Capítulo 2

MEU PROBLEMA

Chegou o fim do mês e não sei direito onde meu dinheiro foi parar, no meu bolso não está. Nem na conta bancária. É um problema comum, e como podemos fazer para controlar os gastos?

A maneira mais simples de entender onde o dinheiro está indo é anotar todos os nossos gastos, sendo uma das mais tradicionais e antigas escrevê-los em um caderninho. Por exemplo dia 05/01/2016 gastei R$ 20 em uma Lanchonete, escrevo isso bonitinho em uma linha.

```
R$20 05/01/2016 Lanchonete
```

Tive um outro gasto de lanchonete no dia 06/01/2016 no valor de R$ 15, vou e anoto novamente:

```
R$20 05/01/2016 Lanchonete
R$15 06/01/2016 Lanchonete
```

Por fim, minha esposa comprou um guarda-roupa pro quarto e ele ainda não chegou, então anotei:

```
R$20 05/01/2016 Lanchonete
R$15 06/01/2016 Lanchonete
R$915,5 06/01/2016 Guarda-roupa (não recebi)
```

Repara que do jeito que anotei, as linhas acabam se assemelhando a uma tabelinha:

```
+---------+------------+--------------+--------------+
+ R$20    + 05/01/2016 + Lanchonete   + recebida     +
+ R$15    + 06/01/2016 + Lanchonete   + recebida     +
+ R$915,5 + 06/01/2016 + Guarda-roupa + não recebida +
+---------+------------+--------------+--------------+
```

Mas o que significa a primeira coluna dessa tabela mesmo? O valor gasto? Ok. E a segunda? É a data da compra? E a terceira, é o lugar que comprei ou são anotações relativas aos gastos? Repara que sem dar nome as colunas, minhas compras ficam confusas, a tabela fica estranha, e posso cada vez preencher com algo que *acho* que devo, ao invés de ter certeza do que cada campo significa. Como identificar cada um deles? Vamos dar um nome aos três campos que compõem a minha tabela: o valor, a data e as observações:

```
+---------+------------+--------------+--------------+
+ valor   + data       + observacoes  + recebida     +
+---------+------------+--------------+--------------+
+ R$20    + 05/01/2016 + Lanchonete   + recebida     +
+ R$15    + 06/01/2016 + Lanchonete   + recebida     +
+ R$915,5 + 06/01/2016 + Guarda-roupa + não recebida +
+---------+------------+--------------+--------------+
```

Neste curso utilizaremos o MySQL, um SGBD gratuito que pode ser instalado seguindo as maneiras tradicionais de cada sistema operacional. Por exemplo, no Windows, o processo é realizado através do download de um arquivo .msi, enquanto no Linux (versões baseadas em Debian), pode ser feito através de um simples comando apt-get, e no MacOS pode ser instalado com um pacote .dmg. Baixe o seu instalador em http://dev.mysql.com/downloads/mysql/

2.1 CRIANDO O NOSSO BANCO DE DADOS

Durante todo o curso usaremos o terminal do MySQL. Existe também a interface gráfica do Workbench do MySQL, que não utilizaremos. Ao utilizarmos o terminal não nos preocupamos com o aprendizado de mais uma ferramenta que é opcional, podendo nos concentrar no SQL, que é o objetivo deste curso.

Abra o terminal do seu sistema operacional. No Windows, digite `cmd` no Executar. No Mac e Linux, abra o terminal. Nele, vamos entrar no MySQL usando o usuário *root*:

```
mysql -uroot -p
```

O `-u` indica o usuário `root`, e o `-p` é porque digitaremos a senha. Use a senha que definiu durante a instalação do MySQL, note que por padrão a senha pode ser em branco e, nesse caso, basta pressionar enter.

2.2 COMEÇANDO UM CADERNO NOVO: CRIANDO O BANCO

Agora que estamos conectados ao MySQL precisamos dizer a ele que queremos um caderno novo. Começaremos com um que gerenciará todos os dados relativos aos nossos gastos, permitindo que controlemos melhor nossos gastos, portanto quero um banco de dados chamado *ControleDeGastos*, pode parecer estranho, mas não existe um padrão para nomear um banco de dados, por isso utilizamos o

nome de um banco de dados, uma tabela, uma coluna ou qualquer outro tipo de palavra que é uma escolha livre minha, como usuário do banco.

2.4 A TABELA DE COMPRAS

Agora que já tenho um caderno, já estou louco para escrever o meu primeiro gasto, que foi numa lanchonete. Por exemplo, peguei uma parte do meu caderno, de verdade, de 2016:

```
+---------+------------+--------------+--------------+
+ valor   + data       + observacoes  + recebida     +
+---------+------------+--------------+--------------+
+ R$20    + 05/01/2016 + Lanchonete   + sim          +
+ R$15    + 06/01/2016 + Lanchonete   + sim          +
+ R$915,5 + 06/01/2016 + Guarda-roupa + não          +
+     ... +        ... + ...          + ...          +
+---------+------------+--------------+--------------+
```

Ok. Quero colocar a minha primeira anotação no banco. Mas se o banco é o caderno... como que o banco sabe que existe uma tabela de valor, data e observações? Aliás, até mesmo quando comecei com o meu caderno, fui eu, Guilherme, que tive que desenhar a tabela com suas três coluninhas em cada uma das páginas que eu quis colocar seus dados.

Então vamos fazer a mesma coisa com o banco. Vamos dizer a ele que queremos ter uma tabela: descrevemos a estrutura dela através de um comando SQL. Nesse comando devemos dizer quais são os campos (valor, data e observações) que serão utilizados, para que ele separe o espaço específico para cada um deles toda vez que inserimos uma nova compra, toda vez que registramos um novo dado (um registro novo).

Para criar uma tabela, novamente falamos inglês, por favor senhor MySQL, crie uma tabela (`CREATE TABLE`) chamada *compras*:

```
mysql> CREATE TABLE compras;
ERROR 1113 (42000): A table must have at least 1 column
```

Como assim erro 1113? Logo após o código do erro, o banco nos informou que toda tabela deve conter pelo menos uma coluna. Verdade, não falei as colunas que queria criar. Vamos olhar novamente nossas informações:

```
+---------+------------+--------------+--------------+
+ valor   + data       + observacoes  + recebida     +
+---------+------------+--------------+--------------+
+ R$20    + 05/01/2016 + Lanchonete   + sim          +
+ R$15    + 06/01/2016 + Lanchonete   + sim          +
+ R$915,5 + 06/01/2016 + Guarda-roupa + não          +
+     ... +        ... + ...          + ...          +
+---------+------------+--------------+--------------+
```

A estrutura da tabela é bem clara: são 4 campos distintos, valor que é um número com ponto decimal, data é uma data, observações que é um texto livre e recebida que é ou sim ou não.

```
mysql> CREATE TABLE compras(
valor DECIMAL(18,2),
data DATE,
observacoes VARCHAR(255),
recebida TINYINT);
Query OK, 0 rows affected (0.04 sec)
```

A tabela *compras* foi criada com sucesso.

2.5 CONFERINDO A EXISTÊNCIA DE UMA TABELA

Mas você confia em tudo que todo mundo fala? Vamos ver se realmente a tabela foi criada em nosso banco de dados. Se você possuisse um caderninho de tabelas, o que eu poderia fazer para pedir para me mostrar sua tabela de compras? Por favor, me descreve sua tabela de compras?

```
mysql> desc compras;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| valor       | decimal(18,2) | YES  |     | NULL    |       |
| data        | date          | YES  |     | NULL    |       |
| observacoes | varchar(255)  | YES  |     | NULL    |       |
| recebida    | tinyint(4)    | YES  |     | NULL    |       |
+-------------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

E aí está nossa tabela de compras. Ela possui 4 colunas, que são chamados de *campos* (*fields*). Cada campo é de um tipo diferente (*decimal*, *date*, *varchar* e *tinyint*) e possui até mesmo informações extras que utilizaremos no decorrer do curso!

2.6 INSERINDO REGISTROS NO BANCO DE DADOS

Chegou a hora de usar nossa tabela: queremos inserir dados, então começamos pedindo para inserir algo em *compras*:

```
INSERT INTO compras
```

Não vamos cometer o mesmo erro da última vez. Se desejamos fazer algo com os 4 campos da tabela, temos que falar os valores que desejamos adicionar. Por exemplo, vamos pegar as informações da primeira linha da planilha:

valor	data	observacoes	recebida
20	30/05/2015	Lanchonete	sim

Figura 2.2: Planilha exemplo

Então nós temos uma compra com valor *20*, observação *Lanchonete*, data *05/01/2016*, e que foi recebida com sucesso (*1*):

```
mysql> INSERT INTO compras VALUES (20, 'Lanchonete', '05/01/2016', 1);
ERROR 1292 (22007): Incorrect date value: 'Lanchonete' for column 'data' at row 1
```

Parece que nosso sistema pirou. Ele achou que a *Lanchonete* era a data. *Lanchonete* era o campo *observacoes*. Mas... como ele poderia saber isso? Eu não mencionei para ele a ordem dos dados, então ele assumiu uma ordem determinada, uma ordem que eu não prestei atenção, mas foi fornecida quando executamos o `DESC compras` :

```
| valor       | decimal(18,2) | YES  |     | NULL    |       |
| data        | date          | YES  |     | NULL    |       |
| observacoes | varchar(255)  | YES  |     | NULL    |       |
| recebida    | tinyint(4)    | YES  |     | NULL    |       |
```

Tentamos novamente, agora na ordem correta dos campos:

```
mysql> INSERT INTO compras VALUES (20, '05/01/2016', 'Lanchonete', 1);
ERROR 1292 (22007): Incorrect date value: '05/01/2016' for column 'data' at row 1
```

Outro erro? Agora ele está dizendo que o valor para data está incorreto, porém, aqui no Brasil, usamos esse formato para datas... O que faremos agora? Que tal tentarmos utilizar o formato que já vem por padrão no MySQL que é `ano-mês-dia` ?

Vamos tentar mais uma vez, porém, dessa vez, com a data '2016-01-05':

```
mysql> INSERT INTO compras VALUES (20, '2016-01-05', 'Lanchonete', 1);
Query OK, 1 row affected (0.01 sec)
```

Agora sim os dados foram inseridos com sucesso. Temos um novo registro em nossa tabela. Mas porque será que o MySQL adotou o formato `ano-mês-dia` ao invés de `dia/mês/ano` ? O formato `ano-mês-dia` é utilizado pelo padrão SQL, ou seja, por questões de padronização, ele é o formato mais adequado para que funcione para todos.

2.7 SELECÃO SIMPLES

Como fazemos para conferir se ele foi inserido? Pedimos para o sistema de banco de dados

Vamos dar uma olhada dentro de *VALUES*?

```
(915,5, '2016-01-06', 'Guarda-roupa', 0)
```

A vírgula apareceu 4 vezes, portanto teríamos 5 campos diferentes! Repare que o valor `915,5` está formatado no estilo português do Brasil, diferente do estilo inglês americano, que usa ponto para definir o valor decimal, como em `915.5`. Alteramos e executamos:

```
mysql> INSERT INTO compras VALUES (915.5, '2016-01-06', 'Guarda-roupa', 0);
Query OK, 1 row affected (0.00 sec)
```

Podemos conferir os 3 registros que inserimos através do `SELECT` que fizemos antes:

```
mysql> SELECT * FROM compras;
+--------+------------+--------------+----------+
| valor  | data       | observacoes  | recebida |
+--------+------------+--------------+----------+
|  20.00 | 2016-01-05 | Lanchonete   |        1 |
|  15.00 | 2016-01-06 | Lanchonete   |        1 |
| 915.50 | 2016-01-06 | Guarda-roupa |        0 |
+--------+------------+--------------+----------+
3 rows in set (0.01 sec)
```

Mas e se eu quisesse inserir os dados em uma ordem diferente? Por exemplo, primeiro informar a `data`, depois as `observacoes`, `valor`, e por fim se foi `recebida` ou não? Será que podemos fazer isso? Quando estamos fazendo um `INSERT`, podemos informar o que estamos querendo inserir por meio de parênteses:

```
mysql> INSERT INTO compras (data, observacoes, valor, recebida)
```

Note que agora estamos informando **explicitamente** a ordem dos valores que informaremos após a instrução `VALUES`:

```
mysql> INSERT INTO compras (data, observacoes, valor, recebida)
VALUES ('2016-01-10', 'Smartphone', 949.99, 0);
Query OK, 1 row affected (0,00 sec)
```

Se consultarmos a nossa tabela:

```
mysql> SELECT * FROM compras;
+--------+------------+--------------+----------+
| valor  | data       | observacoes  | recebida |
+--------+------------+--------------+----------+
|  20.00 | 2016-01-05 | Lanchonete   |        1 |
|  15.00 | 2016-01-06 | Lanchonete   |        1 |
| 915.50 | 2016-01-06 | Guarda-roupa |        0 |
| 949.99 | 2016-01-10 | Smartphone   |        0 |
+--------+------------+--------------+----------+
4 rows in set (0,00 sec)
```

A nossa compra foi inserida corretamente, porém com uma ordem diferente da estrutura da tabela.

2.9 A CHAVE PRIMÁRIA

2.11 CONSULTAS COM FILTROS

Aprendemos a criar as nossas tabelas e inserir registros nelas, então vamos adicionar todas as nossas compras. Eu já tenho um arquivo com as minhas compras irei executá-lo, não se preocupe, nos exercícios forneceremos o link do arquivo. Saia do terminal com o comando `exit` :

```
mysql> exit
```

Execute o comando:

```
mysql -uroot -p ControleDeGastos < compras.sql
```

Agora todas as compras foram registradas no nosso banco de dados! Vamos consultar todas as compras novamente:

```
mysql> SELECT * FROM compras;
+----+----------+------------+------------------------------+----------+
| id | valor    | data       | observacoes                  | recebida |
+----+----------+------------+------------------------------+----------+
|  1 |    20.00 | 2016-01-05 | Lanchonete                   |        1 |
|  2 |    15.00 | 2016-01-06 | Lanchonete                   |        1 |
|  3 |   915.50 | 2016-01-06 | Guarda-roupa                 |        0 |
|  4 |   949.99 | 2016-01-10 | Smartphone                   |        0 |
|  5 |   200.00 | 2012-02-19 | Material escolar             |        1 |
|  6 |  3500.00 | 2012-05-21 | Televisao                    |        0 |
|  7 |  1576.40 | 2012-04-30 | Material de construcao       |        1 |
|  8 |   163.45 | 2012-12-15 | Pizza pra familia            |        1 |
|  9 |  4780.00 | 2013-01-23 | Sala de estar                |        1 |
| 10 |   392.15 | 2013-03-03 | Quartos                      |        1 |
| 11 |  1203.00 | 2013-03-18 | Quartos                      |        1 |
| 12 |   402.90 | 2013-03-21 | Copa                         |        1 |
| 13 |    54.98 | 2013-04-12 | Lanchonete                   |        0 |
| 14 |    12.34 | 2013-05-23 | Lanchonete                   |        0 |
| 15 |    78.65 | 2013-12-04 | Lanchonete                   |        0 |
| 16 |    12.39 | 2013-01-06 | Sorvete no parque            |        0 |
| 17 |    98.12 | 2013-07-09 | Hopi Hari                    |        1 |
| 18 |  2498.00 | 2013-01-12 | Compras de janeiro           |        1 |
```

```
| 15 | 78.65 | 2013-12-04 | Lanchonete  |        0 |
| 37 | 32.09 | 2015-07-02 | Lanchonete  |        1 |
| 39 | 98.70 | 2015-02-07 | Lanchonete  |        1 |
+----+-------+------------+-------------+----------+
7 rows in set (0,00 sec)
```

Agora preciso saber todas as minhas compras que foram **Parcelas**. Então novamente eu usarei o mesmo filtro, porém para Parcelas:

```
mysql> SELECT * FROM compras WHERE observacoes = 'Parcelas';
Empty set (0,00 sec)
```

Que estranho eu lembro de ter algum registro de parcela... Sim, existem registros de parcelas, porém não existe apenas uma observação "Parcela" e sim "Parcela do carro" ou "Parcela da casa", ou seja, precisamos filtrar as observações verificando se existe um **pedaço** do texto que queremos na coluna desejada. Para verificar um pedaço do texto utilizamos o argumento `LIKE` :

```
mysql> SELECT * FROM compras WHERE observacoes LIKE 'Parcela%';
+----+---------+------------+------------------+----------+
| id | valor   | data       | observacoes      | recebida |
+----+---------+------------+------------------+----------+
| 27 | 1709.00 | 2014-08-25 | Parcela da casa  |        0 |
| 28 |  567.09 | 2014-09-25 | Parcela do carro |        0 |
+----+---------+------------+------------------+----------+
2 rows in set (0,00 sec)
```

Perceba que utilizamos o "%". Quando adicionamos o "%" durante um filtro utilizando o `LIKE` significa que queremos todos os registros que iniciem com Parcela e que tenha qualquer tipo de informação a direita, ou seja, se a observação for "Parcela da casa" ou "Parcela de qualquer coisa" ele retornará para nós, mas suponhamos que quiséssemos saber todas as compras com observações em que o "de" estivesse no meio do texto? Bastaria adicionar o "%" tanto no início quanto no final:

```
mysql> SELECT * FROM compras WHERE observacoes LIKE '%de%';
+----+---------+------------+------------------------------+----------+
| id | valor   | data       | observacoes                  | recebida |
+----+---------+------------+------------------------------+----------+
|  7 | 1576.40 | 2012-04-30 | Material de construcao       |        1 |
|  9 | 4780.00 | 2013-01-23 | Sala de estar                |        1 |
| 18 | 2498.00 | 2013-01-12 | Compras de janeiro           |        1 |
| 20 |  223.09 | 2013-12-17 | Compras de natal             |        1 |
| 32 |  434.00 | 2014-04-01 | Rodeio interior de Sao Paulo |        0 |
| 36 |  370.15 | 2014-12-25 | Compras de natal             |        0 |
+----+---------+------------+------------------------------+----------+
6 rows in set (0,00 sec)
```

Veja que agora foram devolvidas todas as compras que possuem um "DE" na coluna `observacoes` independente de onde esteja.

2.12 MODELANDO TABELAS

Mas como foi que chegamos na tabela de compras mesmo? Começamos com uma idéia dos dados que gostaríamos de armazenar: nossas compras. Cada uma delas tem um valor, uma data, uma

```
- nomeDaMae
- telefoneDoPai
- telefoneDaMae
```

Um outro detalhe que sumiu é a questão da vacinação. A diretora gosta de conversar com os pais para indicar quando é época de vacinação. Ela sabe da importância do efeito de vacinação em grupo (todos devem ser vacinados, se um não é, existe chance de infectar muitos). Por isso ela sempre envia cartas (campo endereco) para os pais (campo nomeDoPai e nomeDaMae). Mas... como saber qual vacina ele precisa tomar? Depende da idade, sugerimos então armazenar a data de nascimento do aluno:

```
Aluno
- id
- nascimento
- nome
- serie
- sala
- email
- telefone
- endereco
- nomeDoPai
- nomeDaMae
- telefoneDoPai
- telefoneDaMae
```

Além disso, ela comenta que este ano o aluno está na terceira série, mas ano que vem estará na quarta série! Tanto a série quanto a sala são valores que mudam anualmente, e precisamos atualizá-los sempre, quase que todos os alunos tem sua série e sala atualizadas de uma vez só (algumas escolas permitem a troca de salas no meio do ano letivo). Para deixar isso claro, alteramos nossos campos, refletindo que tanto a série quanto a sala são atuais:

```
Aluno
- id
- nascimento
- nome
- serieAtual
- salaAtual
- email
- telefone
- endereco
- nomeDoPai
- nomeDaMae
- telefoneDoPai
- telefoneDaMae
```

Esse nosso processo de criar, de moldar, de modelar uma tabela é o que chamamos de modelagem de dados. Na verdade estamos modelando a estrutura que será capaz de receber os dados. E não existem regras 100% fixas nesse processo, por isso mesmo é importantíssimo levantarmos as necessidades, os requisitos dos nossos clientes junto a eles. Conversando com eles, vendo o trabalho deles no dia a dia, entendemos o que eles precisam e como modelar essas informações no banco. E mesmo tendo modelado uma vez, pode ser que daqui um ano temos que evoluir nosso modelo.

2.13 RESUMINDO

Capítulo 3

ATUALIZANDO E EXCLUINDO DADOS

Cadastramos todas as nossas compras no banco de dados, porém, no meu rascunho onde eu coloco todas as minhas compras alguns valores não estão batendo. Vamos selecionar o valor e a observação de todas as compras com valores entre 1000 e 2000:

```
mysql> SELECT valor, observacoes FROM compras
WHERE valor >= 1000 AND valor <= 2000;
+---------+-----------------------+
| valor   | observacoes           |
+---------+-----------------------+
| 1576.40 | Material de construcao |
| 1203.00 | Quartos               |
| 1501.00 | Presente da sogra     |
| 1709.00 | Parcela da casa       |
| 1245.20 | Roupas                |
+---------+-----------------------+
5 rows in set (0,00 sec)
```

Fizemos um filtro para um intervalo de valor, ou seja, **entre** 1000 e 2000. Em SQL, quando queremos filtrar um intervalo, podemos utilizar o operador `BETWEEN` :

```
mysql> SELECT valor, observacoes FROM compras
WHERE valor BETWEEN 1000 AND 2000;
+---------+-----------------------+
| valor   | observacoes           |
+---------+-----------------------+
| 1576.40 | Material de construcao |
| 1203.00 | Quartos               |
| 1501.00 | Presente da sogra     |
| 1709.00 | Parcela da casa       |
| 1245.20 | Roupas                |
+---------+-----------------------+
5 rows in set (0,00 sec)
```

O resultado é o mesmo que a nossa *query* anterior, porém agora está mais clara e intuitura.

No meu rascunho informa que a compra foi no ano de 2013, porém não foi feito esse filtro. Então vamos adicionar mais um filtro pegando o intervalo de 01/01/2013 e 31/12/2013:

```
mysql> SELECT valor, observacoes FROM compras
WHERE valor BETWEEN 1000 AND 2000
AND data BETWEEN '2013-01-01' AND '2013-12-31';
+---------+-------------+
| valor   | observacoes |
+---------+-------------+
| 1203.00 | Quartos     |
+---------+-------------+
1 row in set (0,00 sec)
```

3.1 UTILIZANDO O UPDATE

Encontrei a compra que foi feita com o valor errado, no meu rascunho a compra de Quartos o valor é de 1500. Então agora precisamos **alterar** esse valor na nossa tabela. Para alterar/atualizar valores em SQL utilizamos a instrução `UPDATE`:

```
UPDATE compras
```

Agora precisamos adicionar o que queremos alterar por meio da instrução `SET`:

```
UPDATE compras SET valor = 1500;
```

Precisamos sempre **tomar cuidado** com a instrução `UPDATE`, pois o exemplo acima executa normalmente, porém o que ele vai atualizar? Não informamos em momento algum qual compra precisa ser atualizar, ou seja, ele iria **atualizar TODAS as compras** e não é isso que queremos. Antes de executar o `UPDATE`, precisamos verificar o `id` da compra que queremos alterar:

```
mysql> SELECT id, valor, observacoes FROM compras
WHERE valor BETWEEN 1000 AND 2000
AND data BETWEEN '2013-01-01' AND '2013-12-31';
+----+---------+-------------+
| id | valor   | observacoes |
+----+---------+-------------+
| 11 | 1203.00 | Quartos     |
+----+---------+-------------+
1 row in set (0,00 sec)
```

Agora que sabemos qual é o `id` da compra, basta informá-lo por meio da instrução `WHERE`:

```
mysql> UPDATE compras SET valor = 1500 WHERE id = 11;
Query OK, 1 row affected (0,01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Vamos verificar se o valor foi atualizado:

```
mysql> SELECT valor, observacoes FROM compras
WHERE valor BETWEEN 1000 AND 2000
AND data BETWEEN '2013-01-01' AND '2013-12-31';
+---------+-------------+
| valor   | observacoes |
+---------+-------------+
| 1500.00 | Quartos     |
+---------+-------------+
1 row in set (0,00 sec)
```

Analisando melhor essa compra eu sei que se refere aos móveis do quarto, mas uma observação como "Quartos" que dizer o que? Não é clara o suficiente, pode ser que daqui a 6 meses, 1 ano ou mais tempo eu nem saiba mais o que se refere essa observação e vou acabar pensando: "Será que eu comprei móveis ou comprei um quarto todo?". Um tanto confuso... Então vamos alterar também as observações e deixar algo mais claro, como por exemplo: "Reforma de quartos".

```
UPDATE compra SET observacoes = 'Reforma de quartos' WHERE id = 11;
```

Verificando a nova observação:

3.3 UTILIZANDO UMA COLUNA COMO REFERÊNCIA PARA OUTRA COLUNA

Esqueci de adicionar 10% de imposto que paguei na compra de id 11... portanto quero fazer algo como:

```
mysql> SELECT valor FROM compras WHERE id = 11;
+---------+
| valor   |
+---------+
| 1500.00 |
+---------+
1 row in set (0,00 sec)
```

Descobri o valor atual, agora multiplico por `1.1` para aumentar `10%`: `1500 * 1.1` são `1650` reais. Então atualizo agora manualmente:

```
UPDATE compras SET valor = 1650 AND observacoes = 'Reforma de quartos novos' WHERE id = 11;
```

Mas e se eu quisesse fazer o mesmo para diversas linhas?

```
mysql> SELECT valor FROM compras WHERE id > 11 AND id <= 14;
+--------+
| valor  |
+--------+
| 402.90 |
|  54.98 |
|  12.34 |
+--------+
3 rows in set (0,00 sec)

UPDATE .... e agora???
```

E agora? Vou ter que calcular na mão cada um dos casos, com 10% a mais, e fazer uma linha de `UPDATE` para cada um? A solução é fazer um único `UPDATE` em que digo que quero alterar o valor para o valor dele mesmo, vezes os meus `1.1` que já queria antes:

```
UPDATE compras SET valor = valor * 1.1
WHERE id >= 11 AND id <= 14;
```

Nessa query fica claro que eu posso usar o valor de um campo (qualquer) para atualizar um campo da mesma linha. No nosso caso usamos o valor original para calcular o novo valor. Mas estou livre para usar outros campos da mesma linha, desde que faça sentido para o meu problema, claro. Por exemplo, se eu tenho uma tabela de `produtos` com os campos `precoLiquido` eu posso atualizar o `precoBruto` quando o imposto mudar para 15%:

```
UPDATE produtos SET precoBruto = precoLiquido * 1.15;
```

3.4 UTILIZANDO O DELETE

Observei o meu rascunho e percebi que essa compra eu não devia ter sido cadastrada na minha tabela de compras, ou seja, eu preciso excluir esse registro do meu banco de dados. Em SQL, quando

queremos excluir algum registro, utilizamos a instrução `DELETE`:

```
DELETE FROM compras;
```

O `DELETE` tem o comportamento similar ao `UPDATE`, ou seja, **precisamos sempre tomar cuidado** quando queremos excluir algum registro da tabela. Da mesma forma que fizemos com o `UPDATE`, precisamos adicionar a instrução `WHERE` para informa o que queremos excluir, justamente para **evitamos a exclusão de todos os dados**:

```
mysql> DELETE FROM compras WHERE id = 11;
Query OK, 1 row affected (0,01 sec)
```

Se verificarmos novamente se existe o registro dessa compra:

```
mysql> SELECT valor, observacoes FROM compras
WHERE valor BETWEEN 1000 AND 2000
AND data BETWEEN '2013-01-01' AND '2013-12-31';
Empty set (0,00 sec)
```

A compra foi excluída conforme o esperado!

3.5 CUIDADOS COM O DELETE E UPDATE

Vimos como o `DELETE` e `UPDATE` podem ser **perigosos** em banco de dados, se não definirmos uma condição para cada uma dessas instruções podemos excluir/alterar **TODAS** as informações da nossa tabela. A boa prática para executar essas instruções é **sempre** escrever a instrução `WHERE` antes, ou seja, definir primeiro qual será a condição para executar essas instruções:

```
WHERE id = 11;
```

Então adicionamos o `DELETE` / `UPDATE`:

```
DELETE FROM compras WHERE id = 11;
```

```
UPDATE compras SET valor = 1500 WHERE id = 11;
```

Dessa forma garantimos que o nosso banco de dados não tenha risco.

3.6 RESUMINDO

Nesse capítulo aprendemos como fazer *queries* por meio de intervalos utilizando o `BETWEEN` e como alterar/excluir os dados da nossa tabela utilizando o `DELETE` e o `UPDATE`. Vamos para os exercícios?

EXERCÍCIOS

1. Altere as compras, colocando a observação 'preparando o natal' para todas as que foram efetuadas no dia 20/12/2014.
2. Altere o VALOR das compras feitas antes de 01/06/2013. Some R$10,00 ao valor atual.
3. Atualize todas as compras feitas entre 01/07/2013 e 01/07/2014 para que elas tenham a observação 'entregue antes de 2014' e a coluna recebida com o valor TRUE.
4. Em um comando WHERE é possível especificar um intervalo de valores. Para tanto, é preciso dizer qual o valor mínimo e o valor máximo que define o intervalo. Qual é o operador que é usado para isso?
5. Qual operador você usa para remover linhas de compras de sua tabela?
6. Exclua as compras realizadas entre 05 e 20 março de 2013.
7. Existe um operador lógico chamado `NOT`. Esse operador pode ser usado para negar qualquer condição. Por exemplo, para selecionar qualquer registro com data diferente de 03/11/2014, pode ser construído o seguinte `WHERE`:

```
WHERE NOT DATA = '2011-11-03'
```

Use o operador `NOT` e monte um `SELECT` que retorna todas as compras com valor diferente de R$ 108,00.

CAPÍTULO 4

ALTERANDO E RESTRINGINDO O FORMATO DE NOSSAS TABELAS

Muitas vezes que inserimos dados em um banco de dados, precisamos saber quais são os tipos de dados de cada coluna da tabela que queremos popular. Vamos dar uma olhada novamente na estrutura da nossa tabela por meio da instrução `DESC`:

```
mysql> DESC compras;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| id          | int(11)       | NO   | PRI | NULL    | auto_increment |
| valor       | decimal(18,2) | YES  |     | NULL    |                |
| data        | date          | YES  |     | NULL    |                |
| observacoes | varchar(255)  | YES  |     | NULL    |                |
| recebida    | tinyint(4)    | YES  |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
5 rows in set (0,00 sec)
```

Como já vimos, o MySQL demonstra algumas características das colunas da nossa tabela. Vamos verificar a coluna `Null`, o que será que ela significa? Será que está dizendo que nossas colunas aceitam valores vazios? Então vamos tentar inserir uma compra com `observacoes` **vazia**, ou seja, **nula**, com o valor *null*:

```
INSERT INTO compras (valor, data, recebida, observacoes)
VALUES (150, '2016-01-04', 1, NULL);
Query OK, 1 row affected (0,01 sec)
```

Vamos verificar o resultado:

```
mysql> SELECT * FROM compras WHERE data = '2016-01-04';
+----+--------+------------+-------------+----------+
| id | valor  | data       | observacoes | recebida |
+----+--------+------------+-------------+----------+
| 47 | 150.00 | 2016-01-04 | NULL        |        1 |
+----+--------+------------+-------------+----------+
1 row in set (0,00 sec)
```

O nosso banco de dados permitiu o registro de uma compra sem observação. Mas em qual momento informamos ao MySQL que queriámos valores nulos? Em nenhum momento! Porém, quando criamos uma tabela no MySQL e não informamos se queremos ou não valores nulos para uma determinada coluna, por padrão, o MySQL aceita os valores nulos.

Note que um texto vazio é diferente de não ter nada! É diferente de um texto vazio como ''. Nulo é

nada, é a inexistência de um valor. Um texto sem caracteres é um texto com zero caracteres. Um valor nulo nem texto é, nem nada é. Nulo é o não ser. Uma questão filosófica milenar, discutida entre os melhores filósofos e programadores, além de ser fonte de muitos bugs. Cuidado.

4.1 RESTRINGINDO OS NULOS

Para resolver esse problema podemos criar restrições, *Constraints*, que tem a capacidade de determinar as regras que as colunas de nossas tabelas terão. Antes de configurar o *Constraints*, vamos verificar todos os registros que tiverem observações nulas e vamos apagá-los. Queremos selecionar todas as observações que são nulas, são nulas, SÃO NULAS, `IS NULL` :

```
mysql> SELECT * FROM compras WHERE observacoes IS NULL;
+----+--------+------------+-------------+----------+
| id | valor  | data       | observacoes | recebida |
+----+--------+------------+-------------+----------+
| 47 | 150.00 | 2016-01-04 | NULL        |        1 |
+----+--------+------------+-------------+----------+
1 row in set (0,00 sec)
```

Vamos excluir todas as compras que tenham as observações nulas:

```
DELETE FROM compras WHERE observacoes IS NULL;
Query OK, 1 row affected (0,01 sec)
```


4.2 ADICIONANDO CONSTRAINTS

Podemos definir *Constraints* no momento da criação da tabela, como no caso de definir que nosso valor e data não podem ser nulos (`NOT NULL`):

```
CREATE TABLE compras(
id INT AUTO_INCREMENT PRIMARY KEY,
valor DECIMAL(18,2) NOT NULL,
data DATE NOT NULL,
```

...

Porém, a tabela já existe! E não é uma boa prática excluir a tabela e cria-la novamente, pois podemos **perder registros**! Para fazer alterações na estrutura da tabela, podemos utilizar a instrução `ALTER TABLE` que vimos antes:

```
ALTER TABLE compras;
```

Precisamos especificar o que queremos fazer na tabela, nesse caso modificar uma coluna e adicionar uma *Constraints*:

```
mysql> ALTER TABLE compras MODIFY COLUMN observacoes VARCHAR(255) NOT NULL;
Query OK, 45 rows affected (0,04 sec)
Records: 45  Duplicates: 0  Warnings: 0
```

Observe que foram alteradas todas as 45 linhas existentes no nosso banco de dados. Se verificarmos a estrutura da nossa tabela novamente:

```
mysql> DESC compras;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| id          | int(11)       | NO   | PRI | NULL    | auto_increment |
| valor       | decimal(18,2) | YES  |     | NULL    |                |
| data        | date          | YES  |     | NULL    |                |
| observacoes | varchar(255)  | NO   |     | NULL    |                |
| recebida    | tinyint(4)    | YES  |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
5 rows in set (0,01 sec)
```

Na coluna *Null* e na linha das `observacoes` está informando que não é mais permitido valores nulos. Vamos testar:

```
mysql> INSERT INTO compras (valor, data, recebida, observacoes)
VALUES (150, '2016-01-04', 1, NULL);
ERROR 1048 (23000): Column 'observacoes' cannot be null
```

4.3 VALORES DEFAULT

Vamos supor que a maioria das compras que registramos não são entregues e que queremos que o próprio MySQL entenda que, quando eu não informar a coluna recebida, ela seja populada com o valor 0. No MySQL, além de *Constraints*, podemos adicionar valores padrões, no inglês *Default*, em uma coluna utilizando a instrução `DEFAULT`:

```
mysql> ALTER TABLE compras MODIFY COLUMN recebida tinyint(1) DEFAULT 0;
Query OK, 0 rows affected (0,00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Inserindo uma compra nova sem informa a coluna `recebida`:

```
INSERT INTO compras (valor, data, observacoes)
VALUES (150, '2016-01-05', 'Compra de teste');
Query OK, 1 row affected (0,01 sec)
```

Capítulo 5

AGRUPANDO DADOS E FAZENDO CONSULTAS MAIS INTELIGENTES

Já adicionamos muitos dados em nosso banco de dados e seria mais interessante fazermos *queries* mais robustas, como por exemplo, saber o total que já gastei. O MySQL fornece a função `SUM()` que soma todos dos valores de uma coluna:

```
mysql> SELECT SUM(valor) FROM compras;
+------------+
| SUM(valor) |
+------------+
|   43967.91 |
+------------+
1 row in set (0,00 sec)
```

Vamos verificar o total de todas as compras recebidas:

```
mysql> SELECT SUM(valor) FROM compras WHERE recebida = 1;
+------------+
| SUM(valor) |
+------------+
|   31686.75 |
+------------+
1 row in set (0,00 sec)
```

Agora todas as compras que não foram recebidas:

```
mysql> SELECT SUM(valor) FROM compras WHERE recebida = 0;
+------------+
| SUM(valor) |
+------------+
|   12281.16 |
+------------+
1 row in set (0,00 sec)
```

Podemos também, contar quantas compras foram recebidas por meio da função `COUNT()`:

```
SELECT COUNT(*) FROM compras WHERE recebida = 1;
+----------+
| COUNT(*) |
+----------+
|       26 |
+----------+
```

Agora vamos fazer com que seja retornado a soma de todas as compras recebidas e não recebidas, porém retornaremos a coluna recebida, ou seja, em uma linha estará as compras recebidas e a sua soma e

```
mysql> SELECT MONTH(data) as mes, YEAR(data) as ano, recebida,
SUM(valor) AS soma FROM compras
GROUP BY recebida;
+------+------+----------+----------+
| mes  | ano  | recebida | soma     |
+------+------+----------+----------+
|    1 | 2016 |        0 | 12281.16 |
|    1 | 2016 |        1 | 31686.75 |
+------+------+----------+----------+
2 rows in set (0,00 sec)
```

Lembre-se que estamos lidando com uma função de agregação! Por isso precisamos informar todas as colunas que queremos **agrupar**, ou seja, a coluna de mês e de ano:

```
mysql> SELECT MONTH(data) as mes, YEAR(data) as ano, recebida,
SUM(valor) AS soma FROM compras
GROUP BY recebida, mes, ano;
+------+------+----------+----------+
| mes  | ano  | recebida | soma     |
+------+------+----------+----------+
|    1 | 2013 |        0 |    12.39 |
|    1 | 2016 |        0 |  2015.49 |
|    4 | 2013 |        0 |    54.98 |
|    4 | 2014 |        0 |   434.00 |
|    5 | 2012 |        0 |  3500.00 |
|    5 | 2013 |        0 |    12.34 |
|    5 | 2015 |        0 |    87.43 |
|    6 | 2014 |        0 |  1616.90 |
|    7 | 2015 |        0 |    12.34 |
|    8 | 2014 |        0 |  1709.00 |
|    9 | 2014 |        0 |   567.09 |
|    9 | 2015 |        0 |   213.50 |
|   10 | 2014 |        0 |    98.00 |
|   10 | 2015 |        0 |  1245.20 |
|   12 | 2013 |        0 |    78.65 |
|   12 | 2014 |        0 |   623.85 |
|    1 | 2013 |        1 |  8046.90 |
|    1 | 2014 |        1 |   827.50 |
|    1 | 2016 |        1 |    35.00 |
|    2 | 2012 |        1 |   200.00 |
|    2 | 2014 |        1 |   921.11 |
|    2 | 2015 |        1 |   986.36 |
|    3 | 2013 |        1 |   795.05 |
|    4 | 2012 |        1 |  1576.40 |
|    4 | 2014 |        1 | 11705.30 |
|    5 | 2014 |        1 |   678.43 |
|    7 | 2013 |        1 |    98.12 |
|    7 | 2015 |        1 |    32.09 |
|    9 | 2015 |        1 |   576.12 |
|   10 | 2014 |        1 |   631.53 |
|   11 | 2013 |        1 |  3212.40 |
|   11 | 2015 |        1 |   954.12 |
|   12 | 2012 |        1 |   163.45 |
|   12 | 2013 |        1 |   223.09 |
|   12 | 2015 |        1 |    23.78 |
+------+------+----------+----------+
35 rows in set (0,00 sec)
```

5.1 ORDENANDO OS RESULTADOS

```
|    4 | 2014 |        0 |  434.000000 |
|    4 | 2014 |        1 | 5852.650000 |
|    5 | 2014 |        1 |  678.430000 |
|    6 | 2014 |        0 |  808.450000 |
|    8 | 2014 |        0 | 1709.000000 |
|    9 | 2014 |        0 |  567.090000 |
|   10 | 2014 |        1 |  631.530000 |
|   10 | 2014 |        0 |   98.000000 |
|   12 | 2014 |        0 |  311.925000 |
|    2 | 2015 |        1 |  493.180000 |
|    5 | 2015 |        0 |   87.430000 |
|    7 | 2015 |        1 |   32.090000 |
|    7 | 2015 |        0 |   12.340000 |
|    9 | 2015 |        1 |  576.120000 |
|    9 | 2015 |        0 |  213.500000 |
|   10 | 2015 |        0 | 1245.200000 |
|   11 | 2015 |        1 |  954.120000 |
|   12 | 2015 |        1 |   23.780000 |
|    1 | 2016 |        1 |   17.500000 |
|    1 | 2016 |        0 |  671.830000 |
+------+------+----------+-------------+
35 rows in set (0,00 sec)
```


5.2 RESUMINDO

Vimos como podemos fazer *queries* mais robustas e inteligentes utilizando funções de agregação, como por exemplo, a `SUM()` para somar e a `AVG()` para tirar a média. Vimos também que quando queremos retornar outras colunas ao utilizar funções de agregação, precisamos utilizar a instrução `GROUP BY` para determinar quais serão as colunas que queremos que seja feita o agrupamento e que nem sempre o resultado vem organizado e por isso, em determinados casos, precisamos utilizar a instrução `ORDER BY` para ordenar a nossa *query* por meio de uma coluna.

EXERCÍCIOS

1. Calcule a média de todas as compras com datas inferiores a 12/05/2013.

CAPÍTULO 6

JUNTANDO DADOS DE VÁRIAS TABELAS

A nossa tabela de compras está bem populada, com muitas informações das compras realizadas, porém está faltando uma informação muito importante, que são os compradores. Por vezes foi eu quem comprou algo, mas também o meu irmão pode ter comprado ou até mesmo o meu primo... Como podemos fazer para identificar o comprador de uma compra? De acordo com o que vimos até agora podemos adicionar uma nova coluna chamada `comprador` com o tipo `varchar(200)` :

```
mysql> ALTER TABLE compras ADD COLUMN comprador VARCHAR(200);
Query OK, 0 rows affected (0,04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Vamos verificar como ficou a estrutura da nossa tabela:

```
mysql> DESC compras;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| id          | int(11)       | NO   | PRI | NULL    | auto_increment |
| valor       | decimal(18,2) | YES  |     | NULL    |                |
| data        | date          | YES  |     | NULL    |                |
| observacoes | varchar(255)  | NO   |     | NULL    |                |
| recebida    | tinyint(1)    | YES  |     | 0       |                |
| comprador   | varchar(200)  | YES  |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
6 rows in set (0,00 sec)
```

Ótimo! Agora podemos adicionar os compradores de cada compra! Mas, antes de adicionarmos precisamos verificar novamente verificar as informações das compras e identificar quem foi que comprou:

```
mysql> SELECT id, valor, observacoes, data FROM compras;
+----+----------+------------------------------+------------+
| id | valor    | observacoes                  | data       |
+----+----------+------------------------------+------------+
|  1 |    20.00 | Lanchonete                   | 2016-01-05 |
|  2 |    15.00 | Lanchonete                   | 2016-01-06 |
|  3 |   915.50 | Guarda-roupa                 | 2016-01-06 |
|  4 |   949.99 | Smartphone                   | 2016-01-10 |
|  5 |   200.00 | Material escolar             | 2012-02-19 |
|  6 |  3500.00 | Televisao                    | 2012-05-21 |
|  7 |  1576.40 | Material de construcao       | 2012-04-30 |
|  8 |   163.45 | Pizza pra familia            | 2012-12-15 |
|  9 |  4780.00 | Sala de estar                | 2013-01-23 |
| 10 |   392.15 | Quartos                      | 2013-03-03 |
| 12 |   402.90 | Copa                         | 2013-03-21 |
| 13 |    54.98 | Lanchonete                   | 2013-04-12 |
| 14 |    12.34 | Lanchonete                   | 2013-05-23 |
```

comprador? Eu teria que **lembrar todas** essas informações **cada vez** que eu **inserir apenas uma compra**! Que horror!

Veja o quão problemático está sendo manter as informações de um comprador em uma tabela de compras. Além de trabalhosa, a inserção é bem problemática, pois há um grande risco de falhas no meu banco de dados como por exemplo: informações redundantes (que se repetem) ou então informações incoerentes (o mesmo comprador com alguma informação diferente). Com toda certeza não é uma boa solução para a nossa necessidade! Será que não existe uma forma diferente de resolver isso?

6.1 NORMALIZANDO NOSSO MODELO

Repara que temos dois elementos, duas entidades, em uma unica tabela: a compra e o comprador. Quando nos deparamos com esses tipos de problemas **criamos novas tabelas**. Então vamos criar uma tabela chamada compradores.

No nosso caso queremos que cada comprador seja representado pelo seu nome, endereço e telefone. Como já pensamos em modelagem antes, aqui fica o nosso modelo de tabela para representar os compradores:

```
mysql> CREATE TABLE compradores (
id INT PRIMARY KEY AUTO_INCREMENT,
nome VARCHAR(200),
endereco VARCHAR(200),
telefone VARCHAR(30)
);
Query OK, 0 rows affected (0,01 sec)
```

Vamos verificar a nossa tabela por meio do `DESC` :

```
mysql> DESC compradores;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| nome     | varchar(200) | YES  |     | NULL    |                |
| endereco | varchar(200) | YES  |     | NULL    |                |
| telefone | varchar(30)  | YES  |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
4 rows in set (0,00 sec)
```

Agora que criamos a nossa tabela, vamos inserir alguns compradores, no meu caso, as compras foram feitas apenas por mim e pelo meu tio João da Silva, então adicionaremos apenas 2 compradores:

```
mysql> INSERT INTO compradores (nome, endereco, telefone) VALUES
('Alex Felipe', 'Rua Vergueiro, 3185', '5571-2751');
Query OK, 1 row affected (0,01 sec)

mysql> INSERT INTO compradores (nome, endereco, telefone) VALUES
('João da Silva', 'Av. Paulista, 6544', '2220-4156');
Query OK, 1 row affected (0,01 sec)
```

Vamos verificar os nossos compradores:

```
| 21 |   768.90 | 2013-01-16 | Festa                          |        1 |              1 |
| 22 |   827.50 | 2014-01-09 | Festa                          |        1 |              2 |
| 23 |    12.00 | 2014-02-19 | Salgado no aeroporto           |        1 |              2 |
| 24 |   678.43 | 2014-05-21 | Passagem pra Bahia             |        1 |              2 |
| 25 | 10937.12 | 2014-04-30 | Carnaval em Cancun             |        1 |              2 |
| 26 |  1501.00 | 2014-06-22 | Presente da sogra              |        0 |              2 |
| 27 |  1709.00 | 2014-08-25 | Parcela da casa                |        0 |              2 |
| 28 |   567.09 | 2014-09-25 | Parcela do carro               |        0 |              2 |
| 29 |   631.53 | 2014-10-12 | IPTU                           |        1 |              2 |
| 30 |   909.11 | 2014-02-11 | IPVA                           |        1 |              2 |
| 31 |   768.18 | 2014-04-10 | Gasolina viagem Porto Alegre   |        1 |              2 |
| 32 |   434.00 | 2014-04-01 | Rodeio interior de Sao Paulo   |        0 |              2 |
| 33 |   115.90 | 2014-06-12 | Dia dos namorados              |        0 |              2 |
| 34 |    98.00 | 2014-10-12 | Dia das crianças               |        0 |              2 |
| 35 |   253.70 | 2014-12-20 | Natal - presentes              |        0 |              2 |
| 36 |   370.15 | 2014-12-25 | Compras de natal               |        0 |              2 |
| 37 |    32.09 | 2015-07-02 | Lanchonete                     |        1 |              2 |
| 38 |   954.12 | 2015-11-03 | Show da Ivete Sangalo          |        1 |              2 |
| 39 |    98.70 | 2015-02-07 | Lanchonete                     |        1 |              2 |
| 40 |   213.50 | 2015-09-25 | Roupas                         |        0 |              2 |
| 41 |  1245.20 | 2015-10-17 | Roupas                         |        0 |              2 |
| 42 |    23.78 | 2015-12-18 | Lanchonete do Zé               |        1 |              2 |
| 43 |   576.12 | 2015-09-13 | Sapatos                        |        1 |              2 |
| 44 |    12.34 | 2015-07-19 | Canetas                        |        0 |              2 |
| 45 |    87.43 | 2015-05-10 | Gravata                        |        0 |              2 |
| 46 |   887.66 | 2015-02-02 | Presente para o filhao         |        1 |              2 |
| 48 |   150.00 | 2016-01-05 | Compra de teste                |        0 |              2 |
+----+----------+------------+--------------------------------+----------+----------------+
46 rows in set (0,00 sec)
```


6.2 ONE TO MANY/MANY TO ONE

Repare que o que fizemos foi deixar claro que um comprador pode ter muitas compras (um para muitos), ou ainda que muitas compras podem vir do mesmo comprador (many to one), só depende do ponto de vista. Chamamos então de uma relação One to Many (ou Many to One).

6.3 FOREIGN KEY

CAPÍTULO 8

AGRUPANDO DADOS COM GROUP BY

A instuição solicitou a média de todos os cursos para fazer uma comparação de notas para verificar se todos os cursos possuem a mesma média, quais cursos tem menores notas e quais possuem as maiores notas. Vamos verificar a estrutura de algumas tabelas da nossa base de dados, começaremos pela tabela `curso`:

```
DESC curso;
```

```
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| nome  | varchar(255) | NO   |     |         |                |
+-------+--------------+------+-----+---------+----------------+
```

Agora vamos verificar a tabela `secao`:

```
DESC secao;
```

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| curso_id   | int(11)      | NO   |     | NULL    |                |
| titulo     | varchar(255) | NO   |     |         |                |
| explicacao | varchar(255) | NO   |     | NULL    |                |
| numero     | int(11)      | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

Já podemos perceber que existe uma relação entre `curso` de `secao`. Vamos também dar uma olhada na tabela `exercicio`:

```
DESC exercicio;
```

```
+------------------+--------------+------+-----+---------+----------------+
| Field            | Type         | Null | Key | Default | Extra          |
+------------------+--------------+------+-----+---------+----------------+
| id               | int(11)      | NO   | PRI | NULL    | auto_increment |
| secao_id         | int(11)      | NO   |     | NULL    |                |
| pergunta         | varchar(255) | NO   |     | NULL    |                |
| resposta_oficial | varchar(255) | NO   |     | NULL    |                |
+------------------+--------------+------+-----+---------+----------------+
```

Observe que na tabela `exercicio` temos uma associação com a tabela `secao`. Agora vamos verificar a tabela `resposta`:

```
DESC resposta
```

```
JOIN matricula m ON m.curso_id = c.id
JOIN aluno a ON m.aluno_id = a.id;
```

Precisamos agora contar a quantidade de alunos:

```
SELECT c.nome, COUNT(a.id) AS quantidade FROM curso c
JOIN matricula m ON m.curso_id = c.id
JOIN aluno a ON m.aluno_id = a.id;

+----------------------+------------+
| nome                 | quantidade |
+----------------------+------------+
| SQL e banco de dados |         14 |
+----------------------+------------+
```

Lembre-se que precisamos agrupar a contagem pelo nome do curso:

```
SELECT c.nome, COUNT(a.id) AS quantidade FROM curso c
JOIN matricula m ON m.curso_id = c.id
JOIN aluno a ON m.aluno_id = a.id
GROUP BY c.nome;

+-----------------------------------+------------+
| nome                              | quantidade |
+-----------------------------------+------------+
| C# e orientação a objetos         |          4 |
| Desenvolvimento mobile com Android |          2 |
| Desenvolvimento web com VRaptor   |          2 |
| Scrum e métodos ágeis             |          2 |
| SQL e banco de dados              |          4 |
+-----------------------------------+------------+
```

Agora conseguimos realizar o nosso relatório conforme o esperado.

8.1 RESUMINDO

Vimos nesse capítulo como podemos gerar relatórios utilizando funções como `AVG()` e `COUNT()`. Vimos também que, se precisamos retornar as colunas para verificar qual é o valor de cada linha, como por exemplo, a média de cada curso, precisamos agrupar essas colunas por meio do `GROUP BY`. Vamos para os exercícios?

EXERCÍCIOS

1. Exiba a média das notas por curso.
2. Devolva o curso e as médias de notas, levando em conta somente alunos que tenham "Silva" ou "Santos" no sobrenome.
3. Conte a quantidade de respostas por exercício. Exiba a pergunta e o número de respostas.
4. Você pode ordenar pelo COUNT também. Basta colocar ORDER BY COUNT(coluna).

Capítulo 9

FILTRANDO AGREGAÇÕES E O HAVING

Todo o fim de semestre, a instituição de ensino precisa montar os boletins dos alunos. Então vamos montar a *query* que retornará todas as informações para montar o boletim. Começaremos retornando todas as notas dos alunos:

```
SELECT n.nota FROM nota n
```

Agora vamos associar a com as respostas com as notas:

```
SELECT n.nota FROM nota n
JOIN resposta r ON r.id = n.resposta_id
```

Associaremos agora com os exercícios com as respostas:

```
SELECT n.nota FROM nota n
JOIN resposta r ON r.id = n.resposta_id
JOIN exercicio e ON e.id = r.exercicio_id
```

Agora associaremos a seção com os exercícios:

```
SELECT n.nota FROM nota n
JOIN resposta r ON r.id = n.resposta_id
JOIN exercicio e ON e.id = r.exercicio_id
JOIN secao s ON s.id = e.secao_id
```

Agora o curso com a seção:

```
SELECT n.nota FROM nota n
JOIN resposta r ON r.id = n.resposta_id
JOIN exercicio e ON e.id = r.exercicio_id
JOIN secao s ON s.id = e.secao_id
JOIN curso c ON c.id = s.curso_id
```

Por fim, a resposta com o aluno:

```
SELECT n.nota FROM nota n
JOIN resposta r ON r.id = n.resposta_id
JOIN exercicio e ON e.id = r.exercicio_id
JOIN secao s ON s.id = e.secao_id
JOIN curso c ON c.id = s.curso_id
JOIN aluno a ON a.id = r.aluno_id;
```

Verificando o resultado:

```
+-------+
| nota  |
+-------+
|  8.00 |
```

```
| nome           | nome                 | AVG(n.nota) |
+----------------+----------------------+-------------+
| João da Silva  | SQL e banco de dados |    5.740741 |
+----------------+----------------------+-------------+
```

Lembre-se que estamos lidando com uma função de agregação, ou seja, se não informarmos a forma que ela precisa **agrupar** as colunas, ela retornará apenas uma linha! Porém, precisamos sempre pensar em qual tipo de agrupamento é necessário, nesse caso queremos que mostre a média de acada aluno, então agruparemos pelos alunos, porém também que queremos que a cada curso que o alunos fez mostre a sua :

```
SELECT a.nome, c.nome, AVG(n.nota) FROM nota n
JOIN resposta r ON r.id = n.resposta_id
JOIN exercicio e ON e.id = r.exercicio_id
JOIN secao s ON s.id = e.secao_id
JOIN curso c ON c.id = s.curso_id
JOIN aluno a ON a.id = r.aluno_id
GROUP BY a.nome, c.nome;
```

```
+-----------------+---------------------------------+-------------+
| nome            | nome                            | AVG(n.nota) |
+-----------------+---------------------------------+-------------+
| Alberto Santos  | Scrum e métodos ágeis           |    5.777778 |
| Frederico José  | Desenvolvimento web com VRaptor |    8.000000 |
| Frederico José  | SQL e banco de dados            |    5.666667 |
| João da Silva   | SQL e banco de dados            |    6.285714 |
| Renata Alonso   | C# e orientação a objetos       |    4.857143 |
+-----------------+---------------------------------+-------------+
```

9.1 CONDIÇÕES COM HAVING

Retornamos todas as médias dos alunos, porém a instituição precisa de um relatório separado para todos os alunos que reprovaram, ou seja, que tiraram nota baixa, nesse caso médias menores que 5. De acordo com o que vimos até agora bastaria adicionarmos um `WHERE` :

```
SELECT a.nome, c.nome, AVG(n.nota) FROM nota n
JOIN resposta r ON r.id = n.resposta_id
JOIN exercicio e ON e.id = r.exercicio_id
JOIN secao s ON s.id = e.secao_id
JOIN curso c ON c.id = s.curso_id
JOIN aluno a ON a.id = r.aluno_id
WHERE AVG(n.nota) < 5
GROUP BY a.nome, c.nome;

ERROR 1111 (HY000): Invalid use of group function
```

Nesse caso, estamos tentando adicionar condições para uma função de agregação, porém, quando queremos **adicionar condições para funções de agregação** precisamos utilizar o `HAVING` ao invés de `WHERE` :

```
SELECT a.nome, c.nome, AVG(n.nota) FROM nota n
JOIN resposta r ON r.id = n.resposta_id
JOIN exercicio e ON e.id = r.exercicio_id
JOIN secao s ON s.id = e.secao_id
JOIN curso c ON c.id = s.curso_id
```

Agora podemos enviar o relatório para a instituição.

Você pode também fazer o curso data dessa apostila na Caelum

Querendo aprender ainda mais sobre? Esclarecer dúvidas dos exercícios? Ouvir explicações detalhadas com um instrutor?
A Caelum oferece o **curso data** presencial nas cidades de São Paulo, Rio de Janeiro e Brasília, além de turmas incompany.

Consulte as vantagens do curso *SQL e modelagem com banco de dados*

9.2 RESUMINDO

Sabemos que para adicionarmos filtros apenas para colunas utilizamos a instrução `WHERE` e indicamos todas as peculiaridades necessárias, porém quando precisamos adicionar filtros para funções de agregação, como por exemplo o `AVG()` , precisamos utilizar a instrução `HAVING` . Além disso, é sempre bom lembrar que, quando estamos desenvolvendo *queries* grandes, é recomendado que faça passa-a-passo *queries* menores, ou seja, resolva os menores problemas juntando cada tabela por vez e teste para verificar se está funcionando, pois isso ajuda a verificar aonde está o problema da *query*. Vamos para os exercícios?

EXERCÍCIOS

1. Qual é a principal diferença entre as instruções having e where do sql?
2. Devolva todos os alunos, cursos e a média de suas notas. Lembre-se de agrupar por aluno e por curso. Filtre também pela nota: só mostre alunos com nota média menor do que 5.
3. Exiba todos os cursos e a sua quantidade de matrículas. Mas, exiba somente cursos que tenham mais de 1 matrícula.
4. Exiba o nome do curso e a quantidade de seções que existe nele. Mostre só cursos com mais de 3 seções.

Capítulo 10

MÚLTIPLOS VALORES NA CONDIÇÃO E O IN

O setor de financeiro dessa instituição, solicitou um relatório informando todas as formas de pagamento cadastradas no banco de dados para verificar se está de acordo com o que eles trabalham. Na base de dados, se verificarmos a tabela `matricula` :

```
DESC matricula;
```

```
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| id       | int(11)     | NO   | PRI | NULL    | auto_increment |
| aluno_id | int(11)     | NO   |     | NULL    |                |
| curso_id | int(11)     | NO   |     | NULL    |                |
| data     | datetime    | NO   |     | NULL    |                |
| tipo     | varchar(20) | NO   |     |         |                |
+----------+-------------+------+-----+---------+----------------+
```

Observe que existe a coluna `tipo` que representa qual é a forma de pagamento. E precisamos pegar um relatório da seguinte maneira:

Forma de pagamento
Forma 1
Forma 2
Forma 3

Figura 10.1: Planilha exemplo

Vamos selecionar apenas a coluna `tipo` da tabela `matricula` :

```
SELECT m.tipo FROM matricula m;
```

```
+-------------+
| tipo        |
+-------------+
| PAGA_PF     |
| PAGA_PJ     |
| PAGA_PF     |
| PAGA_CHEQUE |
| PAGA_BOLETO |
| PAGA_PJ     |
| PAGA_PF     |
| PAGA_PJ     |
| PAGA_PJ     |
| PAGA_CHEQUE |
| PAGA_BOLETO |
```

```
| João da Silva   | C# e orientação a objetos   |
| Frederico José  | C# e orientação a objetos   |
| Alberto Santos  | C# e orientação a objetos   |
+-----------------+-----------------------------+
```

Novamente o resultado está desordenado, vamos ordenar pelo nome do aluno:

```
SELECT a.nome, c.nome FROM aluno a
JOIN matricula m ON m.aluno_id = a.id
JOIN curso c ON m.curso_id = c.id
WHERE c.id IN (1, 4)
ORDER BY a.nome;

+-----------------+-----------------------------+
| nome            | nome                        |
+-----------------+-----------------------------+
| Alberto Santos  | C# e orientação a objetos   |
| Frederico José  | SQL e banco de dados        |
| Frederico José  | C# e orientação a objetos   |
| João da Silva   | SQL e banco de dados        |
| João da Silva   | C# e orientação a objetos   |
| Manoel Santos   | SQL e banco de dados        |
| Paulo José      | SQL e banco de dados        |
| Renata Alonso   | C# e orientação a objetos   |
+-----------------+-----------------------------+
```

Agora sabemos que apenas os alunos Frederico José e João da Silva, são os ex-alunos aptos para realizar os novos cursos de *.NET*.

10.1 RESUMINDO

Nesse capítulo vimos que quando precisamos saber todos os valores de uma determinada coluna podemos utilizar a instrução `DISTINCT` para retornar todos os valores **distintos**, ou seja, sem nenhuma repetição. Vimos também que quando precisamos realizar vários filtros para uma mesma coluna, podemos utilizar a instrução `IN` passando por parâmetro todos os valores que esperamos que seja retornado, ao invés de ficar preenchendo a nossa *query* com vários `OR` s. Vamos para os exercícios?

EXERCÍCIOS

1. Exiba todos os tipos de matrícula que existem na tabela. Use DISTINCT para que não haja repetição.
2. Exiba todos os cursos e a sua quantidade de matrículas. Mas filtre por matrículas dos tipos PF ou PJ.
3. Traga todas as perguntas e a quantidade de respostas de cada uma. Mas dessa vez, somente dos cursos com ID 1 e 3.

Capítulo 12

ENTENDENDO O LEFT JOIN

Os instrutores da instituição pediram um relatório com os alunos que são mais participativos na sala de aula, ou seja, queremos retornar os alunos que responderam mais exercícios. Consequentemente encontraremos também os alunos que não estão participando muito, então já aproveitamos e conversamos com eles para entender o que está acontecendo. Então começaremos retornando o aluno:

```
SELECT a.nome FROM aluno n;
```

Agora vamos contar a quantidade de respostas por meio da função `COUNT()` e agrupando pelo nome do aluno:

```
SELECT a.nome, COUNT(r.id) AS respostas
FROM aluno a
JOIN resposta r ON r.aluno_id = a.id
GROUP BY a.nome;

+-----------------+-----------+
| nome            | respostas |
+-----------------+-----------+
| Alberto Santos  |         9 |
| Frederico José  |         4 |
| João da Silva   |         7 |
| Renata Alonso   |         7 |
+-----------------+-----------+
```

Mas onde estão todos os meus alunos? Fugiram? Aparentemente essa *query* não está trazendo exatamente o que a gente esperava... Vamos contar a quantidade de alunos existentes:

```
SELECT COUNT(a.id) FROM aluno a;

+-------------+
| COUNT(a.id) |
+-------------+
|          16 |
+-------------+
```

Observe que existe 16 alunos no banco de dados, porém só foram retornados 4 alunos e suas respostas. Provavelmente não está sendo retornando os alunos que não possuem respostas! Vamos verificar o que está acontecendo exatamente. Vamos pegar um aluno que não foi retornado, como o de id 5. Quantas respostas ele tem?

```
SELECT r.id FROM resposta r WHERE r.aluno_id = 5;

+------------------+
| id               |
```

```
| Carlos Cunha     |             0 |              0 |
| Claudio Soares   |             0 |              0 |
| Cristaldo Santos |             0 |              0 |
| Danilo Cunha     |             0 |              0 |
| Frederico José   |            12 |             12 |
| João da Silva    |            14 |             14 |
| Jose da Silva    |             0 |              0 |
| Manoel Santos    |             0 |              2 |
| Osmir Ferreira   |             0 |              0 |
| Paula Soares     |             0 |              1 |
| Paulo da Silva   |             0 |              0 |
| Paulo José       |             0 |              1 |
| Renata Alonso    |            14 |             14 |
| Renata Ferreira  |             0 |              1 |
| Zilmira José     |             0 |              0 |
+------------------+---------------+----------------+
```

O resultado além de ser bem maior que o esperado, repete nas duas colunas , pois está acontecendo aquele problema da multiplicação das linhas! Perceba que só conseguimos verificar de uma forma rápida o problema que aconteceu, pois fizemos a *query* **passo-a-passo**, verificando cada resultado e, ao mesmo tempo, corringo os problemas que surgiam.

12.3 RESUMINDO

Neste capítulo aprendemos como utilizar os diferentes tipos de `JOIN` s, como por exemplo o `LEFT JOIN` que retorna os registros da tabela a esquerda e o `RIGHT JOIN` que retorna os da direita mesmo que não tenham associações. Vimos também que o `JOIN` também é conhecido como `INNER JOIN` que retorna apenas os registros que estão associados. Além disso, vimos que algumas *queries* podem ser resolvidas utilizando *subqueries* ou `LEFT/RIGHT JOIN` , porém é importante lembrar que os SGBDs sempre terão melhor desempenho com o os `JOIN` s, por isso é recomendado que utilize os `JOIN` s. Vamos para os exercícios?

EXERCÍCIOS

1. Exiba todos os alunos e suas possíveis respostas. Exiba todos os alunos, mesmo que eles não tenham respondido nenhuma pergunta.

2. Exiba agora todos os alunos e suas possíveis respostas para o exercício com ID = 1. Exiba todos os alunos mesmo que ele não tenha respondido o exercício.

   Lembre-se de usar a condição no JOIN.

1. Qual a diferença entre o JOIN convencional (muitas vezes chamado também de INNER JOIN) para o LEFT JOIN?

CAPÍTULO 13

MUITOS ALUNOS E O LIMIT

Precisamos de um relatório que retorne todos os alunos, algo como selecionar o nome de todos eles, com um `SELECT` simples:

```
SELECT a.nome FROM aluno a;
```

```
+------------------+
| nome             |
+------------------+
| João da Silva    |
| Frederico José   |
| Alberto Santos   |
| Renata Alonso    |
| Paulo da Silva   |
| Carlos Cunha     |
| Paulo José       |
| Manoel Santos    |
| Renata Ferreira  |
| Paula Soares     |
| Jose da Silva    |
| Danilo Cunha     |
| Zilmira José     |
| Cristaldo Santos |
| Osmir Ferreira   |
| Claudio Soares   |
+------------------+
```

Para melhorar o resultado podemos ordernar a *query* por ordem alfabética do nome:

```
SELECT a.nome FROM aluno a ORDER BY a.nome;
```

```
+------------------+
| nome             |
+------------------+
| Alberto Santos   |
| Carlos Cunha     |
| Claudio Soares   |
| Cristaldo Santos |
| Danilo Cunha     |
| Frederico José   |
| João da Silva    |
| Jose da Silva    |
| Manoel Santos    |
| Osmir Ferreira   |
| Paula Soares     |
| Paulo da Silva   |
| Paulo José       |
| Renata Alonso    |
| Renata Ferreira  |
| Zilmira José     |
```

```
+------------------+
```

Vamos verificar agora quantos alunos estão cadastrados:

```
SELECT count(*) FROM aluno;

+----------+
| count(*) |
+----------+
|       16 |
+----------+
```

Como podemos ver, é uma quantidade relativamente baixa, pois quando estamos trabalhando em uma aplicação real, geralmente o volume de informações é muito maior.

No facebook, por exemplo, quantos amigos você tem? Quando você entra no facebook, aparece todas as atualizações dos seus amigos de uma vez? Todas as milhares de atualizações de uma única vez? Imagine a loucura que é trazer milhares de dados de uma única vez para você ver apenas 5, 10 notificações. Provavelmente vai aparecendo aos poucos, certo? Então que tal mostrarmos os alunos aos poucos também? Ou seja, fazermos uma **paginação** no nosso relatório, algo como 5 alunos "por página". Mas como podemos fazer isso? No MySQL, podemos **limitar** em 5 a quantidade de registros que desejamos retornar:

```
SELECT a.nome FROM aluno a
ORDER BY a.nome
LIMIT 5;

+------------------+
| nome             |
+------------------+
| Alberto Santos   |
| Carlos Cunha     |
| Claudio Soares   |
| Cristaldo Santos |
| Danilo Cunha     |
+------------------+
```

Nesse caso retornamos os primeiros 5 alunos em ordem alfabética. O ato de limitar é extremamente importante a medida que os dados crescem. Se você tem mil mensagens antigas, não vai querer ver as mil de uma vez só, traga somente as 10 primeiras e, se tiver interesse, mais 10, mais 10 etc.

13.1 LIMITANDO E BUSCANDO A PARTIR DE UMA QUANTIDADE ESPECÍFICA

Agora vamos pegar os próximos 5 alunos, isto é, senhor MySQL, ignore os 5 primeiros, e depois pegue para mim os próximos 5:

```
SELECT a.nome FROM aluno a
ORDER BY a.nome
LIMIT 5,5;

+-----------------+
| nome            |
```

13.2 RESUMINDO

Nesse capítulo vimos como nem sempre retornar todos os registros das tabelas são necessários, algumas vezes, precisamos filtrar a quantidade de linhas, pois em uma aplicação real, podemos lidar com uma quantidade bem grande de dados. Justamente por esse caso, podemos limitar as nossas *queries* utilizando a instrução `LIMIT` . Vamos para os exercícios?

EXERCÍCIOS

1. Escreva uma query que traga apenas os dois primeiros alunos da tabela.
2. Escreva uma SQL que devolva os 3 primeiros alunos que o e-mail termine com o domínio ".com".
3. Devolva os 2 primeiros alunos que o e-mail termine com ".com", ordenando por nome.
4. Devolva todos os alunos que tenham Silva em algum lugar no seu nome.