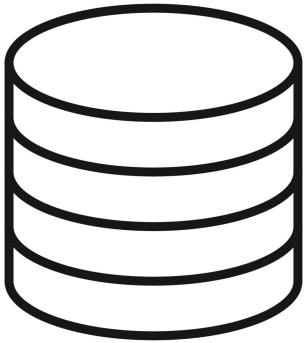
SQL (*Standard Query Language*)

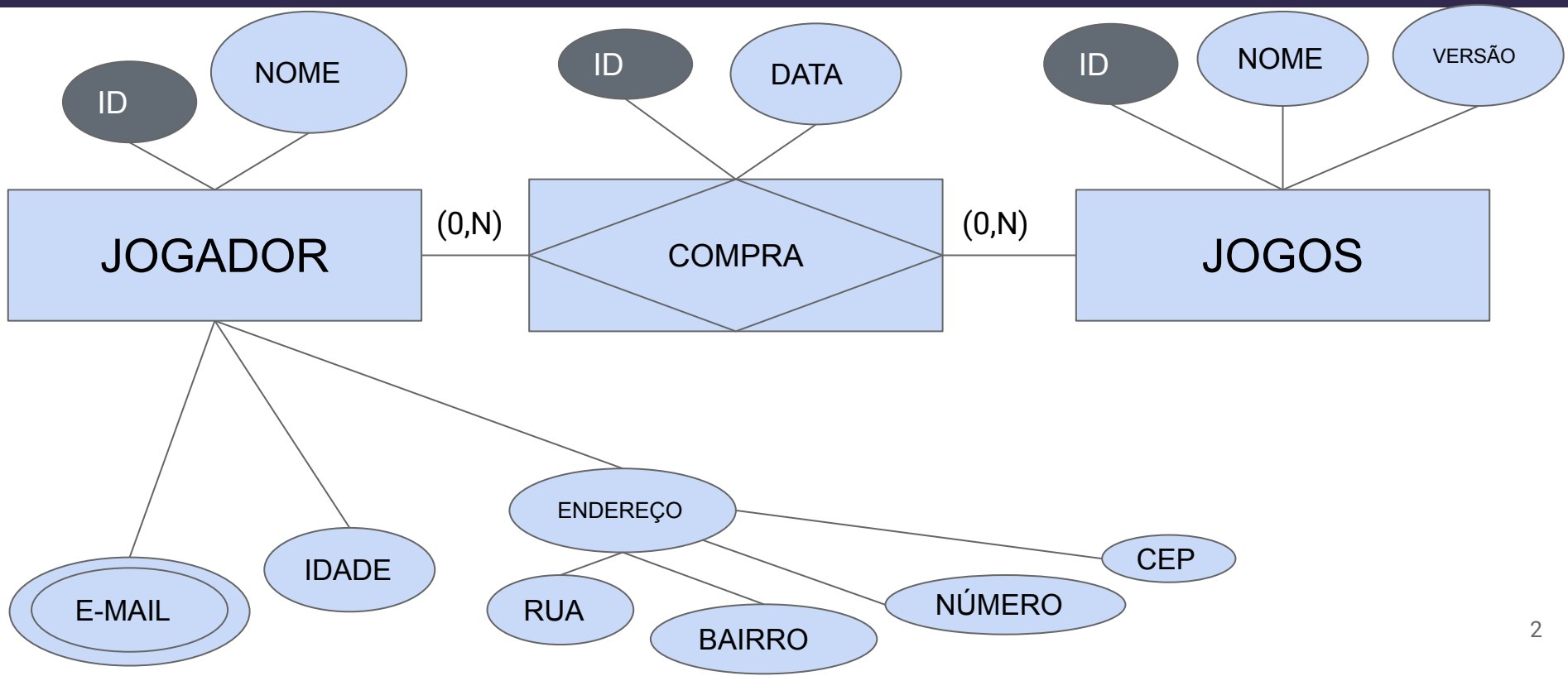


Banco de Dados

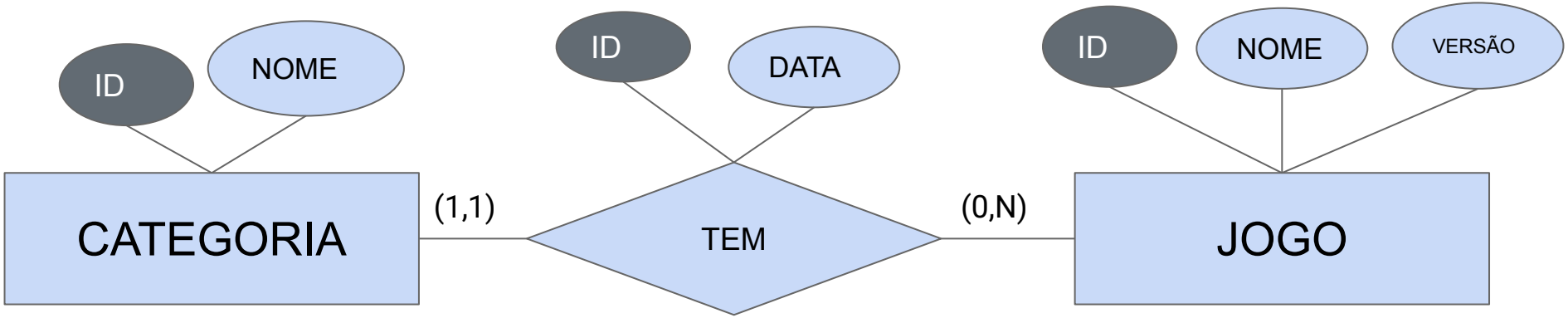
marisangila.alves@catolicasc.org.br

Marisangila Alves, MSc

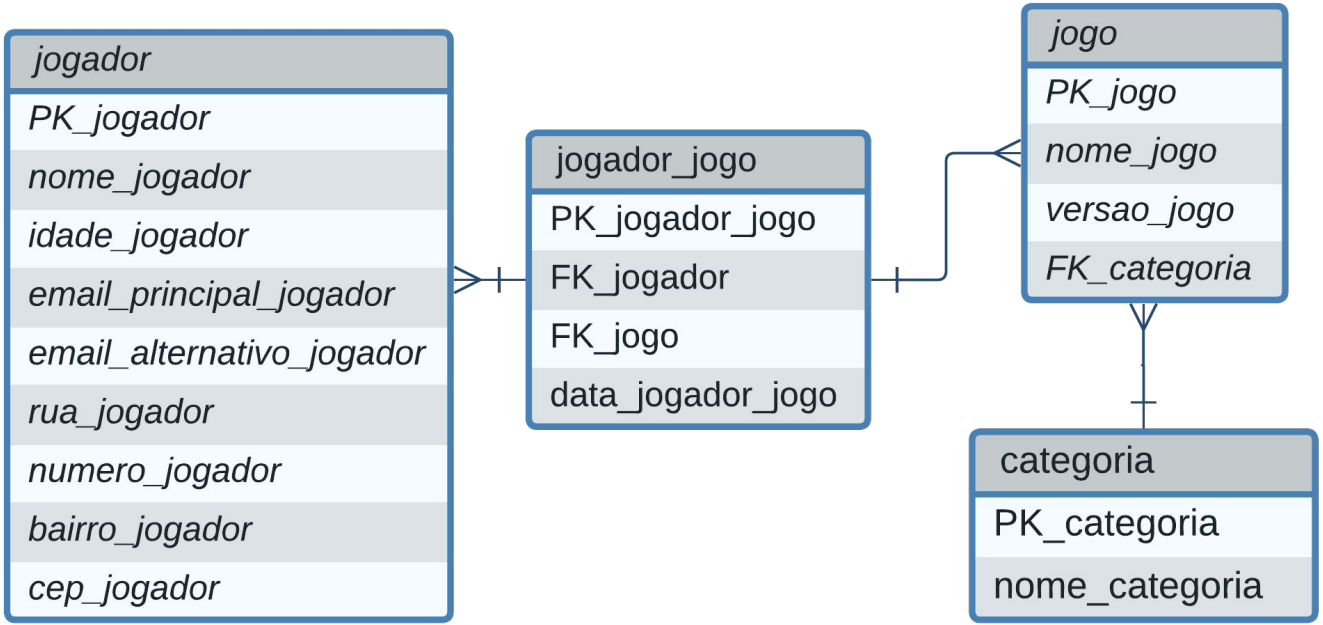
Modelo Conceitual - Diagrama Entidade Relacionamento



Modelo Conceitual - Diagrama Entidade Relacionamento



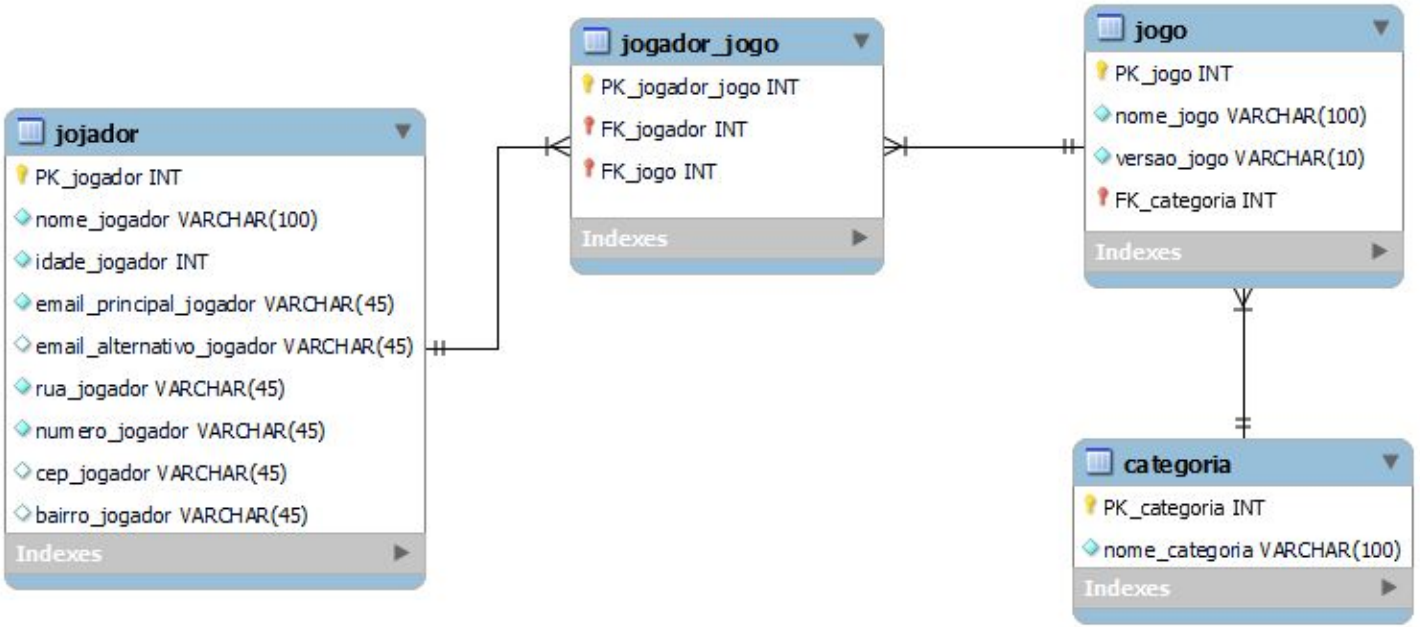
Modelo Lógico



Como definir a chave estrangeira

- Grau de cardinalidade:
  - (1, 1): A chave estrangeira deve ser inserida em qualquer uma das duas entidades.
  - (1, N): A chave estrangeira deve ser inserida na entidade de grau N.
  - (N, N): É necessário criar uma nova entidade (tabela) associativa com duas chaves estrangeiras. Uma para cada entidade que gerou o relacionamento.

Modelo Físico



Entidades

Entidades (Tabelas)

Jogador		
ID	Nome	Idade
1	Boby	18
2	Andy	20

Jogo		
ID	Nome	Versão
1	CS	1.2
2	LoL	8.9.2005
3	Minecraft	4.9.2002
4	Valorant	20.3.2005

Jogador_Jogos			
ID	Data	ID_Jogo	ID_Jogador
1	16/07/2022	1	1
2	22/07/2022	4	1
3	10/08/2022	3	2
4	24/08/2022	2	1

Atributos

Colunas (Atributos)

Atributo Identificador -> Chave Primária

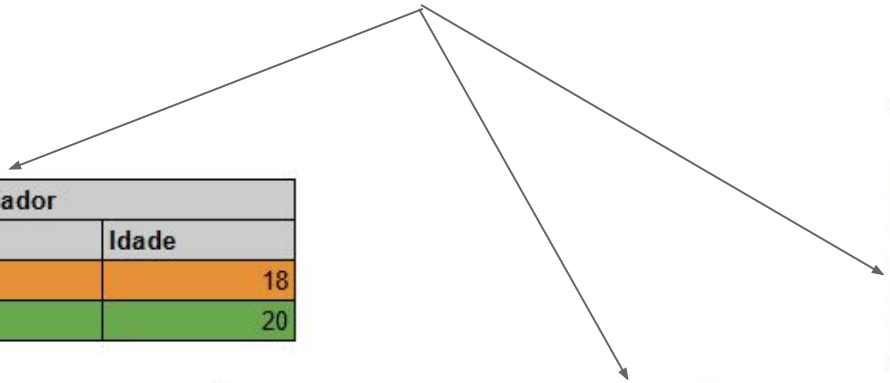
Jogador		
ID	Nome	Idade
1	Boby	18
2	Andy	20

Jogo		
ID	Nome	Versão
1	CS	1.2
2	LoL	8.9.2005
3	Minecraft	4.9.2002
4	Valorant	20.3.2005

Jogador_Jogos			
ID	Data	ID_Jogo	ID_Jogador
1	16/07/2022	1	1
2	22/07/2022	4	1
3	10/08/2022	3	2
4	24/08/2022	2	1

Relacionamento

Chaves Estrangeiras (Relacionamento)



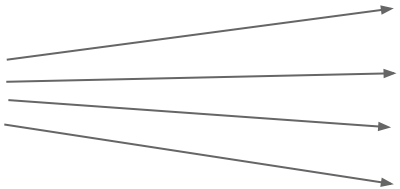
Jogador		
ID	Nome	Idade
1	Boby	18
2	Andy	20

Jogo		
ID	Nome	Versão
1	CS	1.2
2	LoL	8.9.2005
3	Minecraft	4.9.2002
4	Valorant	20.3.2005

Jogador_Jogos			
ID	Data	ID_Jogo	ID_Jogador
1	16/07/2022	1	1
2	22/07/2022	4	1
3	10/08/2022	3	2
4	24/08/2022	2	1

Dados

Linhas(Registros)



Jogo		
ID	Nome	Versão
1	CS	1.2
2	LoL	8.9.2005
3	Minecraft	4.9.2002
4	Valorant	20.3.2005

DDL - *Data Definition Language*

DDL - *Data Definition Language*

- CREATE
- DROP
- ALTER
- TRUNCATE
- COMMENT
- RENAME

Criar Banco de Dados

```
CREATE DATABASE escola;
```

```
USE escola;
```

CRIAR TABELA

```
CREATE TABLE nome_tabela(
   nome_campo1 tipo,
   nome_campo2 tipo
   );
CREATE TABLE aluno(
   PK_aluno INT,
   nome_aluno VARCHAR(100),
   email_aluno VARCHAR(100)
   );
```

CRIAR TABELA: Exemplo

```
CREATE TABLE aluno(
PK_aluno INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    nome_aluno VARCHAR(100) NOT NULL,
    email_aluno VARCHAR(200) NOT NULL,
    data_nascimento_aluno DATE NOT NULL,
    idade_aluno INT NULL,
    horario_entrada_aluno TIME NOT NULL,
    horario_saida_aluno TIME NOT NULL,
    mensalidade_aluno DECIMAL(10,2) NOT NULL,
    eh_bolsista_aluno BIT DEFAULT(0) NULL,
    ano_ingresso_aluno YEAR NOT NULL
);
```

CRIAR TABELA: Tipos de Dados

- INT: valores inteiros
- DECIMAL(MAXIMO_CASAS, MAXIMO_CASAS_DECIMAIS): valores decimais
    - DECIMAL(10,2)
- VARCHAR(MAXIMO): caractere
    - VARCHAR(50)
- BIT: valores binários (0 ou 1)

CRIAR TABELA: Tipos de Dados

- DATE: data
- TIME: horário
- DATETIME: data e horário
- YEAR: ano

CRIAR TABELA: Chave primária

PRIMARY KEY: Define a coluna como chave primária.

AUTO_INCREMENT: A inserção da chave primária é automática.

O banco de dados é responsável por inserir valores únicos.

CRIAR TABELA: colunas nulas

NULL: permite que uma coluna tenha valor nulo.

NOT NULL: valores nulos não são permitidos.

CRIAR TABELA: Valor default

DEFAULT: permite que um valor padrão seja definido.
É possível cobinar com o NULL.

Se o valor não for inserido porque valores nulos são permitidos, a coluna recebe o valor padrão

Adicionar chave estrangeira

```
CREATE TABLE turma(
    pk_turma int PRIMARY KEY AUTO_INCREMENT NOT NULL,
    nome_turma varchar(100) NOT NULL
);

CREATE TABLE aluno(
    pk_aluno int PRIMARY KEY AUTO_INCREMENT NOT NULL,
    nome_aluno varchar(100) NOT NULL,
    fk_turma int NOT NULL,
    FOREIGN KEY (fk_turma ) REFERENCES turma(pk_turma )
);
```

Adicionar chave estrangeira

Atenção: ao criar uma chave estrangeira é importante prestar atenção na ordem de criação das tabelas.

Se a tabela tem uma dependência de outra tabela (Chave estrangeira), a tabela referenciada deve ser criada primeiro.

DELETE ON CASCADE

A cláusula "**DELETE CASCADE**" é usada em um banco de dados relacionado para automatizar a exclusão de registros dependentes em tabelas relacionadas quando um registro pai é excluído. Essa cláusula garante a consistência dos dados, removendo automaticamente registros relacionados em cascata.

```
CREATE TABLE aluno(
 pk_aluno int PRIMARY KEY AUTO_INCREMENT NOT NULL,
 nome_aluno varchar(100) NOT NULL,
 fk_turma int NOT NULL,
 FOREIGN KEY (fk_turma ) REFERENCES turma(pk_turma )
 ON DELETE CASCADE
 );
```

Excluir Tabela ou Banco de Dados

```
DROP TABLE aluno;
DROP DATABASE escola;
```

Alterar Tabela

Excluir Coluna:

```
ALTER TABLE aluno
DROP COLUMN  email_aluno;
```

Adicionar Coluna:

```
ALTER TABLE aluno
ADD (telefone_aluno VARCHAR(100));
```

Alterar Tabela

Renomear coluna:

```
ALTER TABLE aluno
CHANGE cpf_aluno CPF_aluno  VARCHAR(100);
```

Alterar Tabela

Alterar o tipo da coluna:

```
ALTER TABLE aluno
MODIFY COLUMN cpf_aluno  VARCHAR(100);
```

Alterar Tabela

Excluir coluna:

```
ALTER TABLE aluno
DROP COLUMN cpf_aluno;
```

Excluir elementos da tabela

```
TRUNCATE aluno;
```

Diferença entre DROP e TRUNCATE:

- TRUNCATE:
  - exclui registros.
- DROP:
  - Exclui toda estrutura da tabela.

Comentário

Única linha:

```
-- DROP TABLE aluno;
```

Múltiplas linhas:

```
/* CREATE TABLE aluno(
   nome VARCHAR(100)
)
*/
```

Renomear

- Renomear tabela:

```
ALTER TABLE aluno
RENAME TO estudante;
```

Alterar o auto incremento

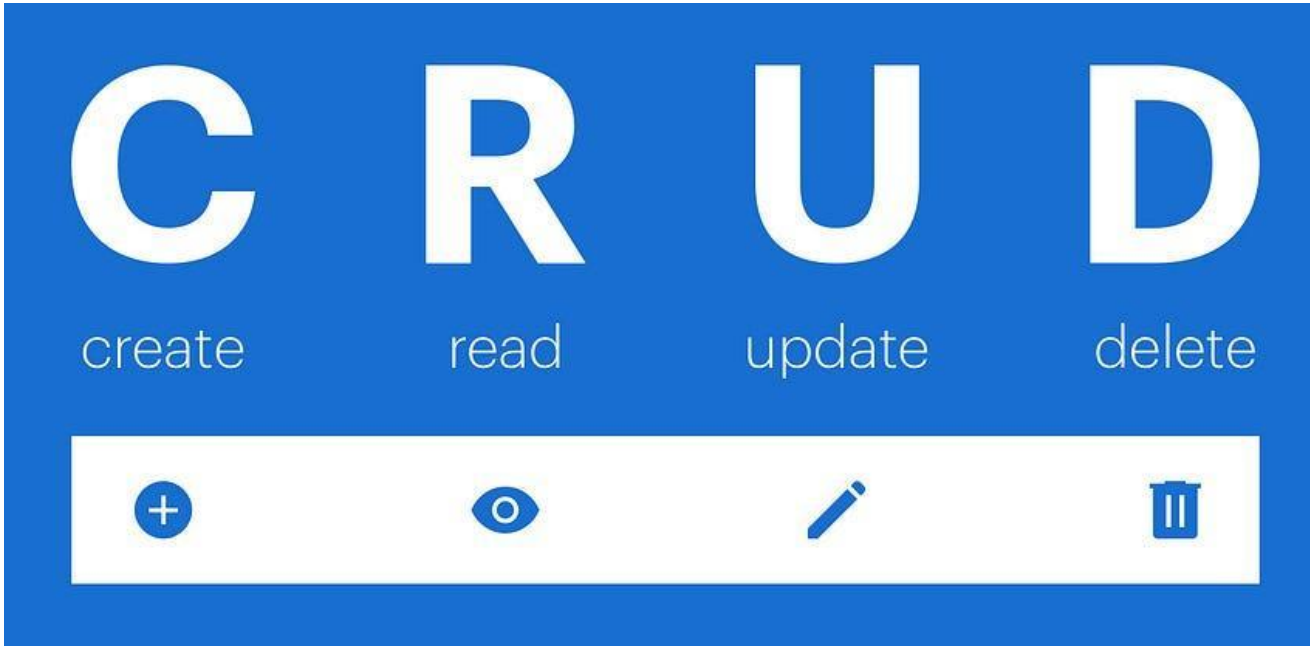
- O auto incremento permite que um identificador único seja gerado quando um novo registro é inserido em uma tabela de forma automática.

- Redefinir auto incremento:

```
ALTER TABLE aluno
AUTO_INCREMENT = 1;
```

DML - *Data Manipulation Language*

DML - *Data Manipulation Language*



DML - *Data Manipulation Language*

INSERT

UPDATE

DELETE

INSERIR REGISTROS

```
INSERT INTO nome_tabela
VALUES(“valor”,”valor”)
```

```
INSERT INTO jogador
(coluna1,coluna2)
VALUES(“Boby”,18)
```

INSERIR REGISTROS - inserir data

```
INSERT INTO aluno (nome_aluno,data_nascimento_aluno, idade_aluno,
horario_entrada_aluno, mensalidade_aluno, eh_bolsista_aluno)

VALUES
('Lucy', 'lucy@outlook.com', STR_TO_DATE('01/10/2005', '%d/%m/%Y'),
18, STR_TO_DATE('07:00','%H:%i'), 1129.90, 0),
```

INSERIR REGISTROS - formatação

- DATE: data
    - STR_TO_DATE('16/10/2023', '%d/%m/%Y')
- TIME: horário
    - STR_TO_DATE('07:35', '%h:%i'),
- DATETIME: data e horário
    - STR_TO_DATE('16/10/2023 07:36', '%d/%m/%Y %H:%i'),
- YEAR: ano

INSERIR REGISTROS - múltiplos registros

```
INSERT INTO aluno (nome_aluno,data_nascimento_aluno, idade_aluno)
VALUES
  ('Lucy', 'lucy@outlook.com', 18),
  ('John', 'john@gmail.com', 25),
  ('Maria', 'maria@yahoo.com', 23);
```

ATUALIZAR REGISTRO

```
UPDATE nome_tabela
SET nome_campo1 = “valor”
```

```
UPDATE jogador
SET nome = “Ane”
```

INCORRETO! Alteração de todos os registros

ATUALIZAR REGISTRO

```
UPDATE nome_tabela
SET nome_campo2 = “valor”
WHERE nome_campo1 = “valor”

UPDATE jogador
SET nome = “Ane”
WHERE id=1
```

EXCLUIR REGISTRO

```
DELETE FROM nome_tabela
WHERE nome_campo2 = “valor”
```

```
DELETE FROM jogador
WHERE id = 1
```

DQL - *Data Query Language*

DQL - *Data Query Language*

SELECT

WHERE

CONSULTAR REGISTRO

```
SELECT * FROM nome_tabela
```

```
SELECT nome_campo1,nome_campo2
FROM nome_tabela
```

CONSULTAR REGISTRO

```
SELECT nome,idade
FROM jogador
```

Jogador	
Nome	Idade
Boby	18
Andy	20

CONSULTAR REGISTRO

```
SELECT id,nome
FROM jogador
WHERE idade > 18
```

	Jogador
ID	Nome
2	Andy

Operadores

Operadores lógicos:

- AND:
- OR
- NOT

```
SELECT nome FROM jogador

WHERE idade >= 16 AND idade = < 20;
```

Operadores de comparação:

- !=
- >=
- <=
- >
- <
- =, <>

```
SELECT nome FROM jogador

WHERE id_jogo= 2 OR id_jogo= 3;
```

FUNÇÕES

Funções

- COUNT(): Retorna o total de linhas.
- AVG(): Retorna a média de uma coluna.
- MIN(): Retorna o registro de menor valor de uma coluna.
- MAX(): Retorna o registro de maior.

- Quantos jogadores são cadastrados?

```
SELECT COUNT(id) FROM jogador;
```

- Qual a média de idade dos jogadores?

```
SELECT AVG (idade) FROM jogador;
```

- Qual idade máxima dos jogadores?

```
SELECT MAX(idade) FROM jogador;
```

- Qual idade mínima dos jogadores?

```
SELECT MIN(idade) FROM jogador;
```

Funções

ORDER BY(): Ordena os registros.

- Jogadores ordenados pela idade em ordem crescente

```
SELECT nome FROM jogador
ORDER BY idade;
```

- Jogadores ordenados pela idade em ordem decrescente

```
SELECT nome FROM jogador
ORDER BY idade DESC;
```

Funções

GROUP BY(): agrupa os registros.

- Jogadores agrupados pelo id do jogo (assim todos os que tiverem o mesmo id de jogo aparecem na mesma linha)

```
SELECT nome FROM jogador GROUP BY id_jogo
```

- Podemos usar função COUNT() para contar quantos jogadores compraram cada jogo.

```
SELECT COUNT(id_jogo)FROM jogador GROUP BY id_jogo
```

Funções

STR_TO_DATE(): Esta função permite formatar uma string em formatos como **datetime, date** ou **time**.

```
INSERT INTO (data_jogador_jogo) VALUES
(STR_TO_DATE('10/05/2020','%d/%m/%Y'));

INSERT INTO (data_hora_jogador_jogo) VALUES
(STR_TO_DATE('10/05/2020 10:35','%d/%m/%Y
%h:%i'));

INSERT INTO (hora_jogador_jogo) VALUES
(STR_TO_DATE('10:35','%h:%i'));
```

Funções

- FLOOR(): arredonda para baixo.
- CEIL(): arredonda para cima.
- TRUNCATE(): trunca o valor considerando apenas a quantidade de casas informadas.
- ROUND():arredonda considerando o número de casas informadas.

```
SELECT FLOOR(AVG(idade_jogador)) FROM JOGADOR;

SELECT CEIL(AVG(idade_jogador)) FROM JOGADOR;

SELECT TRUNCATE(AVG(idade_jogador),1) FROM JOGADOR;

SELECT ROUND(AVG(idade_jogador),1) FROM JOGADOR;
```

NOVO EXEMPLO

pk_turma	nome_turma	ano_turma
1	DESI V1	2022
2	DESI N1	2023
3	DESI M2	2021
4	DESI M1	2020

DDL tabela turma

```
CREATE TABLE turma (
  pk_turma int NOT NULL,
  nome_turma varchar(100) NOT
NULL,
  ano_turma int NOT NULL
);
```

NOVO EXEMPLO

pk_aluno	nome_aluno	sobrenome_aluno	cpf_aluno	telefone_aluno	fk_turma	data_nascimento_aluno	idade_aluno	ano_ingresso_aluno	ano_egresso_aluno
1	João	Silva	12345678910	4799999999	1	01/01/1990	33	2010	2014
2	Maria	Santos	98765432110	4788888888	2	15/03/1992	31	2011	2016
3	Pedro	Lima	45678912310	4777777777	1	20/07/1991	32	2009	2013
4	Ana	Oliveira	65432198710	4766666666	3	05/12/1993	28	2012	2017
5	Carlos	Martins	78912345610	4755555555	2	18/11/1990	33	2010	2014
6	Mariana	Pereira	32165498710	4744444444	1	10/02/1992	31	2011	2016
7	Fernando	Gomes	98712345610	4733333333	3	25/06/1994	27	2013	2018
8	Amanda	Rodrigues	45678932110	4722222222	2	08/09/1991	32	2009	2013
9	Lucas	Costa	78932165410	4711111111	1	03/04/1993	29	2012	2017
10	Juliana	Almeida	65498732110	4700000000	3	12/08/1990	33	2010	2014
11	Marcos	Rocha	98732165410	4799990000	2	06/05/1992	31	2011	2016
12	Larissa	Ferreira	32198765410	4788881111	1	28/10/1991	32	2009	2013
13	Rafael	Carvalho	78965432110	4777772222	3	14/07/1993	28	2012	2017
14	Camila	Mendes	45612378910	4766663333	2	17/12/1990	33	2010	2014
15	Gustavo	Sousa	78945612310	4755554444	1	23/01/1992	31	2011	2016
16	Laura	Pinto	32178945610	4744445555	3	06/08/1994	27	2013	2018
17	Henrique	Barros	98745678910	4733336666	2	30/11/1991	32	2009	2013
18	Isabela	Fernandes	45678998710	4722227777	1	15/06/1993	29	2012	2017
19	Roberto	Cavalcanti	78998745610	4711118888	3	09/09/1990	33	2010	2014
20	Luana	Gonçalves	32145678910	4700009999	2	24/04/1992	31	2011	2016
23	Luiza	Pereira	12345678911	4134343434	NULL	10/07/2020	22	2020	2023

DDL tabela estudante

```
CREATE TABLE estudante (
  pk_aluno int NOT NULL PRIMARY KEY AUTO_INCREMENT ,
  nome_aluno varchar(100) NOT NULL,
  sobrenome_aluno varchar(100) DEFAULT NULL,
  cpf_aluno varchar(11) DEFAULT NULL,
  telefone_aluno varchar(11) DEFAULT NULL,
  fk_turma int DEFAULT NULL,
  data_nascimento_aluno date DEFAULT NULL,
  idade_aluno int NOT NULL,
  ano_ingresso_aluno int NOT NULL,
  ano_egresso_aluno int NOT NULL
  FOREIGN KEY fk_turma REFERENCES turma (pk_turma)
)
```

GROUP BY:

1. Seleciona estudantes da turma 2.

1. Seleciona estudantes da turma 2 ou da turma 3 e os agrupa por turma.

1. Seleciona estudantes da turma 2 ou turma 3 e os agrupa por turma e conta quando estudante tem por turma.

```
SELECT * FROM estudante WHERE fk_turma=2;
```

```
SELECT * FROM estudante WHERE fk_turma=2 OR fk_turma=3 GROUP BY fk_turma;
```

```
SELECT count(pk_aluno) FROM estudante WHERE fk_turma=2 OR fk_turma=3 GROUP BY fk_turma;
```

INNER JOIN
JOIN/LEFT
JOIN/RIGHT

JOIN/ INNER JOIN

Junção que combina registros de duas ou mais tabelas

```
SELECT * FROM estudante
JOIN turma;
```

```
SELECT * FROM estudante
INNER JOIN turma;
```

INNER JOIN condição: ON

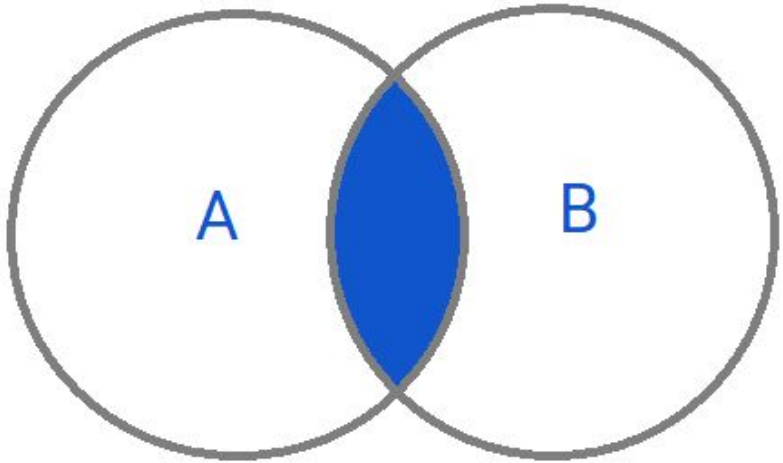
Retorna apenas os registros que têm correspondência nas duas tabelas envolvidas na junção.

Combina as linhas com base em uma condição de correspondência especificada na cláusula **ON**.

Se não houver correspondência, as linhas não serão incluídas no resultado.

Não traz valores nulos.

```
SELECT * FROM estudante
INNER JOIN turma
ON estudante.fk_turma =
turma.pk_turma
```

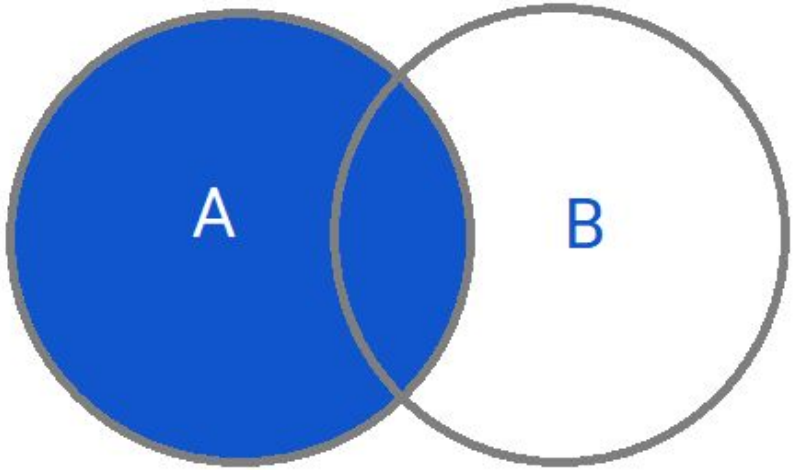


LEFT JOIN

Retorna todos os registros da tabela à esquerda da junção (tabela esquerda) e os registros correspondentes da tabela à direita da junção (tabela direita).

Se não houver correspondência na tabela direita, serão retornados valores NULL.

```
SELECT * FROM estudante
LEFT JOIN turma
ON estudante.fk_turma =
turma.pk_turma
```

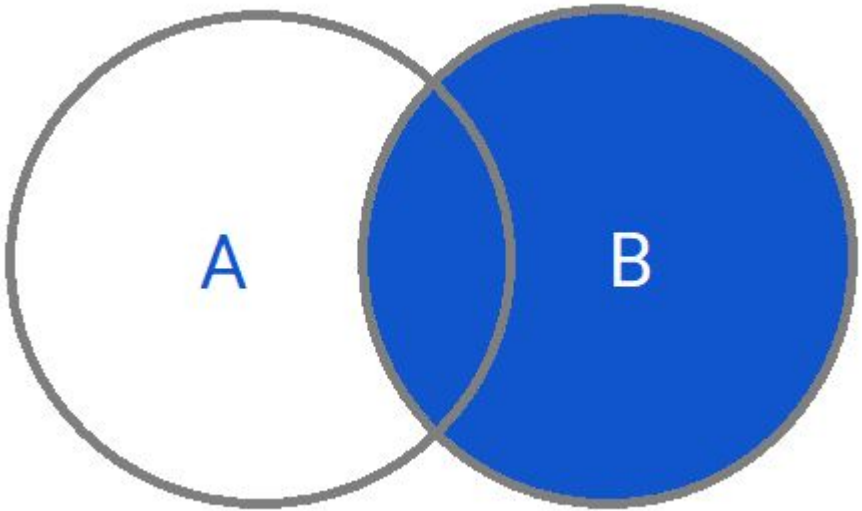


RIGHT JOIN

Retorna todos os registros da tabela à direita da junção (tabela direita) e os registros correspondentes da tabela à esquerda da junção (tabela esquerda).

Se não houver correspondência na tabela esquerda, serão retornados valores NULL.

```
SELECT * FROM estudante
RIGHT JOIN turma
ON estudante.fk_turma =
turma.pk_turma
```



UNION

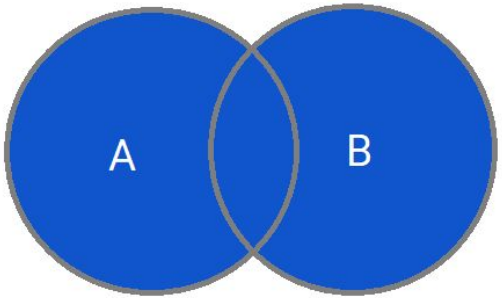
UNION

Combina os resultados de duas consultas diferentes realizadas nas tabelas.

```
SELECT E.nome_aluno, T.nome_turma
FROM estudante AS E
LEFT JOIN turma AS T ON E.fk_turma =
T.pk_turma

UNION

SELECT E.nome_aluno, T.nome_turma
FROM estudante AS E
RIGHT JOIN turma AS T ON E.fk_turma =
T.pk_turma;
```



CONSULTAS ANINHADAS

Uma consulta **aninhada**, também conhecida como subconsulta (**subselect**), é uma consulta SQL que é incorporada dentro de outra consulta SQL. Ela permite usar o resultado de uma consulta interna como parte da condição ou da seleção na consulta externa.

Em uma consulta aninhada, a consulta interna é executada primeiro e seu resultado é usado na consulta externa. A consulta interna é encapsulada dentro da cláusula FROM, WHERE, ou SELECT da consulta externa.

```
SELECT nome
FROM estudante
WHERE pk_aluno IN (SELECT
pk_aluno  FROM estudante
WHERE idade = 32);
```

VIEW

VIEW:

Uma view (visão) em um banco de dados é uma representação virtual de uma tabela ou de uma combinação de tabelas. Ela é criada a partir de uma consulta SQL e pode ser usada como uma tabela normal para consulta, mas não armazena dados fisicamente.

```
CREATE VIEW
viewAlunosEntre30e32 AS

SELECT E.nome_aluno,
E.idade_aluno,
T.nome_turma FROM
estudante AS E
JOIN turma AS T
ON E.fk_turma = T.pk_turma
WHERE E.idade_aluno >= 30
AND E.idade_aluno <= 32
ORDER BY E.nome_aluno;
```

Por que view?

- Reutilização: As views são objetos permanentes, o que é altamente vantajoso do ponto de vista produtivo, já que podem ser acessadas simultaneamente por vários usuários.

- Segurança: As views possibilitam ocultar colunas específicas de uma tabela. Ao criar uma view com as colunas que desejamos exibir, podemos disponibilizá-la aos usuários, fornecendo maior controle sobre a exposição de dados sensíveis.

Por que view?

- Simplificação do código: As views nos permitem criar um código de programação mais conciso, pois podem conter uma consulta SELECT complexa. Assim, ao criar views para os programadores, poupamos seu esforço de criar consultas SELECT, resultando em maior produtividade para a equipe de desenvolvimento.

- Desempenho: Em determinados casos, as visualizações podem aprimorar o desempenho do sistema. Por exemplo, é possível criar uma visualização que armazene o resultado de uma consulta complexa e atualizá-la periodicamente em segundo plano. Dessa forma, evitamos executar a consulta a cada acesso aos dados, o que contribui para a eficiência geral do sistema.

TRIGGERS

Trigger:

Uma trigger (gatilho) em um banco de dados é um objeto associado a uma tabela que é acionado automaticamente quando ocorre um evento específico, como inserção, atualização ou exclusão de dados nessa tabela. Ela consiste em um conjunto de instruções SQL que são executadas em resposta ao evento acionador.

```
CREATE TRIGGER
trg_UpdateLastModified
BEFORE UPDATE ON clientes
FOR EACH ROW
BEGIN
    SET
NEW.ultima_atualizacao =
CURRENT_TIMESTAMP;
END
```

STORED PROCEDURE

Stored Procedure:

Uma ***stored procedure***, é um objeto do banco de dados que contém um conjunto de instruções SQL predefinidas. Ela é armazenada no banco de dados e pode ser chamada e executada posteriormente.

```
CREATE PROCEDURE
getAnimalsByType (
    IN p_tipo VARCHAR(50)
)
BEGIN
    SELECT
        id,
        nome,
        tipo,
        idade,
        peso
    FROM animal
    WHERE tipo = p_tipo
    ORDER BY nome;
END
```

DCL - *Data Control Language*

DQL - *Data Control Language*

CREATE USER
DROP USER
CREATE ROLE
DROP ROLE
ALTER USER
ALTER ROLE
GRANT
REVOKE

Controle os privilégios e permissões de acesso em um banco de dados.

1. Criar usuário.
2. Excluir usuário.
3. Cria papel.
4. Excluir papel.
5. Adicionar permissões.
6. Remover permissões.
7. Atribuir permissão geral.

```
CREATE USER fulano;
CREATE USER ciclano
IDENTIFIED BY “1234”;

CREATE ROLE adminstrador;
CREATE ROLE usuario;

GRANT SELECT, INSERT, UPDATE
ON turma TO administrador;

REVOKE SELECT, INSERT, UPDATE
ON estudante FROM usuario;

GRANT ALL PRIVILEGES ON
clinica_veterinaria.* TO
administrador;
```

DCL - *Data Transaction Language*

Usados para garantir a consistência dos dados em um banco de dados, permitindo que as alterações sejam tratadas de forma controlada e revertidas se necessário.

- `START TRANSACTION:`
  Inicia uma sequência de comandos.
- `COMMIT:`
  Confirma as alterações.
- `ROLLBACK:`
  Caso ocorra algum erro, reverta as alterações parciais.

```
START TRANSACTION; -- Inicia a transação

-- Atualiza o saldo da conta do cliente 1
UPDATE clientes SET saldo = saldo - 100
WHERE id = 1;

-- Atualiza o saldo da conta do cliente 2
UPDATE clientes SET saldo = saldo + 100
WHERE id = 2;

COMMIT; -- Confirma a transação

-- Se ocorrer algum erro ou problema, podemos executar ROLLBACK para desfazer as alterações:

-- ROLLBACK; -- Desfaz a transação e restaura o estado anterior
```

REFERÊNCIAS:

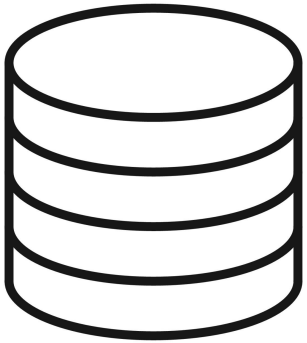
DATE, Christopher J. **Introdução a sistemas de bancos de dados**. Elsevier Brasil, 2004.

MARIADB. *Documentação MariaDB*. MariaDB Knowledge Base, 2026. Disponível em: https://mariadb.com/kb/pt-br/documentacao-mariadb/. Acesso em: 04 fev. 2026

Esses slides estão protegidos por uma licença Creative Commons.

https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode

SQL *(Standard Query Language)*



Banco de Dados

marisangila.alves@catolicasc.org.br

Marisangila Alves, MSc