02

# Consultando os dados

## No papel, numa planilha? Armazenar dados é preciso

Um problema muito comum que temos é controlar nossos gastos durante o ano, por exemplo, saber se gastamos mais nas compras de Natal deste ano do que do ano passado. É claro, para que possamos realizar essas comparações precisamos armazenar os dados das compras de cada evento em algum lugar. Então, armazenar dados, para depois pesquisá-los e manipulá-los é uma necessidade comum no mercado de informática.

## O problema de acesso aos dados

Podemos armazenar informações através de planilhas eletrônicas, como o Excel, onde podemos ter algumas colunas tais como dia, valor e motivo da compra. No entanto, apesar de ser um ambiente interessante, o Excel se torna complexo quando precisamos extrair e manipular suas informações. E para dificultar ainda mais, podem haver outros sistemas interessados nos dados da planilha, que precisarão saber ler e converter dados neste formato.

## Softwares de banco de dados

Para simplificar esse trabalho, existem os softwares de bancos de dados, que nos permitem armazenar e manipular informações de uma maneira mais simples através de uma linguagem de manipulação de dados chamada **SQL** (**S**tructured **Q**uery **L**anguage), um padrão mundialmente utilizado. Para conseguirmos utilizar essa linguagem, precisamos instalar um software, o servidor de banco de dados, que nos permita armazenar essas informações, dos quais se destacam o MySQL, PostgreSQL, SQLServer e o Oracle, o banco que utilizaremos neste treinamento.

## Oracle Database

O Oracle Database é um software que pode ser instalado seguindo as maneiras tradicionais de cada sistema operacional. No primeiro exercício deste capítulo (https://cursos.alura.com.br/course/oracle-sql/section/1/task/3), você terá instruções de como configurá-lo em seu ambiente de desenvolvimento preferido. Esta é uma boa hora de fazê-lo antes de continuar. Vamos assumir a partir deste ponto que você realizou este primeiro exercício garantindo toda infraestrutura necessária para este treinamento.

## O terminal do Oracle

Com o Oracle instalado, precisamos nos comunicar com ele através do SQL*Plus, seu cliente de linha de comando. Apesar dele não ter a melhor interface gráfica, ele é igual para todas as plataformas, e não muda de versão para versão. É através dele que executaremos uma série de instruções SQL à medida que formos evoluindo no treinamento.

Para abrirmos o cliente do Oracle, abra o terminal do seu sistema operacional. No Windows, digite `cmd` no Executar. No Linux, abra o terminal. Nele, vamos entrar no Oracle:

```
sqlplus
```

Esse é o comando para nos conectarmos ao Oracle Database que está instalado na sua máquina. Basta digitar o usuário e senha que você criou no primeiro exercício do capítulo que pedimos para você fazer antes de continuar.

## Precisamos armazenar dados: nossa primeira tabela

Agora, vamos criar nossa primeira tabela. Tabela é como se fosse uma tabela mesmo, ou uma planilha do Excel, onde temos colunas, cada uma representando uma informação diferente, por exemplo, nome do produto, preço, etc, e linhas, onde cada linha é um dado em particular, por exemplo, geladeira, 50 reais.

```
| Id   | Valor | Observacoes                          |
| 1    | 50    | Geladeira branca com freezer         |
| 2    | 2000  | Notebook novo preto com tela touch screen |
```

Para usarmos o SQL para criar uma tabela, dizemos o mesmo em inglês, que queremos criar ( `create` ) uma tabela ( `table` ) chamada `compras` :

```
create table compras
```

Mas só isso é suficiente? O que queremos ter nessa tabela?

## Tabelas precisam de colunas, e colunas aceitam tipos de dados!

Precisamos saber o nome, uma descrição, uma observação de cada produto em nossa tabela de compras, para isso podemos utilizar um texto. Na linguagem do banco utilizaremos `varchar2(30)` para texto.

```
observacoes varchar2(30)
```

Também vamos precisar saber o preço, que é representado por um número, na linguagem SQL é chamado de `number` .

```
valor number, observacoes varchar2(30)
```

Vamos completar a tabela com mais algumas colunas. Queremos uma `data` para saber quando a compra foi executada e queremos saber se o valor já foi `recebido` . Para `data` utilizaremos o tipo `date` e para recebido utilizaremos char.

```
valor number, data date, observacoes varchar2(30), recebido char
```

Mas queremos que `recebido` só aceite dois valores, 0 e 1. O valor 0 para representar `false` e 1 para `true` . Para isso, vamos checar o valor de `recebido` , pois só podemos aceitar 0s e 1s, nenhum outro caractere além desses dois. Na linguagem do banco utilizaremos `check (recebido in (0,1))` para verificar essa particularidade.

```
valor number, data date, observacoes varchar2(30), recebido char check (recebido in (0,1))
```

Agora que já temos as colunas que queremos em nossa tabela, podemos passar a instrução que insere a tabela junto com a lista das colunas e seus respectivos tipos:

```
create table compras (
    valor number,
    data date,
```

```
    observacoes varchar2(30),
    recebido char check (recebido in (0,1))
);
```

Será que agora está completa?

## Identificadores únicos

Repare que podemos ter linhas em nossa tabela que contenham o mesmo valor. Como faríamos para diferenciar essa linhas?

Precisamos utilizar um `id` identificador único para cada linha. Pense no mundo real, o seu RG é uma **chave importante**, pois ele consegue identificar um brasileiro dentre todos os outros, ele é um número e é sequencial, alguém tinha o RG 1 e agora alguém tem o RG 223334446, essa chave é tão importante que chamamos de **chave primária**. A chave primária não precisa necessariamente ser sequencial, mas deve ser única: não devem existir duas pessoas com o mesmo RG, assim como não devem existir dois cursos com o mesmo código, dois produtos diferentes com o mesmo código de barra, etc. No nosso caso chamaremos essa chave de `id` , que é `primary key` , ou seja, chave primária.

```
create table compras (
    id number primary key,
    valor number,
    data date,
    observacoes varchar2(30),
    recebido char check (recebido in (0,1))
);
```

Além disso, queremos dizer que `id` é sequencial, cresce de um em um. Para isso criaremos um `sequence` para `id` :

```
create sequence id_seq;
```

Na hora que o `id_seq` for criado, ele assumirá o valor 0, e podemos incrementá-lo usando `id_seq.nextval` . Com isso garantiremos que, na hora de inserir um novo dado na tabela, o `id` não será repetido. Veremos isso mais a frente.

## Inserindo dados em nossa tabela

Queremos inserir na tabela `compras` , mas lembramos que a instrução do SQL é inglês, portanto inserimos ( `INSERT` ) na ( `INTO` ) tabela `compras` :

```
 INSERT INTO compras
```

Em seguida, passamos os valores que queremos inserir. Por exemplo, suponha uma compra de R$ 100,00, então falo que os valores que desejo inserir é o de 100 reais::

```
INSERT INTO compras VALUES (100.0);
```

Mas claro, não quero falar só o valor, desejo falar que ela foi efetuada no dia 12 de junho de 2007:

```
INSERT INTO compras VALUES (100.0, '12-JUN-2007');
```

Esse é o formato da data no Oracle, mas atenção ao o mês. No Oracle nós passamos as três primeiras letras do mês que desejamos, mas as letras têm que estar de acordo com a língua padrão do Oracle. Não sabe qual a língua padrão? Digite o comando:

```
show parameter nls_lang;
```

Se for padrão americano, passamos as três primeiras letras do mês em inglês; se for brasileiro, em português. Por exemplo, a data 13 de agosto de 2012 é, no padrão americano, '12-AUG-2012', e no padrão brasileiro é '12-AGO-2012'.

Voltando ao comando `INSERT`, queremos falar que a compra foi recebida e tem a observação 'COMPRAS DE JUNHO':

```
INSERT INTO compras VALUES (100.0, '12-JUN-2007', 'COMPRAS DE JUNHO', '1');
```

Separamos com vírgula cada um dos diferentes valores. Repare o uso de aspas simples (') quando o valor passado é uma data, string ou char. Veja também o valor `'1'` no lugar do campo `recebido` para indicar verdadeiro (colocaríamos `'0'` se quiséssemos passar falso).

Mas o Oracle não sabe que queremos colocar o valor 100.0 no campo `valor` ou 'COMPRAS DE JUNHO' no campo `observacoes`; precisamos passar essa informação pra ele, ou seja, passar o nome de cada coluna que tem um valor a ser inserido:

```
INSERT INTO COMPRAS (ID, VALOR, DATA, OBSERVACOES, RECEBIDO) VALUES (ID_SEQ.NEXTVAL, 100.0, '1
```

Lembra do `sequence` que criamos acima? É aqui que nós o utilizaremos, passando para a coluna `ID` o valor `ID_SEQ.NEXTVAL`. Mas o que isso significa? Significa que o valor do `id` sempre será o próximo valor da sequência `ID_SEQ` (Lembra que esse valor começa como 0?). Esse valor fica guardado no banco de dados, sendo incrementado de 1 em 1 sempre que chamamos o `.NEXTVAL`. Logo, no próximo `INSERT` que dermos na tabela `compras`, o valor da sequência passará a ser 2, e assim por diante .

Mas observe o comando inicial. `INSERT INTO` recebe o nome da tabela (no nosso caso `COMPRAS`). Em seguida, passamos a lista de colunas que vamos colocar valores: `(ID, VALOR, DATA, OBSERVACOES, RECEBIDO)`. Por fim, passamos os valores que gostaríamos de inserir, através do comando `VALUES`, fazendo: `VALUES (ID_SEQ.NEXTVAL, 100.0, '12-JUN-2007', 'COMPRAS DE JUNHO', '1')`. Todas instruções de `INSERT` são similares a essa: passamos o nome da tabela, o nome das colunas, e os valores a serem inseridos, separados por vírgula.

## Consultando nossa tabela

Pronto. Com essa compra já inserida nesse banco de dados, vamos começar a consultá-las. Se quiséssemos, por exemplo, ver todas as compras já cadastradas na minha lista, faríamos:

```
SELECT * FROM COMPRAS;
```

Entendendo o comando que acabamos de digitar: `SELECT` quer dizer que queremos executar a operação de seleção; `*` indica que queremos selecionar todas as colunas daquela tabela; `FROM COMPRAS` quer dizer que a tabela que queremos executar esse comando é a tabela `COMPRAS` .

Antes de começarmos a discutir sobre seleções usando SQL, vamos primeiro importar alguns dados para a tabela que criamos na seção anterior. Faça o download do arquivo .sql no exercício (https://cursos.alura.com.br/course/oracle-sql/section/1/task/5). Abra o arquivo, e veja que ele contém apenas um monte de INSERTs. Importe todos eles, dentro do Oracle Database, e digitando a instrução abaixo. Ela basicamente pegará todas as instruções que está no arquivo **cap1.sql** e mandará para o Oracle.

```
SQL> @DIRETORIO_DO_ARQUIVO_SQL/cap1.sql
```

Exemplo:

```
SQL> @/home/alura/Downloads/cap1.sql
```

Se quiséssemos selecionar apenas o `valor` e a `data` dessa tabela, faríamos:

```
SELECT VALOR, DATA FROM COMPRAS;
```

Podemos criar novas colunas a partir das já existentes. Por exemplo, sabemos que 20% do valor de cada produto é imposto que estamos pagando. Podemos calcular o valor de imposto de cada produto se multiplicarmos o `valor` por 0.20.

```
SELECT VALOR*0.20 FROM COMPRAS;
```

Podemos até dar um nome para essa nova coluna, por exemplo, `"imposto"` , usando a instrução `AS` :

```
SELECT VALOR, VALOR *0.20 AS IMPOSTOS FROM COMPRAS;
```

## Consultando com condições

Agora queremos recuperar os produtos mais caros, pois queremos controlar o nosso orçamento. Podemos definir a busca para trazer os resultados acima de um certo valor, para isso precisamos dos resultados que respeitem as condições que escolhermos. Por exemplo, queremos os resultados nos quais ( `where` ) o valor é maior que 1000.00 reais.

```
SELECT * FROM COMPRAS WHERE VALOR > 1000;
```

Podemos querer saber quantas compras foram feitas em um determinado ano. Por exemplo, imagine que precisamos saber as compras do primeiro semestre do ano de 2010 ou ( `or` ) as compras do mesmo período de 2012. Nesse caso, precisamos determinar o intervalo da data em que ocorreram as compras, data inicial e ( `and` ) data final:

```
SELECT * FROM COMPRAS WHERE (DATA > '01-JAN-2010' AND DATA < '01-JUL-2010') or (DATA > '01-JAN-
```

Repare no uso dos parênteses; eles indicam que a linha deve ser selecionada caso `DATA > '01-JAN-2010' AND DATA < '31-JUL-2010'`. Você pode montar a expressão que quiser, usando `AND`s, `OR`s. Além de maior e menor, a SQL suporta outros tipos de consultas, como por exemplo:

Maior ou igual:

```
SELECT * FROM COMPRAS WHERE VALOR >= 1000;
```

Menor ou igual:

```
SELECT * FROM COMPRAS WHERE VALOR <= 1000;
```

Diferente:

```
SELECT * FROM COMPRAS WHERE VALOR <> 1000;
```

Podemos filtrar também por textos. Se quiséssemos, por exemplo, buscar todas as compras cuja observação seja igual a "COMPRAS DE JANEIRO", podemos fazer:

```
SELECT * FROM COMPRAS WHERE OBSERVACOES = 'COMPRAS DE JANEIRO';
```

Repare o uso das aspas simples ('). Ela indica que o conteúdo que será passado é do tipo texto. O "igual" (=) irá buscar por linhas que possuem essa observação. Mas às vezes queremos buscar por apenas uma parte do texto.

## Consultando por parte do texto

Por exemplo, caso quiséssemos todas as compras cuja observação começasse com o texto "COMPRAS", independentemente do que viesse a seguir, deveríamos fazer:

```
SELECT * FROM COMPRAS WHERE OBSERVACOES LIKE 'COMPRAS%';
```

O caractere `%` funciona como um coringa, ou seja, não importa o que houver dali pra frente. Ele pode ser usado em qualquer lugar da consulta. Por exemplo, caso quiséssemos todas as compras cuja observação contenha a palavra "COMPRA" em qualquer lugar, faríamos:

```
SELECT * FROM COMPRAS WHERE OBSERVACOES LIKE '%COMPRAS%';
```

A instrução `SELECT` é poderosíssima. Boa sorte nos exercícios!