SQL com mySQL

Elaborado por: Gustavo Mota

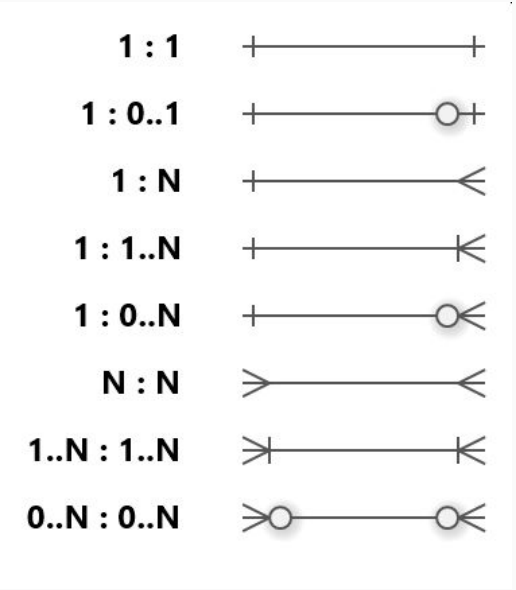
SQL e Bancos de Dados Relacionais

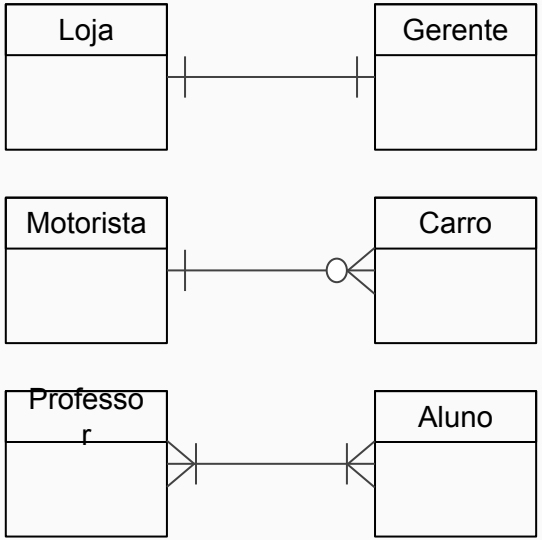
SQL (Structured Query Language), é uma linguagem de programação declarativa, de domínio específico, usada para armazenar, processar e consultar informações em um banco de dados relacional. Um banco de dados relacional guarda informações em forma de tabelas, com as colunas sendo atributos e as linhas guardando instâncias das entidades, sendo que essas últimas possuem relações entre si estabelecidas por colunas especiais chamadas chaves.

DBMS/SGBD, mySQL e mySQL Workbench

- O SGBD (Sistema de Gerenciamento de Banco de Dados), ou DBMS em inglês, é o software utilizado para criar, gerenciar e manipular os bancos de dados.
- mySQL é um SGBD
- mySQL Workbench é uma GUI (Graphical User Interface) para o mySQL, ou seja, uma ferramenta visual para usar as funções do mySQL

Modelagem de Dados: Entidades, Relações e Tabelas

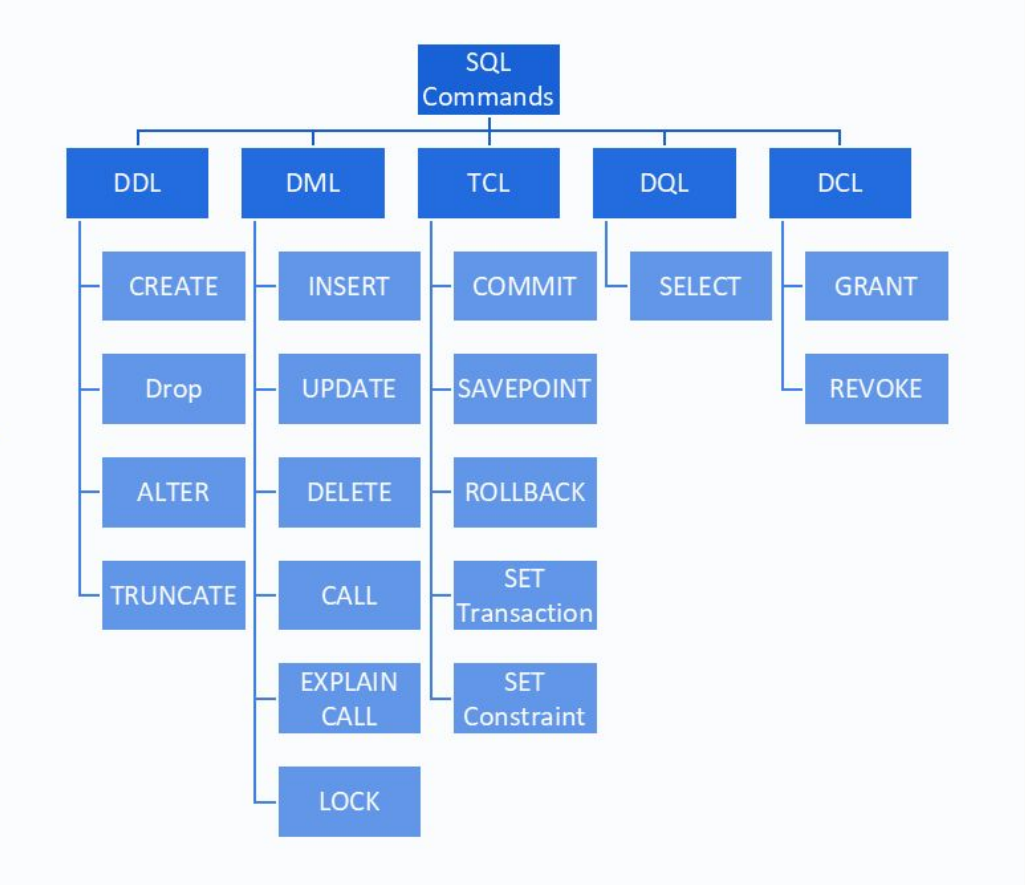




Placa	Modelo	Ano
PJX-2345	Pord Foco	2021
QRS-4567	Chev Prism	2022
ONQ-1234	Ren Quid	2020
NQW-0987	HP20	2019
OPS-2345	Fyat Mobo	2020

DDL, DML, TCL, DQL e DCL

- Data Definition Language
- Data Query Language
- Data Manipulation Language
- Data Control Language
- Transaction Control Language



Tipos de Dados

Para strings, texto:

- **CHAR(size)**: tamanho fixo, 0 a 255 caracteres
- **VARCHAR(size)**: tamanho variável, 64KB
- **TINYTEXT**: tamanho variável, 255 caracteres
- **TEXT(size)**: tamanho variável, 64KB
- **MEDIUMTEXT**: tamanho variável 16MB
- **LONGTEXT**: tamanho variável 4GB

Para valores binários:

- **BIT(size)**: armazena de 1 a 64 bits
- **BOOL** ou **BOOLEAN**: 0 é falso, resto verdadeiro, é equivalente a TINYINT(1)
- **BINARY(size)**: CHAR porém binário
- **VARBINARY(size)**: VARCHAR porém binário
- **TINYBLOB**: TINYTEXT porém binário
- **BLOB(size)**: TEXT porém binário
- **MEDIUMBLOB**: MEDIUMTEXT porém binário
- **LONGBLOB**: LONGTEXT porém binário

Tipos de Dados

Para números inteiros:

- **TINYINT**: inteiro de -128 a 127 ou de 0 a 255
- **SMALLINT**: inteiro de -32768 a 32767 ou de 0 a 65535
- **MEDIUMINT**: inteiro de -8M a 8M ou de 0 a 16M
- **INT** ou **INTEGER**: inteiro de -2B a 2B ou de 0 a 4B
- **BIGINT**: inteiro de -9Q a 9Q ou de 0 a 18Q

Para números quebrados:

- **FLOAT**: para números quebrados aproximados, ponto flutuante, 4 bytes
- **DOUBLE**: para números quebrados aproximados de maior precisão, ponto flutuante, 8 bytes
- **DEC(m, d)** ou **DECIMAL(m, d)**: para números quebrados exatos, ponto fixo, m é o número de dígitos significativos e d é o número de dígitos após a vírgula

Tipos de Dados

Para datas e tempo:

- **DATE**: data em formato AAAA-MM-DD, de 1000-01-01 a 9999-12-31
- **DATETIME**: data e hora em formato AAAA-MM-DD hh:mm:ss, de 1000-01-01 00:00:00 a 9999-12-31 23:59:59
- **TIMESTAMP**: data e hora armazenada no formato Unix Epoch, número de segundos que passaram desde 1970-01-01 00:00:00 UTC. De 1970-01-01 00:00:01 a 2038-01-09 03:14:07, suporta inicialização e atualização automáticas para data atual

Mais datas:

- **TIME**: hora ou intervalo de tempo em formato hh:mm:ss, pode ser negativo, de -838:59:59 a 838:59:59
- **YEAR**: ano em formato AAAA, de 1901 a 2155 ou 0000

Chaves

Chave Primária: valor não nulo, que não se repete, que identifica unicamente uma linha de uma tabela

Chave Estrangeira: valor que é “emprestado” de uma linha de outra tabela para estabelecer uma relação entre as duas entidades, a entidade dependente é a portadora da chave estrangeira e a entidade independente fornece o valor

Comentários no mySQL

este é um comentário de uma linha

-- este é um comentário de uma linha

/* este é

um comentário

de várias linhas */

CREATE, DROP & USE DATABASE

CREATE DATABASE [**IF NOT EXISTS**] *nome_db*;

USE *nome_db*;

...

DROP DATABASE [**IF EXISTS**] *nome_db*;

...

CREATE & DROP TABLE

CREATE TABLE *nome_tabela* (

nome_coluna1 tipo1 restrição,

nome_coluna2 tipo2 restrição,

nome_coluna3 tipo3 restrição,

...

);

DROP TABLE [IF EXISTS] *nome_tabela*;

...

Constraints ou Restrições

- **NOT NULL**: Proíbe a coluna de ter valor nulo
- **UNIQUE**: Faz com que a coluna aceite apenas valores únicos naquela tabela, proibindo valores duplicados
- **DEFAULT**: Especifica um valor padrão para a coluna, que será usado caso o usuário não especifique qual ele deseja
- **PRIMARY KEY**: identifica a coluna como chave primária da tabela
- **FOREIGN KEY**: identifica a coluna como chave estrangeira da tabela, usa a palavra **REFERENCES** para especificar a qual chave primária ela se refere

Constraints ou Restrições

- **ON UPDATE**: especifica o comportamento da entidade dependente em caso de modificação ou exclusão da entidade independente, opções: **RESTRICT**, **CASCADE**, **SET NULL**

- **RESTRICT**: Impede a operação
- **CASCADE**: Propaga a modificação para a entidade dependente, ou seja, modifica a chave estrangeira ou apaga a linha
- **SET NULL**: altera a chave estrangeira para NULL

Tabela para exemplos futuros

```
CREATE TABLE varejista (
    cnpj CHAR(14) NOT NULL UNIQUE,
    razao_social VARCHAR(100) NOT NULL UNIQUE,
    nome_fantasia VARCHAR(100) NOT NULL,
    segmento VARCHAR(50) NOT NULL DEFAULT 'Variedades',
    CONSTRAINT pk_cnpj
    PRIMARY KEY (cnpj)
);
```

```
CREATE TABLE loja (
    codigo_loja CHAR(36) PRIMARY KEY,
    cnpj_varejista CHAR(14) NOT NULL,
    horario_abertura TIME,
    horario_fechamento TIME,
    endereco VARCHAR(100) NOT NULL,
    FOREIGN KEY (cnpj_varejista) REFERENCES varejista(cnpj)
    ON UPDATE CASCADE ON DELETE RESTRICT
);
```

TRUNCATE TABLE

TRUNCATE [**TABLE**] *nome_tabela*;

- Em vez de excluir a tabela em si como o **DROP**, **TRUNCATE** exclui todas as linhas da tabela
- Funciona como se fosse um **DELETE** em todas as linhas ou um **DROP** seguido de um **CREATE**

ALTER TABLE

Adicionar colunas:

ALTER TABLE *nome_tabela*

ADD [**COLUMN**] *nome_coluna1 tipo1 restrição*

[**FIRST** | **AFTER** *nome_coluna2*]

…

;

Remover colunas:

ALTER TABLE *nome_tabela*

DROP [**COLUMN**] *nome_coluna1*

…

;

ALTER TABLE

Modificar colunas:

ALTER TABLE *nome_tabela*

MODIFY *nome_coluna1 tipo1 restrição*

[**FIRST** | **AFTER** *nome_coluna2*]

...

;

Renomear tabela colunas:

ALTER TABLE *nome_tabela*

RENAME [**TO** | **AS**] *nome_tabela_novo*

ALTER TABLE *nome_tabela*

RENAME COLUMN *nome_coluna_antigo*

TO *nome_coluna_novo*

ALTER TABLE

Renomear e modificar colunas juntos:

ALTER TABLE *nome_tabela*

CHANGE *nome_coluna_antigo*

nome_coluna_novo tipo1 restrição

[**FIRST** | **AFTER** *nome_coluna2*]

...

;

Adicionar restrições:

ALTER TABLE *nome_tabela* **ADD** [**CONSTRAINT** [*nome_restricao]]* **PRIMARY KEY** (*nome_coluna*);

ALTER TABLE *nome_tabela* **ADD** [**CONSTRAINT** [*nome_restricao]]* **FOREIGN KEY** (*nome_coluna*) **REFERENCES** *nome_tabela2*(*nome_coluna2*);

ALTER TABLE *nome_tabela* **ADD** [**CONSTRAINT** [*nome_restricao]]* **UNIQUE** (*nome_coluna*);

ALTER TABLE

Remover restrições:

ALTER TABLE *nome_tabela* **DROP CONSTRAINT** *nome_restricao*;

Remover restrições chaves:

ALTER TABLE *nome_tabela* **DROP PRIMARY KEY**

ALTER TABLE *nome_tabela* **DROP FOREIGN KEY** *nome_chave*

INSERT

INSERT [**INTO**] *nome_tabela*

[(*nome_coluna1*, *nome_coluna2*, *nome_coluna3* ...)]

VALUES (*valor1*, *valor2*, *valor3* ...)

AUTO_INCREMENT e DEFAULT

INSERT INTO varejista (cnpj, razao_social, nome_fantasia)

VALUES ('12345678901234', 'Empório Varejista SA', 'Empório Variedades');

INSERT INTO varejista

VALUES ('09876543210987', 'Lojão Varejista SA', 'Lojão Variedades', **DEFAULT**);

CREATE TABLE animal (

    id **MEDIUMINT NOT NULL AUTO_INCREMENT**,

    nome **CHAR(30) NOT NULL**,

    PRIMARY KEY (id)

);

INSERT INTO animal (nome)

VALUES ('cachorro'), ('gato'), ('coelho'), ('cobra');

UUID

INSERT INTO loja (codigo_loja, cnpj_varejista, horario_abertura, horario_fechamento, endereco)

VALUES (**UUID()**, '12345678901234', '09:00:00', '22:00:00', 'Rua da Praça, 223');

UUID: Universally Unique Identifier ou Identificador Único Universal. Às vezes chamado de **GUID**. Consiste em 128 bits (16 bytes) gerados aleatoriamente, pode ser representado como string de 36 caracteres, no formato hexadecimal com hífens.

Exemplo:
52c0c480-be8e-11ed-8a8c-2cf05d9ad862

SELECT

SELECT

[**DISTINCT**]

{***** | *selecionado1* [**AS** *apelido1*], *selecionado2* [**AS** *apelido2*] ...}

[**FROM** *nome_tabela1*, *nome_tabela2*, *nome_tabela3* ...]

[**WHERE** *condicao1*]

[**GROUP BY** *nome_coluna1*, *nome_coluna2*, *nome_coluna3* ...]

[**HAVING** *condicao2*]

[**ORDER BY** *nome_coluna1*, *nome_coluna2*, *nome_coluna3* ...]

[**LIMIT** *numero1* [**OFFSET** *numero2*]]

SELECT, FROM, CAST e AS

SELECT sem referência a nenhuma tabela:

- **SELECT** 1 + 7 * 5 - 1;
- **SELECT UUID()**;
- **SELECT CAST**('19.35' **AS UNSIGNED**);
- **SELECT CAST**('1999-12-31 23:59:59' **AS DATE**);
- **SELECT** 5 + 5 **AS** minha_soma;

SELECT, SELECT * e SELECT DISTINCT:

- **SELECT** nome **FROM** animal;
- **SELECT * FROM** loja;
- **SELECT DISTINCT** segmento **FROM** varejista;

Tabela para exemplos futuros

```
CREATE TABLE produto (
    codigo_produto CHAR(36) PRIMARY KEY,
    codigo_loja CHAR(36) NOT NULL,
    nome VARCHAR(100) NOT NULL UNIQUE,
    descricao VARCHAR(255) NOT NULL,
    preco DECIMAL(7,2) NOT NULL,
    FOREIGN KEY (codigo_loja) REFERENCES loja(codigo_loja)
    ON UPDATE CASCADE ON DELETE RESTRICT
);
```

ORDER BY, ASC, DESC, LIMIT e OFFSET

Obter animais em ordem alfabética:

- **SELECT * FROM** animal **ORDER BY** nome;

Obter lojas na ordem em que abrem mais cedo:

- **SELECT * FROM** loja **ORDER BY** horario_abertura **ASC**;

Obter o produto mais caro:

- **SELECT * FROM** produto **ORDER BY** preco **DESC LIMIT** 1;

Ignorar o produto mais barato e mostrar os dez produtos que vem logo em seguida:

- **SELECT * FROM** produto **ORDER BY** preco **LIMIT** 10 **OFFSET** 1;

Operadores Lógicos

Operadores lógicos:

- **AND, &&**: Conjunção, verdadeiro se todos os valores testados são verdadeiros
- **OR,** ||: Disjunção, verdadeiro se qualquer um dos valores testados é verdadeiro
- **NOT, !**: Negação, inverte o valor verdadeiro para falso e falso para verdadeiro

Como se escreve verdadeiro, falso e nulo:

- **TRUE**
- **FALSE**
- **NULL**

Operadores Relacionais

Operadores relacionais:

- **<**: menor
- **>**: maior
- **<=**: menor ou igual
- **>=**: maior ou igual
- **=**: igual
- **<>**, **!=**: diferente
- **<=>**: similar ao igual, porém retorna 1 para NULL <=> NULL e 0 para NULL <=> X

Funções de Comparação

Funções de comparação:

- [**NOT**] **BETWEEN** *a* **AND** *b*: verifica se valor está no intervalo (x >= a && x <= b)
- [**NOT**] **IN()**: verifica se valor está numa lista de outros valores
- **IS** [**NOT**] {**TRUE**, **FALSE**, **NULL**}: verifica se valor é verdadeiro, falso ou nulo
- [**NOT**] **LIKE**: verifica se o valor corresponde a um padrão simples
- ...

WHERE

Obter lojas que estão abertas durante o almoço:

- **SELECT * FROM** loja **WHERE** horario_abertura <= '11:00:00" **AND** horario_fechamento >= '13:00:00'**;**

Obter produtos com preço inferior a 2000:

- **SELECT * FROM** produto **WHERE** preco < 2000.00**;**

Obter animais exceto o gato:

- **SELECT * FROM** animal **WHERE** nome != 'gato'**;**

WHERE

Obter lojas que funcionam 24h:

- **SELECT * FROM** loja **WHERE** horario_abertura **IS NULL AND** horario_fechamento **IS NULL;**

Obter produtos com preço entre 500 e 1500:

- **SELECT * FROM** produto **WHERE** preco **BETWEEN** 500.00 **AND** 1500.00**;**

Obter varejistas de variedades ou eletrônicos:

- **SELECT * FROM** varejista **WHERE** segmento **IN** ('Variedades', 'Eletrônicos')**;**

Obter animais cuja segunda letra do nome é A:

- **SELECT * FROM** animal **WHERE** nome **LIKE** '_A%'**;**

Funções de Agregação

- **AVG()**: Médio
- **COUNT()**: Contagem
- **COUNT(DISTINCT)**: Contagem de valores distintos
- **MAX()**: Máximo
- **MIN()**: Mínimo
- **SUM()**: Soma
- ...

Obter preço médio dos produtos:

- **SELECT AVG**(preco) **FROM** produto;

Obter menor preço dos produtos:

- **SELECT MIN**(preco) **FROM** produto;

Obter maior preço dos produtos:

- **SELECT MAX**(preco) **FROM** produto;

Funções de Agregação

Obter soma do preço dos produtos:

- **SELECT SUM**(preco) **FROM** produto;

Obter número de produtos que custam acima de 1000:

- **SELECT COUNT**(*) **FROM** produto **WHERE** preco > 1000;

Obter quantos segmentos distintos estão cadastrados no banco de dados:

- **SELECT COUNT**(**DISTINCT** segmento) **FROM** varejista;

Tabela para exemplos futuros

```
CREATE TABLE faturamento (
    data DATE NOT NULL,
    codigo_loja CHAR(36) NOT NULL,
    valor DECIMAL(9,2) NOT NULL,
    FOREIGN KEY (codigo_loja) REFERENCES loja(codigo_loja)
    ON UPDATE CASCADE ON DELETE RESTRICT,
    PRIMARY KEY (data, codigo_loja)
);
```

GROUP BY

GROUP BY permite juntar várias linhas em uma de acordo com uma coluna. Como diversos valores de linhas diferentes precisam ser aglutinados, cada valor selecionado deve fazer parte da cláusula de agrupamento ou estar submetido a uma função de agregação.

GROUP BY

Obter o faturamento anual de uma loja:

- **SELECT** codigo_loja, **SUM**(valor) **AS** faturamento_anual **FROM** faturamento **WHERE YEAR**(dia) = 2022 **GROUP BY** codigo_loja;

A consulta acima juntará todos valores de faturamento do ano 2022 que têm o mesmo código de loja, calculará a soma do total e retornará uma tabela com duas colunas: código da loja e o faturamento anual.

HAVING

Suponha agora que o setor financeiro queira saber quais lojas tiveram faturamento anual de ao menos um milhão:

- **SELECT** codigo_loja, **SUM**(valor) **AS** faturamento_anual **FROM** faturamento **WHERE YEAR**(dia) = 2022 **GROUP BY** codigo_loja **HAVING** faturamento_anual >= 1000000.00;

HAVING faz a mesma função de **WHERE** porém nas funções agregadoras, não é possível usar o **WHERE** sobre resultados dessas funções

UPDATE

UPDATE *nome_tabela*

SET *atribuicao1*, *atribuicao2*, *atribuicao3* ...

[**WHERE** condicao]

[**ORDER BY** nome_coluna1, nome_coluna2, nome_coluna3 ...]

[**LIMIT** numero]

UPDATE

Dar um desconto de 20% em todas as TVs:

- **UPDATE** produto **SET** preco = preco * 0.8 **WHERE** nome **LIKE** 'TV%'

Fazer todas as lojas abrirem uma hora mais tarde:

- **UPDATE** loja **SET** horario_abertura = **ADDTIME**(horario_abertura, '01:00:00') **WHERE** horario_abertura **IS NOT NULL**

Cuidado ao fazer UPDATE sem WHERE pois ele alterará todas as linhas!

DELETE

DELETE

FROM *nome_tabela* [**AS** *apelido*]

[**WHERE** condicao]

[**ORDER BY** nome_coluna1, nome_coluna2, nome_coluna3 ...]

[**LIMIT** numero]

DELETE

Apagar todos os produtos que por algum motivo tem preço zero ou negativo:

- **DELETE FROM** produto **WHERE** preco <= 0.00;

Apagar todos os registros de faturamento anteriores a 2010:

- **DELETE FROM** faturamento **WHERE YEAR**(dia) < 2010;

Cuidado ao fazer DELETE sem WHERE pois ele apagará todas as linhas!

Subquery ou Subconsulta

É possível usar consultas dentro de outras consultas, por exemplo, uma consulta que obtém os produtos com preço acima da média:

- **SELECT * FROM** produto **WHERE** preco > (**SELECT AVG**(preco) **FROM** produto);

UNION

Union permite juntar linhas de duas tabelas diferentes, as partes precisam ter o mesmo número de colunas e ordem das colunas e tipos de dados compatíveis. Por exemplo, juntar duas consultas para fazer uma lista de todos os UUIDs do banco de dados:

- **SELECT** codigo_loja **FROM** produto **UNION SELECT** codigo_produto **FROM** produto;

Tabela para exemplos futuros

```
CREATE TABLE contato_celular (
    ddd_telefone CHAR(11) PRIMARY KEY,
    nome VARCHAR(100) NOT NULL
);
```

```
CREATE TABLE contato_lista_telefonica (
    ddd_telefone CHAR(11) PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    endereco VARCHAR(100) NOT NULL
);
```

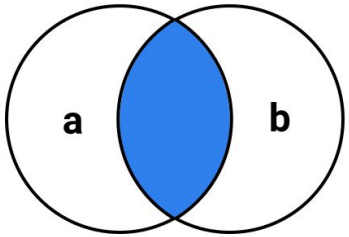
Situação problema

1) Como obter o nome do varejista que vende o produto ao consultá-lo?
2) Dada uma agenda de contatos em um celular, que possui apenas nome e número de telefone e uma lista telefônica que possui além dessas informações, o email e endereço dos listados, como descobrir os emails e endereços das pessoas na agenda do telefone?

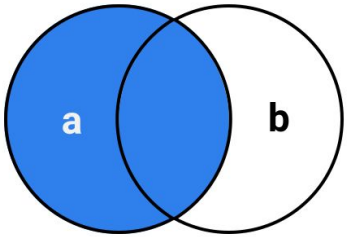
Solução: **JOIN** é capaz de juntar colunas de tabelas diferentes (ou até da mesma tabela) de acordo com condições fornecidas.

Tipos de JOIN

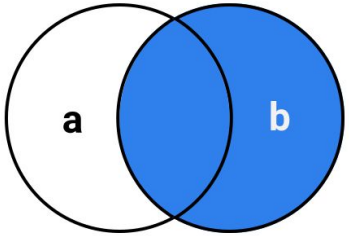
- INNER JOIN: traz apenas as linhas de a e b que satisfazem a condição
- LEFT JOIN: traz todas as linhas de a e apenas as linhas de b que satisfazem a condição
- RIGHT JOIN: traz todas as linhas de b e apenas as linhas de a que satisfazem a condição
- FULL JOIN: traz todas as linhas
- Observação: as linhas onde não houver informação correspondente virão com NULL



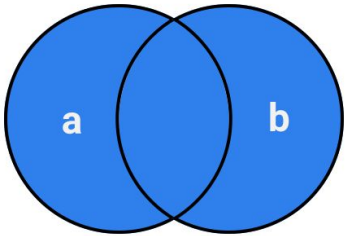
a INNER JOIN b



a LEFT JOIN b



a RIGHT JOIN b



a FULL OUTER JOIN b

INNER JOIN

Solução da situação problema 1

```
SELECT
V.nome_fantasia AS varejista,
P.nome AS nome,
P.descricao AS descricao,
P.preco AS preco
FROM
produto AS P
INNER JOIN
loja AS L
ON P.codigo_loja = L.codigo_loja
INNER JOIN
varejista AS V
ON L.cnpj_varejista = V.cnpj
```

LEFT (OUTER) JOIN

Solução da situação problema 2

Observação: foram comparados o nome e o telefone pois uma das duas listas pode estar desatualizada e o número pode ter mudado de proprietário, ou podemos ter duas pessoas com o mesmo nome.

```
SELECT
C.nome AS nome,
C.ddd_telefone AS ddd_telefone,
L.email AS email,
L.endereco AS endereco
FROM
contato_celular AS C
LEFT JOIN
contato_lista_telefonica AS L
ON C.ddd_telefone = L.ddd_telefone
AND C.nome = L.nome
```

RIGHT (OUTER) JOIN

Outra solução da situação problema 2

Observação: RIGHT JOIN é igual ao LEFT JOIN porém com a ordem das tabelas invertida, se escrito na ordem contrária da anterior o resultado é igual. LEFT JOIN é mais usado que RIGHT JOIN.

```
SELECT
L.nome AS nome,
L.ddd_telefone AS ddd_telefone,
L.email AS email
FROM
contato_lista_telefonica AS L
RIGHT JOIN
contato_celular AS C
ON C.ddd_telefone = L.ddd_telefone
AND C.nome = L.nome
```

FULL (OUTER) JOIN

O FULL JOIN é obtido através de uma UNION do resultado de um LEFT JOIN e de um RIGHT JOIN (sem inverter a ordem das tabelas entre eles).

LOAD DATA

Carregar dados de um CSV para dentro de uma tabela:

LOAD DATA LOCAL INFILE '*caminho_arquivo*' **INTO TABLE** *nome_tabela*

[**FIELDS TERMINATED BY** '*caractere_separador*']

[**ENCLOSED BY** '*caractere_aspas*']

[**LINES TERMINATED BY** '{\n | \r\n}']

[**IGNORE** *numero_linhas* **LINES**]

[(*nome_coluna1*, *nome_coluna2*, *nome_coluna3*, ...)];