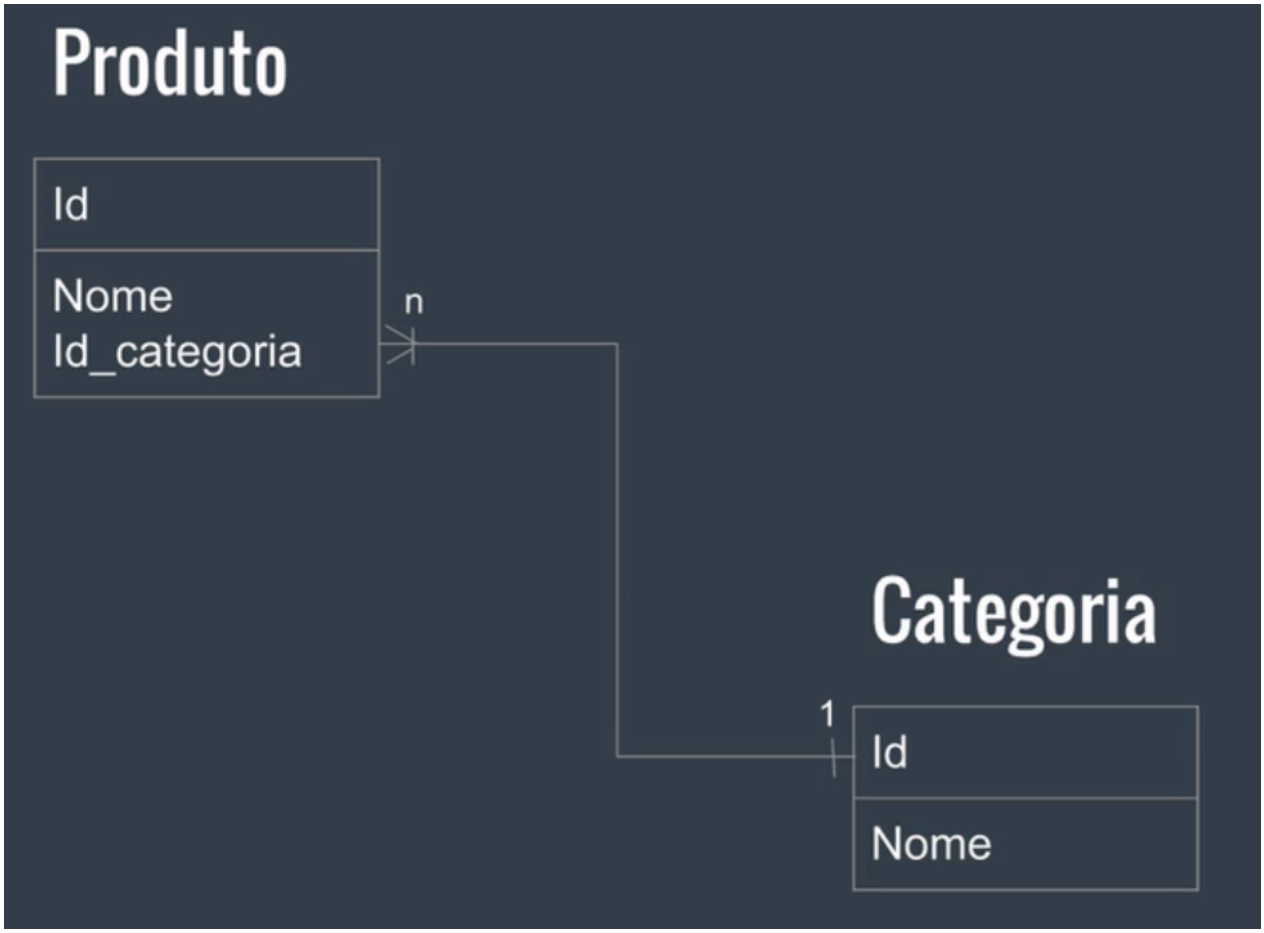
Na tabela ao lado, vemos a mesma coisa. A coluna "id" é a chave primária da minha categoria. Além disso tanto ele, com "nome", são atributo da entidade "Categoria".

Da mesma forma, como precisamos identificar uma categoria de forma única, por exemplo, se eu procurar por um "id", eu só irei encontrar sempre um registro, é necessário identificar um produto de forma única.

Para identificar um produto de forma única, eu vou inserir uma nova coluna, na tabela da esquerda. As novas serão referentes ao `id` dos produtos. A forma mais simples de criar `ids` únicos é numerar os itens a partir do `1`. Como temos seis produtos, iremos criar `ids` do `1` até o `6`.

	A	B	C	D
1	Produto			
2	id	Nome	categoria_id	
3	1	Arroz	1	
4	2	Feijão	1	
5	3	Macarrão	1	
6	4	Sabão em pó	2	
7	5	Amaciante	2	
8	6	Tesoura de jardinagem	3	
9				
10				

Nós vimos que "Produtos" e "Categorias" são **estruturas que iremos salvar as informações**. O "Produto" ficou com os atributos "Nome" e "Categoria". E a estrutura "Categoria" ficou com o atributo "Nome". E como as estruturas se relacionam?



Observe que existe um relação entre elas, porque o meu produto tem uma **chave estrangeira** referente a "Categoria".

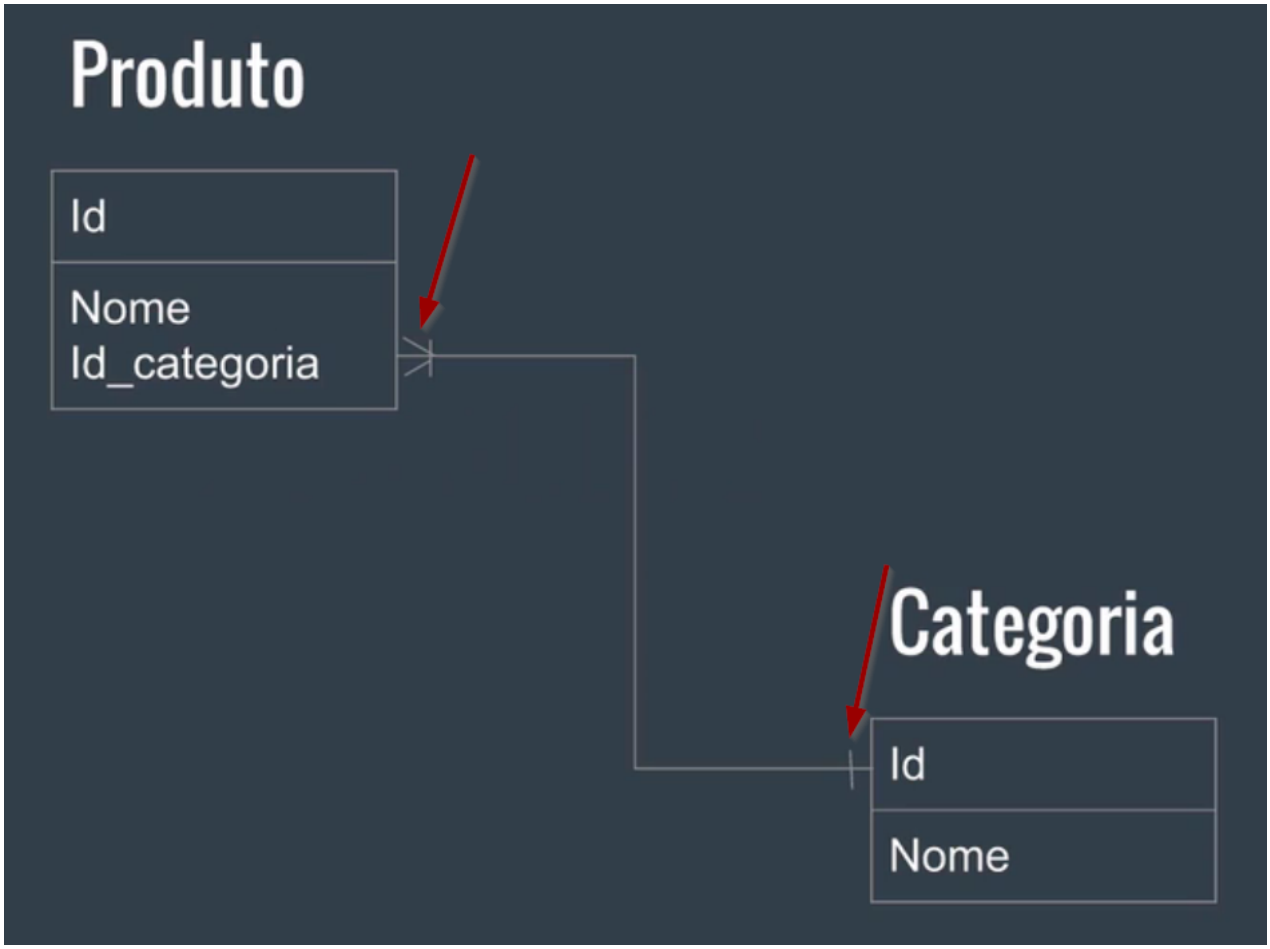
Nós falamos que um produto tem uma categoria, e uma categoria pode estar em vários produtos.

Por exemplo, na nossa tabela "Produto", quantos itens estão indicados como sendo da categoria "Alimentos"? A resposta é: **três** produtos.

Só existe uma categoria "Alimento" e ela pode estar em vários produtos. Outra relação é que um produto pode ter apenas uma categoria, porque só temos um campo para preencher. Ou seja, uma categoria pode estar em vários produtos, mas um produto só pode ter uma categoria.

Este tipo de relacionamento é o que chamamos de ***One-to-Many*** (o significado é que uma categoria está em muitos produtos).

Nós representamos eles utilizando com **tracinhos**. Se eu tenho uma categoria, eu represento com um traço na vertical e onde a categoria pode estar em diferentes produtos, representamos com um **pé de galinha**.



One-to-One

Nós já conseguimos selecionar apenas os alimentos da lista. Depois, colocamos todos no carrinho, e vamos pagar a conta! Em alguns supermercados, quando estamos no caixa, eles nos perguntam se fazemos parte do **clube especial** de clientes. Se nós formos cadastrados, geralmente, ganhamos um desconto. Para o caixa saber se você é um cliente especial ou não, o que ele precisa te pedir? O seu CPF.

Com o número, de alguma forma ele fará a consulta e irá verificar o meu cadastro. Mas que tipo de informações têm o banco de dados dele? Ele terá uma entidade `cliente`.

Vamos voltar para o Excel. Na planilha, vamos criar uma nova estrutura, chamada "Cliente". Vamos pensar quais campos podemos salvar? Toda entidade precisa ser identificada de forma única, nós vamos criar o `id`. Também é interessante sabermos o `nome`, `cpf`. Iremos adicionar os seguintes dados do endereço: `logradouro`, `cep`, `bairro`, `cidade`, `estado`, `pais`, `complemento`, `número`.

Compra	
id	produto_id
1	3

Na minha tabela cliente, a maioria dos meus dados são relacionados ao endereço. Se ele é tão importante e tem tantos campos, não faz sentido mantê-lo na entidade "Cliente". Então, vamos criar uma entidade chamada "Endereço" e iremos transferir os dados que não são referentes ao cliente para a nova tabela. Como também precisaremos identificar o endereço, vamos criar um campo "id".

Compra		
id	produto_id	cliente_id
1	3	5
2	4	5

Vou começar a preencher as tabelas com os dados. O nome do cliente será "Renan Saggio" e o CPF é "123456789-11". Iremos preencher os campos do endereço com os seguintes dados: Rua Vergueiro, Bairro Vila Mariana. A cidade é São Paulo e o estado, São Paulo. Também iremos adicionar o CEP: 1234050 e o país é Brasil.

Compra		
id	produto_id	cliente_id
1	3	5
2	4	5

Compra_Produto	
compra_id	produto_id
1	3

Os clientes costumam ter apenas um endereço. E o endereço pode estar relacionados a quantos clientes? Geralmente, está relacionado a um também. Para linkar o "Cliente" com o "Endereço", eles precisam ter o mesmo número de "id". Quando o número do "id" do cliente for "5", o "id" do endereço deverá ser também "5". Desta forma, conseguiremos pesquisar o dados dos clientes.

Observe que a tabela grande que criamos ainda existe, apenas a quebramos em pedaços menores.

Vamos analisar o que fizemos e ver quais foram as vantagens e desvantagens do processo.

Temos a entidade **Cliente**, concluímos que precisaríamos criar a entidade **Endereço**. A pergunta é: como elas se relacionam? Eu tenho um cliente, que tem um endereço. E tenho um endereço, que pode ser de apenas um cliente. Este tipo de relacionamento em que um se relaciona com o outro, ou seja, **um para um**, é o que chamamos **One-to-One**.

Você se lembra do relacionamento que vimos anteriormente? Era o **One-to-Many, no qual um registro poderia ocorrer várias vezes em outra tabela. No nosso caso atual, só temos um endereço relacionado com um cliente. Nós chamamos este relacionamento de* One-to-One*.

Da mesma forma que nós representávamos com uma barrinha reto no diagrama anterior, faremos o mesmo com o atual.

Só um endereço?

Podemos nos perguntar: "nós só precisaremos de um endereço?" As pessoas, geralmente, moram em apenas um lugar. Porém, eu não moro sozinho. A minha mãe mora no mesmo endereço, logo, ela irá informar o mesmo endereço, quando for se cadastrar no clube especial do mercado. É possível concluir que o mesmo endereço pode estar no cadastro de mais de um cliente...

Então, qual é o problema de utilizar o *One-to-one*? Com ele, definimos uma regra para o meu banco de dados, em que cada cliente só pode ter um endereço.

Veremos mais adiante que alterar um banco de dados, quando ele já é utilizado por uma aplicação, não é nada fácil. Mas o relacionamento *One-to-Many* atende o caso "um para um" também. Na tabela de produtos da lista, nós também tínhamos um caso "um para um". A tesoura de jardinagem era o único item da categoria "Paisagismo". Da mesma forma que a categoria "Paisagismo" tinha apenas um item.

A diferença do *One-to-Many* para o *One-to-one* é que se precisarmos adicionar um item na categoria "Paisagismo", nós conseguiremos com a primeira. Porém, não conseguiremos adicionar um novo endereço de cliente, porque eu forcei que o "id" das entidades "Clientes" e "Endereço" fossem iguais. Então, só é possível relacionar um para um.

Se optarmos em utilizar o relacionamento *One-to-one* estaremos preso a esta regra e só poderemos ter uma ocorrência, sempre. Se eu posso deixar o processo mais flexível e fazer com que funcione para uma ou mais ocorrências, por que não usaríamos o *one-to-many*? Uma pergunta melhor é: em que casos usamos o *one-to-many*? O caso mais comum é utilizarmos o *One-to-Many* quando precisamos de uma performance melhor, porque dependendo da forma como o nosso banco de dados for implementado - por exemplo, a tabela "Cliente" com menos campos - ficará mais fácil realizarmos buscas nela. Se os campos mais usados pelo meu sistema são o "nome" e o "CPF" do cliente, poderíamos agilizar a busca de uma destas informações, ao não termos que pesquisar em todos os outros campos.

Many-to-many

Continuando com o nosso curso de Oracle, queremos agora pagar a nossa compra. Ela será registrada e por isso, eu preciso salvá-la no meu banco de dados. Para isto, eu precisarei criar uma entidade que receberá o nome de `Compra`.

De volta ao Excel, iremos criar a nova entidade. Iremos adicionar as colunas "id", "produtos_id". Na coluna de produtos, iremos utilizar os "ids" da tabela de **Produtos**. Por exemplo, na primeira tabela, o `id` do "Macarrão" era "3". Se o macarrão for o primeiro produto da minha compra, irei preencher a coluna "id" da nova entidade com o número "1" e a coluna "produto_id" com o número "3".

Compra	
id	produto_id
1	3

Em toda compra, temos um produto, mas também temos um **cliente**. Adicionaremos uma coluna chamada "cliente_id" e sabemos que o `id` do cliente Renan é "5".

Mas e se quisermos registrar um segundo item, teremos que criar uma nova compra, ou seja, teremos que gerar um `id` igual a "2"?

Compra		
id	produto_id	cliente_id
1	3	5
2	4	5

Não é assim que acontece no cotidiano. Nós não vamos no mercado, compramos leite e pagamos, para em seguida, comprarmos um chocolate e pagarmos novamente. Geralmente, em uma mesma compra podemos incluir vários produtos. Além disso, o mesmo produto pode estar em várias compras. Por isso, vamos criar uma nova estrutura chamada `Compra_Produto`. Nela iremos incluir os campos "compra_id", e "produto_id". Nesta estrutura iremos preencher que a minha compra "1" tem o produto "3".

Compra		
id	produto_id	cliente_id
1	3	5
2	4	5

Compra_Produto	
compra_id	produto_id
1	3

Desta forma, não precisaremos adicionar uma nova compra, mas sim, adicionar um outro item no `Compra_Produto`, usando o mesmo `id`. Agora, os itens "3" e "4" estão inclusos na mesma compra.

Compra_Produto	
compra_id	produto_id
1	3
1	4

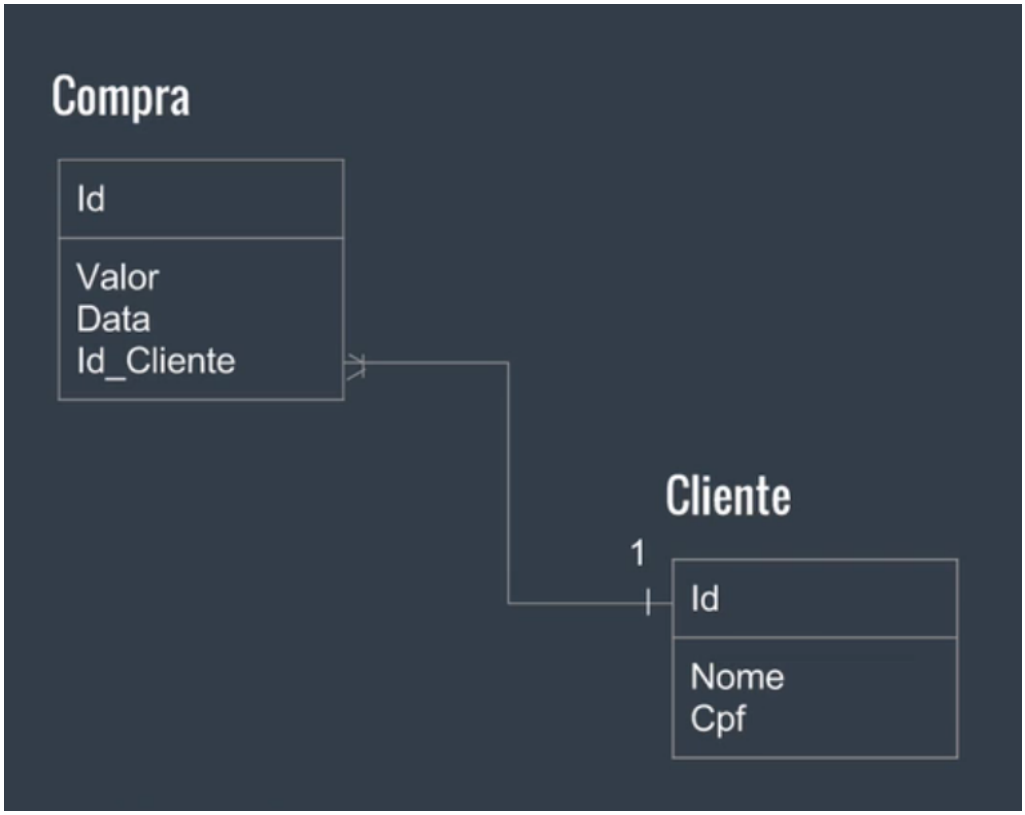
Já não precisaremos da coluna "produto_id" da tabela `Compra`. Mas iremos adicionar uma coluna que mostre o "valor" total da compra, R$200.

Compra		
id	cliente_id	valor
1	5	200

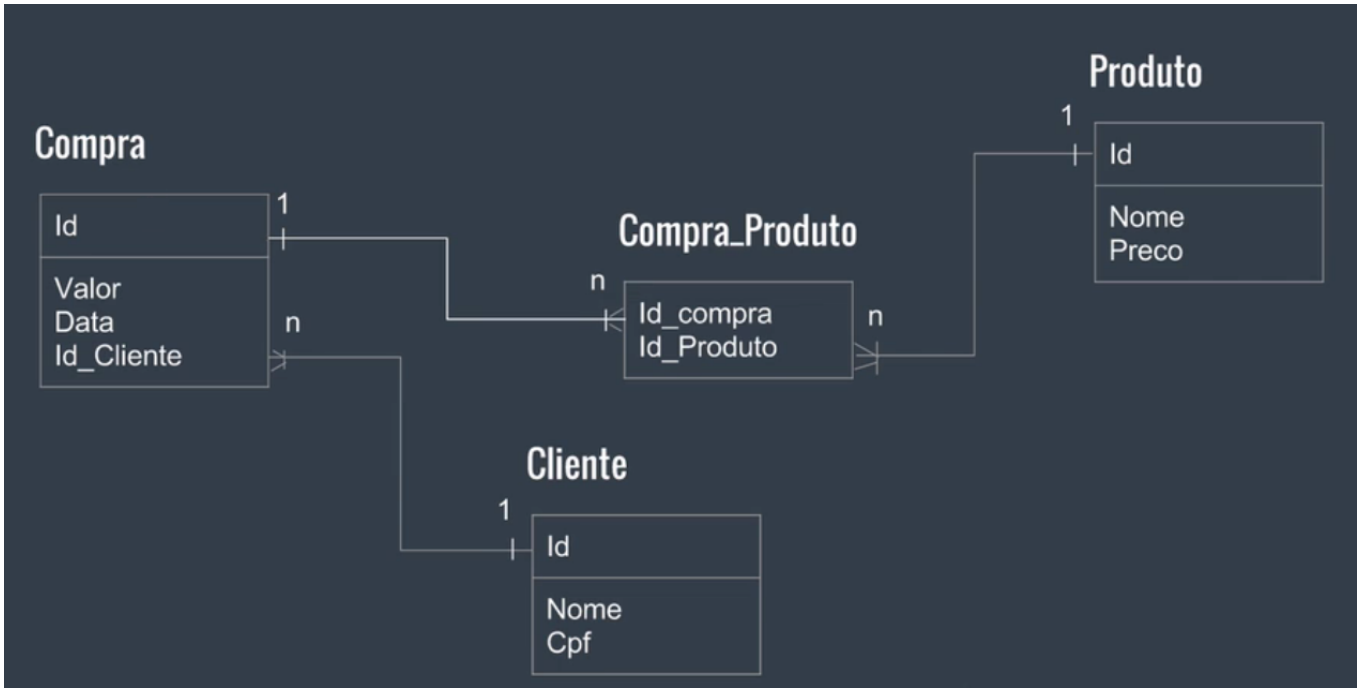
Logo, temos a entidade `Compra` e uma outra tabela que associa a compra com o produto. Mais acima, já havíamos definido o "Nome", a "categoria_id" e o "id" do produto.

Vamos tentar visualizar um pouco melhor todas as tabelas. Com elas se relacionam?

A tabela "Compra" se relaciona de forma *Many-to-One*, ou **1 para N**.



Além disso, tínhamos a tabela "Produto". Os produtos podem estar em várias compras e uma compra pode estar em vários produtos. Por isso, nós criamos o "Compra_Produto", que irá associar o `compra_id` com o `produto_id`. Neste caso, um produto pode estar em várias compras e uma compra pode estar em vários produtos.



O relacionamento entre compra e produto é o que chamamos de *Many-to-many. Isto significa que uma compra pode estar em vários produtos e um produto pode estar em várias compras diferentes. Quando isto acontece, nós precisamos de uma tabela associativa.

Estrutura Física

Vamos começar a falar sobre a estrutura física do nosso banco de dados. Até este ponto, nós discutimos como os dados serão organizados dentro do nosso banco. Mas não falamos sobre como isto acontece ou funciona.

Uma primeira questão é **Onde os dados são salvos?** Além disso, como o acesso é feito?

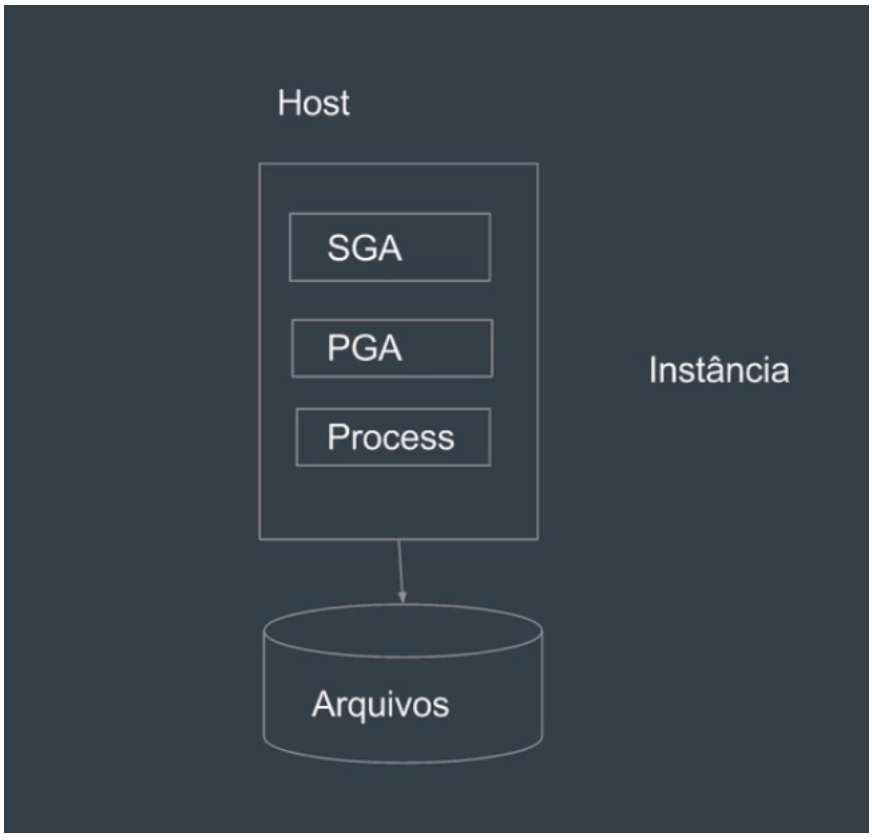
Se eu quero salvar uma lista de compras, a forma mais comum de fazermos isto usando um computador é anotar os itens no bloco de notas e depois, salvarmos em um arquivo. No banco de dados, não é grande a diferença. Na nossa estrutura, também salvaremos os dados em um arquivo.

Porém, ter os meus dados salvos, por exemplo em um *pendrive*, é suficiente para fazer um banco com as informações funcionar? Vamos pensar no que fazemos em um banco: nós consultamos, filtramos, ordenamos, inserimos e removemos registros, além de outras ações. Então, precisaremos de um software que manipule estes arquivos. Nós iremos rodá-lo em uma máquina, que irá cuidar dos arquivos.

Se tenho esta máquina, o software que irá manipular os arquivos terá uma área global da memória, que chamamos de *System Global Area* (**SGA**). Quando eu faço uma consulta no meu banco de dados e peço para ele trazer todos os registros desde um determinada data, a cada consulta executada, iremos iniciar um novo processo.

Também temos uma parte da memória que se chama *Process Global Area* (**PGA**), a área global para os processos. Toda vez que formos realizar uma consulta neste banco de dados, nós teremos uma parte da memória responsável por ter todas as escrituras, para conseguir atender a nossa consulta. Além disso, precisaremos de segurança no nosso banco de dados. Se o meu *Data Center* for incendiado, o que irá garantir que os dados não serão perdidos? Em geral, todo serviço que oferece um banco de dados ou você quando for gerenciar o seu banco de dados, irá querer garantir uma cópia de todos os seus arquivos. Para isto, geralmente, iremos configurar um *backup*. Neste software teremos um monte de **processos** que rodam por trás (*Background Process*) que são responsáveis por cuidar do *backup*, por exemplo.

Este conjunto de *System Global Area*, *Process Global Area* e *Process* ***é o que chamamos de*** *instância*.*



Instância é o software do nosso banco de dados, rodando para manipular os arquivos que têm as nossas informações.

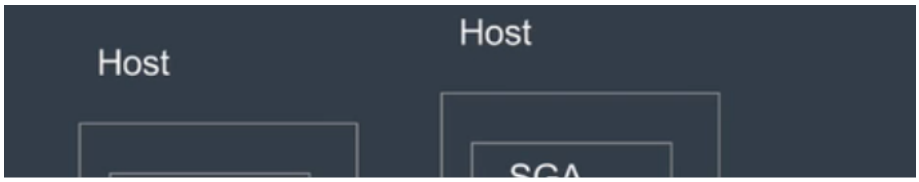
Em seguida, nós iremos discutir outras formas de termos um banco de dados Oracle rodando, sendo que a forma mais simples é a apresentada anteriormente: termos um *Host*, uma instância do Oracle rodando e os dados em um arquivo.

Real Application Clusters (RAC)

Continuando com o nosso curso, nós agora iremos apresentar uma outra forma de ter um banco de dados Oracle. Nós discutimos que os dados estão dentro de arquivos e que uma máquina fica rodando uma instância do meu banco Oracle para manipular estes arquivos.

Em seguida, vamos imaginar que eu tenho uma máquina rodando com um processador razoavelmente bom. Eu divulguei o meu software e comecei a ter 15 mil acessos diários. Porém, a minha máquina só suporta 10 mil acessos. O que acontecerá? O meu computador não irá suportar e começará a recusar as requisições. E quando o usuário for abrir o site do sistema, receberá a mensagem de que o banco não poderá ser carregado e não irá atender a requisição.

Como podemos fazer para o nosso software continuar disponível? Se uma máquina não suporta todo o trabalho, eu posso pegar uma segunda que também irá cuidar dos meus arquivos. Então, as duas serão responsáveis pelo mesmo arquivo.



Nós dissemos que temos duas instâncias rodando para manipular os arquivos. Porém, se o SGA ocupa uma área global da memória - e que todas as instâncias precisam conhecê-lo - o que as duas máquinas precisam fazer? Elas precisam compartilhar os dados entre si.

Observe que teremos a *System Global Area* sendo compartilhada.

Mas, e se mesmo com dois computadores, eu não conseguir atender todas as requisições? Sem problemas, adicionaremos mais um computador.

Usando este modelo para disponibilizar o nosso banco de dados, só conseguiremos mantê-lo funcionando enquanto tivermos dinheiro para investir em máquinas. Esta estrutura nós chamamos de ***Real Application Clusters***(RAC). Trata-se de um conjunto de máquinas para trabalhar em cima do meu banco de dados.

Quais vantagens encontramos nesta arquitetura? É que conseguiremos ter várias máquinas, trabalhando no mesmo banco de dados, e consequentemente, conseguimos aumentar a disponibilidade do nosso software, conseguimos atender mais requisições ao mesmo tempo. Porém, onde está o problema nisto? Colocar outras máquinas para rodar gera custos. Vamos imaginar o Facebook... Iremos supôr que ele tenha 100 máquinas rodando. É barato manter uma quantidade desta de computadores? Não. Então, aumentar o número de máquinas é uma forma de solucionarmos o problema, mas também aumentará os meus gastos com um servidor a mais.

Mais adiante, veremos outra arquitetura que também é possível trabalhar com o banco de dados Oracle 12c.

Compartilhado

Nós conversamos sobre o que acontece quando uma máquina recebe muitos acessos e já não suporta fazer o trabalho sozinha.

Vamos pensar em uma outra situação: temos os nossos arquivos e uma instância que cuida deles. Imagine que o meu banco de dados ocupa apenas 25% da capacidade do meu Host.

Eu tenho um computador potente, que aguenta bastante coisa, mas não estou aproveitando todos os recursos dele. O que isto significa?

Um computador com um bom processador é bastante caro. Se a maior parte do processador está sendo inutilizada, significa que muitos recursos estão ociosos. O que podemos fazer a respeito? Nós podemos rodar mais de uma instância por máquina.

Geralmente, este é o que acontece, quando contratamos um serviço de hospedagem, nós usamos um **servidor compartilhado**. Você tem apenas um servidor, mas dentro estão várias instâncias rodando independentemente entre si. Eles apenas compartilham o *hardware* daquele servidor.

E **quantas instância conseguimos colocar neste servidor?**

Podemos colocar a quantidade que a máquina aguentar. No nosso caso, conseguimos ter até quatro instâncias diferentes rodando.

Observe a diferença desta estrutura para o RAC (*Real Application Clusters*), quando temos um grupo de arquivos acessados por diversas máquinas. Nesta, encontramos a situação contrária, temos uma única máquina, com diversas bases de dados rodando dentro dela.

É por isso, que a minha *System Global Area* não é compartilhada. Neste caso, um banco de dados não precisa saber da existência do outro. São instância independentes, que compartilham o mesmo *hardware*.

Esta é outra forma de trabalhar com um banco de dados Oracle 12c. Em seguida, iremos falar sobre a nova arquitetura que chegou com a nova versão do Oracle, a **Multitenant**. Nós iremos estudar esta arquitetura com cuidado.