

# Banco de Dados

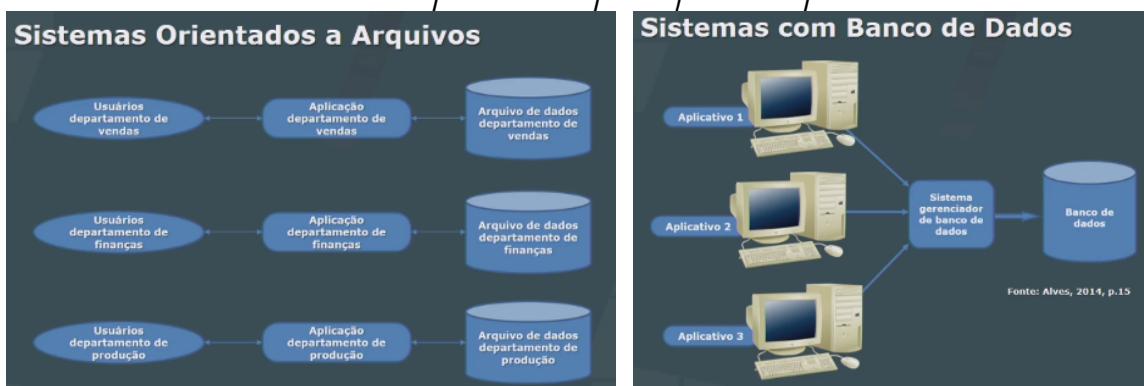
## Definição:

Lugar utilizado para armazenar os dados, precisa estar organizado e permitir manipular os dados, gera informações.

Podem ser manuais (como fichários, arquivos) ou computadorizados (como registros de computador).

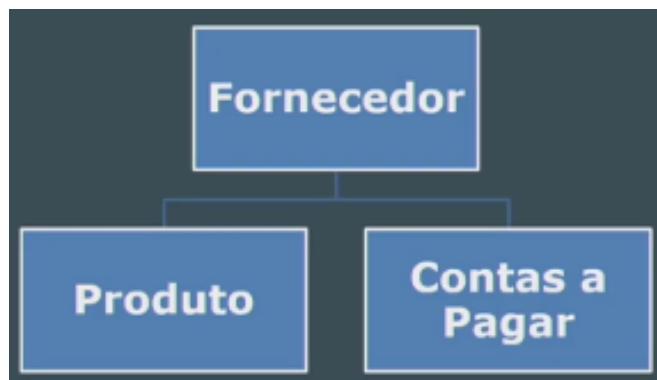
Surgiu da necessidade de registrar dados: pinturas pré-históricas, hieróglifos, pedras para contar ovelhas são bancos de dados.

1º bd computacional: 1889, Herman Hollerith, cartão perfurado lido por máquina usado para o censo dos EUA em 1890.



## Modelos de Dados:

Hierárquicos: mais antigo, tipo árvore, pai e filho. Cada filho só tem um pai, e o caminho pra chegar no fim é longo, precisa passar por todos os pais.



Relacionais: mais comum, até hoje, é composto de tabelas (linhas e colunas), e não tem dados repetidos, só relacionados.

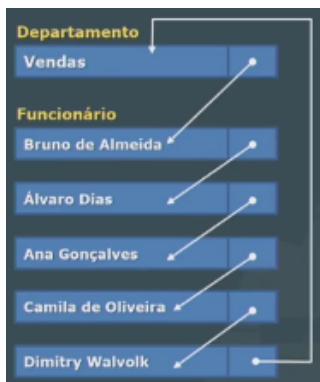
Tabela de produtos					
Produtos					
Código Produto	CódigoCategoria	CódigoFornecedor	Nome Produto	Preço	
0101231	3	2	Teclado musical ExpertMusic	650	
123456	1	3	Aparelho de som Quasar	230	
5123511	4	1	Jogo de dormitório Colibri	1230	
*				0	

Tabela de fornecedores						
Fornecedores						
CódigoFornecedor	NomeFornecedor	Endereço	Bairro	Cidade	Estado	
1	ABC Móveis Domésticos Ltda	R. Doze, 120	Centro	São Paulo	SP	
2	Brinquedos & Jogos Educar	Av. das Nações, 280	Jd. América	Atibaia	SP	
4	SomMaster	Av. do Lago	Jd. do Lago	Osasco	SP	
*	(Novo)					

Orientado a objetos: nasceu em 80, utiliza dados mais complexos. Pode usar tabelas também.

Rede: parecido com o hierárquico, mas pode ter mais de um pai, então não tem hierarquia, dá pra ir mais direto ao ponto.



## Sistema Gerenciador de Banco de Dados:

SGBD > banco de dados.

Utilizado para armazenar e gerenciar dados, é a cabeça de tudo. O banco de dados é só uma ferramenta, algo que o SGBD usa. Ele fica entre o banco de dados e as aplicações.

<b>1960</b> <b>Charles Bachman</b>	<b>Projetou o primeiro SGBD, o de Depósito de Dados Integrados</b>
<b>Final de 1960 IBM</b>	<b>SGBD utilizava o modelo hierárquico e permitindo acesso multiusuário através de uma rede</b>
<b>1970</b> <b>Edgard Codd (IBM)</b>	<b>Desenvolveu o banco de dados relacional, sendo um divisor de águas dos SGBD</b>
<b>1976</b> <b>Peter Chen</b>	<b>Criou o modelo de entidade e relacionamento (MER)</b>
<b>1974-1979</b> <b>Projetos System R (IBM)</b>	<b>Criou a linguagem SQL para banco de dados relacional, passando a ser a linguagem padrão de consulta.</b>

<b>Era da Internet</b>	<b>A primeira geração de sites armazenavam seus dados em arquivos dos sistemas operacionais</b>
<b>Hoje</b>	<b>O que impulsiona atualmente são muitas ideias diferentes, entre elas temos: banco de dados multimídia, vídeo interativo, fluxos de dados, bibliotecas digitais, etc. Além do desejo das empresas em minerar seus repositórios de dados por informações úteis sobre seus negócios</b>

SQL = linguagem padrão dos bancos de dados relacionais. Foi padronizada em ANSI e ISO. Novos padrões são chamados de dialetos.

## Modelagem de Dados:

Planejamento da execução das ideias do negócio para os termos computacionais. É pegar tudo o que a área de negócios disse que precisa e pensar em como fazer isso no computador, em que dados serão armazenados ou não.

Acontece em 3 níveis de abstração de dados: conceitual, lógico e físico.

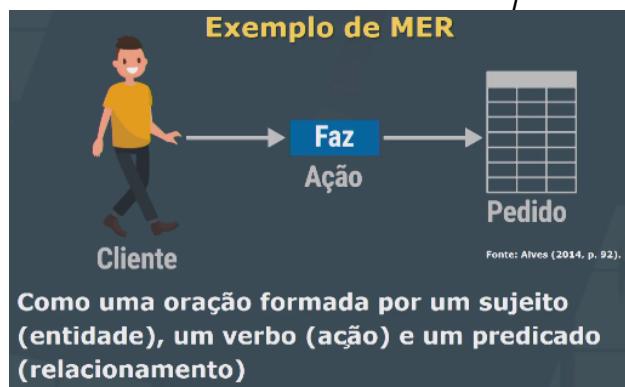
**Conceitual:** o cliente, que quer o banco de dados, e o analista, que entende tanto o cliente quanto o técnico, é o intérprete, conversam e decidem o que é necessário, o que vai ser entregue. É teórico, conversa.

**Lógico:** analista, que diz o que o cliente quer, e profissional do banco de dados, que cria o banco de dados.

**Físico:** profissional do banco de dados, que sabe fazer.

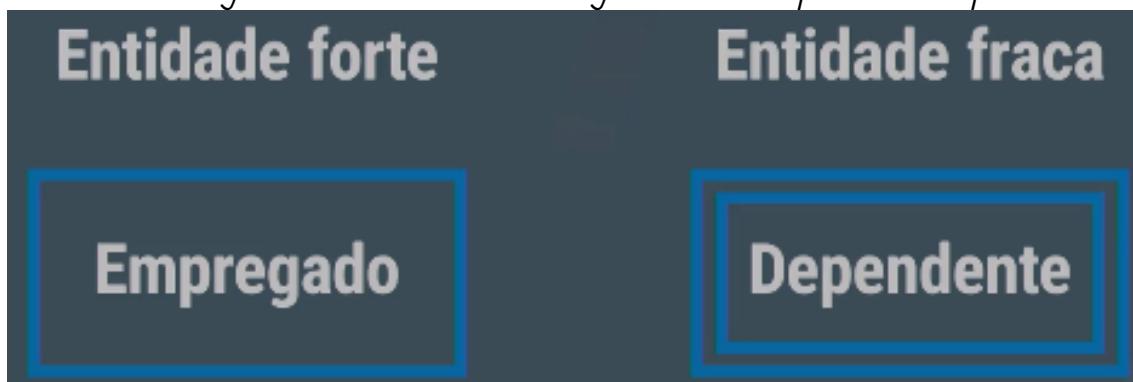
## Modelo de Entidade e Relacionamento:

O MER foi criado por Peter Chen em 1970, é conceitual. E o DER em 1976, é a representação gráfica.

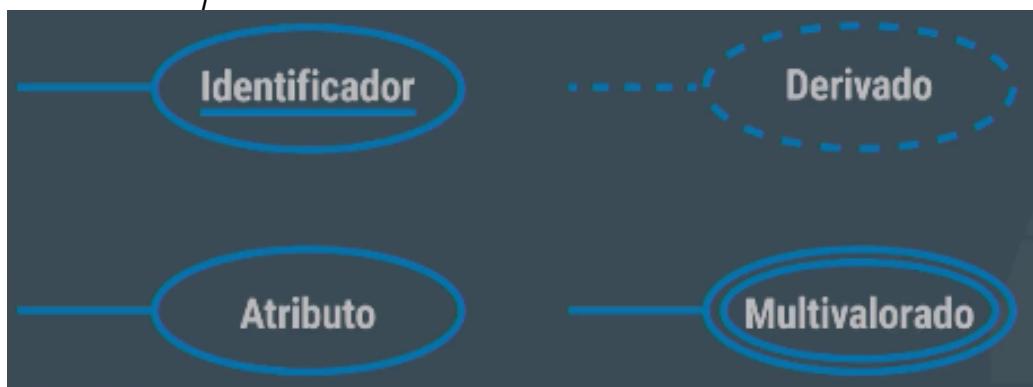


Entidade: sujeito do MER. Pode ser objetos concretos, pessoas ou eventos, como clientes, fornecedores, pagamentos, vendas...

Notação: entidade forte (um retângulo) e entidade fraca (dois retângulos). A fraca depende da forte para existir.



Atributos: descrevem as características de uma entidade. Há o identificador (só há um no mundo, como matrícula ou cpf), atributo (há mais pessoas com isso, mas só um disso para essa pessoa, como nascimento), derivado (vem de outro atributo, como idade, que vem de nascimento) e multivalorado (pode haver mais de um, vários valores, como telefones).



Relacionamento: relação ou associação entre entidades. Relacionamento forte (um losango) e relacionamento fraco (dois losangos). O relacionamento fraco ocorre quando há uma entidade fraca.

Cardinalidade: quantificação de um relacionamento, determinada com base no negócio. O número de ocorrências pode ser zero, uma ou muitas. O sentido da ocorrência é de ida e volta. X é a mínima (diz o número mínimo de ocorrências) e Y é a máxima (diz o número máximo de ocorrências). Sempre colocado do lado oposto à entidade.



Cardinalidade	Descrição
1:1	Um elemento de uma entidade se relaciona com um elemento de outra entidade.
1:N ou N:1	Um elemento de uma entidade pode se relacionar com mais de um elemento de outra entidade.
N:N	Vários elementos de uma entidade podem se relacionar com vários elementos de outra entidade e vice-versa.
0:1	Um elemento de uma entidade pode se relacionar a nenhuma ou um ocorrência de outra entidade.
0:N	Um elemento de uma entidade pode se relacionar a nenhuma ou muitas ocorrências de outra entidade.



### Modelo Lógico:

Detalha os atributos, chaves de acesso e integridade referencial, traduz o conceito. Não usa hardware, e nem define tamanhos ou tipos de dados.

**Cientes (Codigo\_Ciente, Nome\_Ciente, CPF, Endereço)**  
**Produtos (Codigo\_Produto, Descrição\_Produto,**  
**Unidade\_Medida, Preco\_Unitário)**

### Modelo Físico:

Etapa final, mostra a estrutura pro armazenamento físico dos dados. Mostra o tipo de dado, o tamanho e a chave primária do atributo.

```

CREATE TABLE Clientes (
    Código_Ciente INTEGER,
    Nome_Ciente VARCHAR (50),
    CPF CHAR (14),
    Endereço VARCHAR (50),
    PRIMARY KEY (Código_Ciente) );

CREATE TABLE Produtos (
    Código_Produto CHAR (13),
    Descrição_Produto VARCHAR (50),
    Unidade_Medida CHAR (3),
    Preço_Unitário DECIMAL (10, 2),
    PRIMARY KEY (Código_Código) );
  
```

Modelo lógico x Modelo físico	
Modelo lógico	Modelo físico
<b>Motorista</b>	<b>Motorista</b>
<b>cod_motorista</b>	<b>cod_motorista:</b> NUMBER(5)
<b>nome_nascimento</b>	<b>nome_motorista:</b> VARCHAR(40)
<b>data_nascimento</b>	<b>data_nasc:</b> DATE
<b>CPF_motorista</b>	<b>CPF:</b> NUMBER
<b>sexo</b>	<b>sexo:</b> CHAR(1)

Fonte: Adaptado de Puga (2013, p. 163).

### Modelo Relacional:

Transforma o modelo de Entidade de Relacionamento em relacional. Entidade vira tabela, atributos colunas, ...

Modelo ER	Modelo Relacional
ENTIDADE	TABELA
ATRIBUTO SIMPLES	COLUNA
ATRIBUTO DERIVADO	COLUNA
ATRIBUTO IDENTIFICADOR	CHAVE PRIMÁRIA (OU SECUNDÁRIA)
ATRIBUTO MULTIVALORADO	NOVA TABELA E CHAVE ESTRANGEIRA
RELACIONAMENTO 1:1 OU 1:N	CHAVE ESTRANGEIRA
RELACIONAMENTO N:N	NOVA TABELA COM 2 CHAVES ESTRANGEIRAS
CONJUNTO DE VALORES	TIPOS DE DADOS

### Principais tipos de dados (MySQL) – *Data Types*

- **INT** – números inteiros
- **CHAR** – texto com tamanho fixo
- **VARCHAR** – texto
- **DATE** - data
- **FLOAT** - números fracionados

### Linguagem SQL:

Structured Query Language = linguagem de consulta estruturada.

Iniciada por Edgar Frank Codd em 1974-1979, e criada pela IBM em 1986.

São comandos para criação de bancos de dados e tabelas, um padrão de consulta, de língua, para todos os bancos de dados.

Padronização do SQL: tem o ISO e ANSI, ambos dialetos do SQL puro.

Com SQL é possível: criar, alterar e remover elementos do banco de dados; inserir, alterar e apagar dados; consultar o banco de dados; controlar o acesso dos usuários; obter a garantia da consistência e integridade dos dados.

SQL:86	Primeira versão da linguagem, lançada em 1986, consiste basicamente na versão inicial da linguagem criada pela IBM.
SQL:92	Lançada em 1992, inclui novos recursos tais como tabelas temporárias, novas funções, expressões nomeadas, valores únicos, instrução case etc.
SQL:1999	Lançada em 1999, foi a versão que teve mais recursos novos significativos, entre eles: a implementação de expressões regulares, recursos de orientação a objetos, queries recursivas, triggers, novos tipos de dados (boolean, LOB, array e outros), novos predicados etc.
SQL:2003	Lançada em 2003, inclui suporte básico ao padrão XML, sequências padronizadas, instrução MERGE, colunas com valores autoincrementais etc.
SQL:2006	Lançada em 2006, não inclui mudanças significativas para as funções e comandos SQL. Contempla basicamente a interação entre SQL e XML.

**Quadro 1 – Categorias de instrução DDL**

<b>Finalidade</b>	Definição e manutenção das estruturas do banco de dados, tais como a criação do próprio banco de dados e das tabelas que o compõem, além das relações entre as tabelas e os objetos do banco de dados.
<b>Instruções</b>	
Create	Criação de estruturas de objetos do banco de dados.
Alter	Alteração da estrutura de objetos do banco de dados.
Drop	Eliminação das estruturas de objetos do banco de dados.
Truncate	Exclusão física de linhas de tabelas.
Rename	Renomeação de objetos do banco de dados.
Comment	Inclusão de comentários aos objetos do banco de dados.

**Quadro 2 – Categorias de instrução DML**

<b>Finalidade</b>	Consultas, inserções, exclusões e alterações em um ou mais registros, de uma ou mais tabelas, de maneira simultânea.
<b>Instruções</b>	
Insert	Inserção de dados.
Update	Alteração de dados.
Delete	Exclusão de dados.
Select	Consulta de dados.
Merge	Combinação das instruções insert, update e delete.

**Quadro 3 – Subcategoria de instrução DCL**

<b>Finalidade</b>	Controle dos privilégios de usuários, de forma que o administrador do banco de dados possa determinar o nível de acesso de um usuário aos objetos do banco de dados, concedendo privilégios ou retirando esse acesso e revogando os privilégios.
<b>Instruções</b>	
Grant	Atribuição de privilégios aos usuários do banco de dados.
Revoke	Revogação de privilégios dos usuários do banco de dados.

**Quadro 4 – Categorias de instrução TCL**

<b>Finalidade</b>	Controle de transações, consideradas o conjunto de uma ou mais operações DML realizadas no banco de dados.
<b>Instruções</b>	
Commit	Confirmação das manipulações.
Rollback	Desistência das manipulações.
Savepoint	Criação de pontos para o controle das transações.

### Data Definition Language - DDL:

Cuida da estrutura do banco de dados. O banco de dados precisa ter ao menos uma tabela, com ao menos uma coluna.

## Criando o banco de dados

Nos nomes de bancos e tabelas, não são permitidos espaços. Use o underline.

**CREATE DATABASE sistema;**

CREATE DATABASE é o comando

O nome do banco de dados é sistema

Os comandos devem terminar com ponto-e-vírgula

> CREATE DATABASE sistema;  
Query OK, 1 row affected (0.01 sec)

Query OK é o retorno que o SGBD traz quando a instrução é executada com sucesso

## Escolhendo o banco de dados

**USE sistema;**

A partir da instrução executada, todos os comandos serão executados na base de dados selecionada (sistema).

> USE sistema;  
Database changed

Database changed é a confirmação de que a base foi alterada.

## Criando tabelas

O parênteses inicia a lista de colunas a serem criadas

Comando para criar a tabela

Nomes da tabela sem espaços, com underline

Nome da primeira coluna a ser criada

**CREATE TABLE cad\_aluno**

Nome da segunda coluna a ser criada

**( cod\_aluno varchar(5), nome varchar(30) );**

Fechamento da lista de colunas que vão ser criadas. O ponto-e-vírgula determina o fim do comando

A vírgula separa as colunas que vão ser criadas  
Tipo e tamanho máximo da coluna. No exemplo, a coluna será do tipo varchar (texto) com tamanho máximo de 30 caracteres

> DESCRIBE cad\_aluno;

Field	Types	Null	Key	Default	Extra
cod_aluno	Varchar(5)	YES		NULL	
nome	Varchar(30)	YES		NULL	

2 rows in set (0.01 sec)

## Campos nulos

O comando NOT NULL informa que o preenchimento da coluna é obrigatório

```
CREATE TABLE cad_aluno
(
    cod_aluno varchar(5) NOT NULL,
    nome      varchar(30) NOT NULL
);
```

## Chave primária

A coluna cod\_material foi escolhida para ser a chave primária, ela não pode se repetir

```
CREATE TABLE materia
(
    cod_material int NOT NULL,
    data_inicio date NOT NULL,
    nome        varchar(40) NOT NULL,
    PRIMARY KEY (cod_materia)
);
```

Criando a chave primária. Entre parênteses o nome da coluna escolhida

Fonte: F

## AUTO\_INCREMENT

Definindo a coluna cod\_turno como auto\_incremente

```
CREATE TABLE turno
(
    Cod_turno int NOT NULL AUTO_INCREMENT,
    nome_turno varchar(20) NOT NULL
    PRIMARY KEY (cod_cor)
);
```

## Renomeando tabelas

Antigo nome da tabela Novo nome da tabela

```
ALTER TABLE cad_aluno RENAME alunos;
```

Fonte: Próprio Autor

## Alterando o tamanho de uma coluna

Comando para modificar Novo tamanho

```
ALTER TABLE alunos MODIFY nome varchar(70);
```

## Adicionando colunas

Fonte: Próprio Autor  
ALTER TABLE aluno ADD COLUMN email varchar(30);

## Modificando o AUTO\_INCREMENT

Fonte: Próprio Autor  
ALTER TABLE alunos AUTO\_INCREMENT = 15;

## Excluindo colunas

Fonte: Próprio Autor  
ALTER TABLE aluno DROP COLUMN email;

## Excluindo tabelas

Fonte: Próprio Autor  
DROP TABLE é o comando  
DROP TABLE cad\_alunos;

## Excluindo DATABASES

Fonte: Próprio Autor  
DROP DATABASE é o comando  
DROP DATABASE sistema;

## Data Manipulation Language - DML:

### Inserindo registros

Fonte: Próprio autor  
Comando para inserir um registro  
Nome da tabela que vai ser inserida  
Coluna 1 Coluna 2  
VALUES faz parte da sintaxe do comando  
Value coluna 1 Value coluna 2  
INSERT INTO alunos (cod\_aluno, nome)  
values (459, 'João Maria');  
> INSERT INTO alunos (cod\_aluno, nome)  
Values (459, 'João Maria');  
Query OK, 1 row affected (0.02 sec)

### Omitindo colunas

Fonte: Próprio autor  
INSERT INTO alunos values (459, 'João Maria');

A ordem dos dados é EXATAMENTE a ordem que está na tabela  
TODAS as colunas da tabela devem estar incluídas

### Inserindo vários registros

Fonte: Próprio autor  
Coluna 1 Coluna 2  
INSERT INTO alunos (cod\_aluno, nome)  
Values (459, 'João Maria'),  
(89, 'Pedro'),  
(981, 'Maria Paula');  
Value coluna 1 Value coluna 2  
> INSERT INTO alunos (cod\_aluno, nome)  
values (459, 'João Maria'), (89, 'Pedro'),  
(981, 'Maria Paula');  
Query OK, 3 rows affected (0.38 sec)

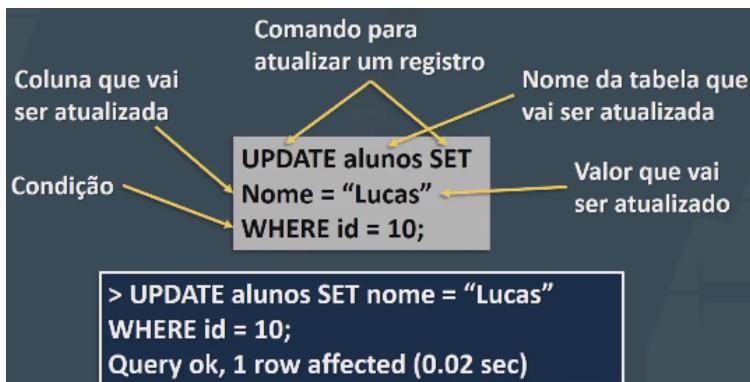
### Consulta simples

Fonte: Próprio autor  
Comando para selecionar registros da tabela  
SELECT \* FROM alunos;  
\* quer dizer todas as colunas

> SELECT \* FROM alunos;  

cod_aluno	nome
459	João Maria
89	Pedro
981	Maria Paula

  
3 rows in set (0.00 sec)



### Escolhendo as colunas

Nome da coluna selecionada

```

SELECT nome FROM alunos;
    
```

Output:

```

> SELECT nome FROM alunos;
+-----+
| nome |
+-----+
| João Maria |
| Pedro   |
| Maria Paula |
+-----+
3 rows in set (0.00 sec)
    
```

### Restringindo consultas

Comando informando que vai ter uma condição

```

SELECT * FROM alunos
WHERE cod_aluno > 300;
    
```

Output:

```

> SELECT * FROM alunos
WHERE cod_aluno > 300;
+-----+-----+
| cod_aluno | nome |
+-----+-----+
| 459       | João Maria |
| 981       | Maria Paula |
+-----+-----+
2 rows in set (0.00 sec)
    
```

### Ordenar resultados

Comando para ordenar

Tipo de ordenação: ASC ou DESC

```

SELECT*FROM alunos ORDER BY nome ASC;
    
```

cod	nome	idade	sexo	função
1	Augusto	30	M	pintor
4	César	36	M	pintor
5	João Vitor	18	M	auxiliar
3	Maria	45	F	recepção
2	Paula	23	F	gerente

### BETWEEN

Output:

```

SELECT * FROM alunos WHERE idade BETWEEN 20 AND 35 ;
    
```

cod	nome	idade	sexo	função
1	Augusto	30	M	pintor
2	Paula	23	F	gerente

### LIKE

- % (porcentagem)
- \_ (underline)

Output:

```

SELECT nome FROM alunos
WHERE nome LIKE '%a';
    
```

Output:

nome
Paula
Maria

Fonte: Próprio autor

Comando dentro da cláusula WHERE

Palavra que vai ser procurada

```
SELECT nome FROM alunos
WHERE nome LIKE 'c%' ;
```

nome
César

Fonte: Próprio autor

```
SELECT nome FROM alunos
WHERE nome LIKE '%i%' ;
```

nome
Maria
João Vitor

nome
Paula
Maria

Fonte: Próprio autor

```
SELECT nome FROM alunos
WHERE nome LIKE '_a%' ;
```

```
SELECT nome FROM alunos
WHERE idade > 18
AND sobrenome LIKE 'f%' ;
```

## NOT LIKE

Negação  
Inverso

```
SELECT nome FROM alunos
WHERE nome NOT LIKE '_a%' ;
```

NOT = negação,  
não padrão

nome
César
Augusto
João Vitor

Fonte: Próprio autor

## Apagando registros

Comando para  
apagar registros

```
DELETE FROM alunos ;
```

Nome da tabela  
que vai ser deletado

```
> DELETE FROM alunos ;
Query OK, 5 rows affected (0.08 sec)
```

Join:

Join: recupera dados em várias tabelas.

## Utilizando JOIN

A sintaxe muda  
para tabela.coluna  
JOIN e ON fazem  
parte da sintaxe  
Primeira tabela  
Segunda tabela  
Condição para que  
haja uma relação

nome	nome
João	Curitiba
Maria	Curitiba
Pedro	Iratí
Ana	Brasília

nome → nome da  
coluna da tabela alunos

Registros da  
tabela alunos

nome → nome da  
coluna da tabela cidade

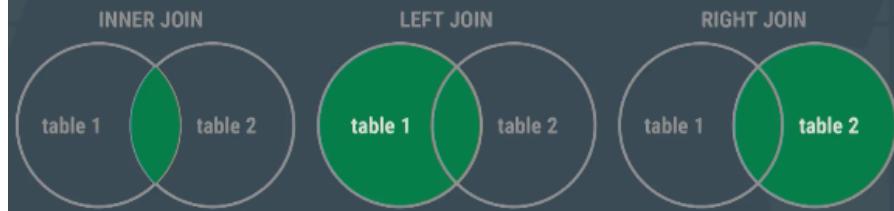
Registros da  
tabela cidade

## Utilizando RIGHT JOIN

```
SELECT alunos.*, cidade.* FROM alunos RIGHT
JOIN cidade ON alunos.cod_cidade = cidade.id ;
```

id	nome	id_cidade	id	nome
1	João	1	1	Curitiba
2	Maria	1	1	Curitiba
3	Pedro	2	2	Iratí
4	Ana	3	3	Brasília
NULL	NULL	NULL	5	Cuiabá

- Ao contrário do RIGHT JOIN, a cláusula LEFT JOIN retorna todos os dados encontrados na tabela à esquerda de JOIN (Camargo, 2010).



**Apelidando colunas**

As colunas podem receber apelidos para facilitar sua identificação

"nome aluno" é o apelido dado à coluna nome da tabela alunos

"cidade" é o apelido dado à coluna nome da tabela cidade

```
SELECT alunos.nome as 'nome aluno', cidade.nome as 'cidade'
FROM alunos JOIN cidade ON alunos.cod_cidade = cidade.id;
```

nome aluno	cidade
João	Curitiba

## Média (AVG)

```
SELECT avg(salario)
FROM funcionários ;
```

**avg (salario)**

**4475.000000**

## GROUP BY

```
SELECT sexo, avg(salario) FROM
funcionarios GROUP BY sexo;
```

sexo	avg(salario)
M	4650.000000
F	4300.000000

## Soma (SUM)

```
SELECT sum(salario)
FROM funcionários ;
```

**sum(salario)**

**17900.00**

## Número de registros (COUNT)

```
SELECT count(*) as  
'funcionarios' FROM  
funcionarios ;
```

funcionarios

4

## Máximo (MAX) e Mínimo (MIN)

```
SELECT max(salario)  
FROM funcionarios ;
```

max(salario)

5300.00

```
SELECT min(salario)  
FROM funcionarios ;
```

min(salario)

3300.00

## Data Control Language - DCL

### Criar usuário

- Somente um usuário com permissão para criar um novo usuário pode executar o comando

```
CREATE USER 'chefe' IDENTIFIED BY '123' ;
```

### Excluindo usuário

Exemplo:

```
DROP USER 'chefe' ;
```

## ■ Sintaxe

**GRANT direitos ON nome\_tabela TO identificação ;**

Fonte: O autor

### • Exemplo

**GRANT select ON alunos TO chefe ;**

#### Lista de permissões

CREATE	criar tabelas ou databases
DROP	deletar tabelas ou databases
DELETE	deletar registros
INSERT	inserir registros
SELECT	selecionar registros
UPDATE	atualizar/modificar registros

Fonte: O autor

**GRANT select, insert, delete, update ON alunos TO chefe ;**

#### Nível de privilégio

*.*	Privilégio global
db.*	Qualquer tabela do banco db
db.tb	Apenas a tabela tb do banco de dados db. Para especificar apenas algumas colunas de determinada tabela, estas deverão ser listadas ao lado do privilégio <a href="#">(priv (colunas))</a>

## ■ Exemplos

**GRANT all privileges ON \*.\* TO chefe ;**

Fonte: O autor

**GRANT all privileges ON sistema.\* TO chefe ;**

#### REVOKE

- Remove os privilégios de um usuário
- Utiliza a mesma lista de permissões e os mesmos níveis de privilégios do comando GRANT

**REVOKE select ON alunos FROM chefe ;**

## ■ Exemplo de remoção de todos os privilégios no banco de *dados sistema*, do usuário-chefe

**REVOKE all privileges ON sistema.\* FROM chefe ;**

Transact Control Language - TCL:

Comandos TCL	Exemplo de transação
<b>Begin:</b> indica o início de uma transação	BEGIN TRANSACTION;
<b>Commit:</b> é o fim da transação, executando as instruções no banco de dados (permanente)	UPDATE CONTA_CORRENTE set saldoConta= saldoConta - @Valor where numConta = @contaDe;
<b>Rollback:</b> é o fim da transação também, mas cancela todas as alterações efetuadas porque algo deu errado	UPDATE CONTA_CORRENTE set saldoConta= saldoConta + @Valor where numConta = @contaPara; COMMIT;

Índices:

Um select menor. Ele procura o que é pedido só na coluna que é especificada, não em todo o banco de dados.

<b>Simples</b>  <b>CREATE INDEX 'NomeFuncionario'</b> <b>ON 'funcionarios' (nome);</b> <small>Fonte: O autor</small>	<b>CREATE TABLE funcionarios</b> ( cod INT(3) auto_increment, nomeVARCHAR(30), sobrenome VARCHAR(30), sexo CHAR, salario DECIMAL(6,2), PRIMARY KEY (cod), INDEX NomeFuncionario (sobrenome) );
<b>Multicolumna</b>  <b>CREATE INDEX 'NomeFuncionario'</b> <b>ON 'funcionarios' (nome, sobrenome);</b>	

Stored Procedure - Procedimento de Armazenagem:

Usada para validação de dados, execução de instruções SQL, controle de acessos, recebimento de parâmetros, e retorno de valores.



## Pontos negativos

**Segundo Rodrigues (2013):**

- Necessidade de maior conhecimento da sintaxe do banco de dados para escrita de rotinas em SQL
- As rotinas ficam mais facilmente acessíveis. Alguém que tenha acesso ao banco poderá visualizar e alterar o código

## Pontos positivos

Simplificação da execução de instruções SQL pela aplicação

Transferência de parte da responsabilidade de processamento para o servidor

Facilidade na manutenção, reduzindo a quantidade de alterações na aplicação

### Delimiter

**Padrão**

É utilizado para trocar o caractere de finalização

**DELIMITER \$\$**

—Seu objetivo 1 aqui  
\$\$  
—Seu objetivo 2 aqui  
\$\$

### Criando Procedures

**CREATE PROCEDURE nome\_procedure (parâmetros)**  
Declarções;

### Chamando Procedures

**CALL nome\_procedure (parâmetros)**

### Parâmetros

**IN** → é utilizado apenas para recebimento de dados, não é utilizado para feedback

**OUT** → é um parâmetro de saída, não sendo informado um valor fixo (direto), apenas uma variável para o retorno

**INOUT** → esse modo de parâmetro pode ser utilizado como entrada ou saída, não podendo ser informado um valor fixo

### Exemplo com parâmetros

```
CREATE PROCEDURE AcrescInventory (
    IN Livro_isbn CHAR (10),
    IN quantidadeAcresc INTEGER)
UPDATE Livros
    SET quantidade_em_estoque = quantidade_em_estoque - quantidadeAcresc
    WHERE livro_isbn = isbn
```

### Apagando procedure

**DROP PROCEDURE nome\_procedure;**

Function - Função:

### Criando uma Function

**CREATE FUNCTION nome\_funcao (parâmetros)**  
**RETURNS tipo\_dados**  
código da função

### Exemplo Function

```
CREATE FUNCTION fn_teste (a DECIMAL (10,2), b INT)
RETURNS INT
RETURN a * b;
```

Fonter: Reis, 2013

### Chamando uma Function

**SELECT nome\_funcao (parâmetros);**

**SELECT fn\_teste(2.5, 4) AS Resultado;**

# Apagando Function

**DROP FUNCTION nome\_funcao;**

## Triggers - Gatilhos:

Gatilhos automáticos sempre associados a uma tabela, disparados antes ou depois de um evento DML.

### Pontos negativos

Alguém que tenha acesso não autorizado ao banco de dados poderá visualizar e alterar o processamento realizado pelos gatilhos

Requer maior conhecimento de manipulação do banco de dados (SQL) para realizar as operações internamente

### Pontos positivos

Segundo Rodrigues (2016):

- Parte do processamento que seria executado na aplicação passa para o banco, poupano recursos da máquina-cliente
- Facilita a manutenção, sem que seja necessário alterar o código fonte da aplicação

### Sintaxe

nome → nome da trigger

momento → quando a trigger vai ser executada: *BEFORE* (antes) ou *AFTER* (depois)

evento → qual o comando que vai fazer a trigger disparar: *INSERT*, *UPDATE*, *DELETE* e *REPLACE*

tabela → é a tabela a que o trigger está associado

### Exemplo Sintaxe

```
CREATE TRIGGER nome momento evento
ON tabela
FOR EACH ROW
BEGIN
/*corpo do código*/
END
```

### Exemplo Trigger

```
CREATE TRIGGER tr_desconto BEFORE INSERT
ON Livros
FOR EACH ROW
SET NEW.preco_desconto=(NEW.preco_produto*0.90);
```

### Registros *NEW* e *OLD*

As palavras *NEW* e *OLD* são utilizadas para acessar os registros antes ou depois da execução da trigger

Pode-se acessar os registros que serão enviados para uma tabela antes (*BEFORE*) ou depois (*AFTER*) de um *UPDATE*

### Bianchi (2008) explica os registros *NEW* e *OLD*:

- ***INSERT*:** o operador *NEW.nome\_coluna* nos permite verificar o valor enviado para ser inserido em uma coluna de uma tabela. *OLD.nome\_coluna* não está disponível

**DELETE:** o operador OLD.nome\_coluna nos permite verificar o valor excluído ou a ser excluído. NEW.nome\_coluna não está disponível

**UPDATE:** tanto OLD.nome\_coluna quanto NEW.nome\_coluna estão disponíveis, antes (BEFORE) ou depois (AFTER) da atualização de uma linha

```
DELIMITER $  
CREATE TRIGGER Tgr_ItensVenda_Insert AFTER INSERT  
ON ItensVenda  
FOR EACH ROW  
BEGIN  
  
    UPDATE Produtos SET Estoque = Estoque - NEW.Quantidade  
    WHERE Referencia = New.Produto;  
END$  
  
CREATE TRIGGER Tgr_ItensVenda_Delete AFTER DELETE  
ON ItensVenda  
FOR EACH ROW  
BEGIN  
  
    UPDATE Produtos SET Estoque = Estoque + OLD.Quantidade  
    WHERE Referencia = OLD.Produto;  
END$  
  
DELIMITER;
```

## Apagando Triggers

**DROP TRIGGER nome\_da\_trigger;**

Stored Procedures Com Processos:

### Sintaxe Condicionais

```
IF<condição>THEN  
    comandos sql caso verdadeiro  
ELSE  
    comandos sql caso falso  
END IF
```

### Condisional

```
DELIMITER//  
CREATE PROCEDURE lista_clientes (IN opção integer)  
  
BEGIN  
    IF opção = 0 THEN  
        SELECT * FROM clientes where sexo = F;  
    ELSE  
        IF opção = 1 THEN  
            SELECT * FROM clientes WHERE sexo = M;  
        ELSE  
            SELECT*FROM clientes;  
        END IF;  
    END IF;  
END//
```

### Laço de Repetição

```
DELIMITER //  
CREATE PROCEDURE acumulador (limite TINYINT UNSIGNED)  
BEGIN  
    DECLARE contador TINYINT UNSIGNED DEFAULT 0;  
    DECLARE soma INT DEFAULT 0;  
    WHILE contador < limite DO  
        SET contador = contador + 1;  
        SET soma = soma + contador;  
    END WHILE;  
    SELECT soma;  
END//
```

```
CALL acumulador(10);
```