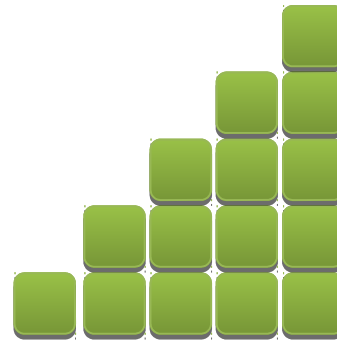


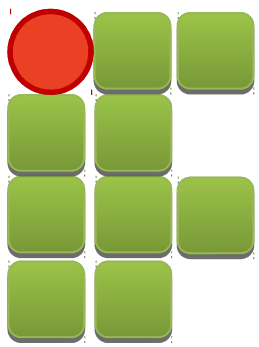
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA



Banco de Dados

Prof. Eliomar Campos

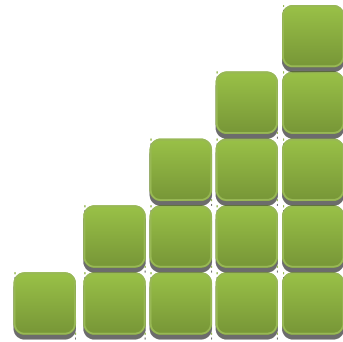




INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA

DDL, DQL, DML, DCL, DTL

Categorias das Cláusulas da Linguagem SQL

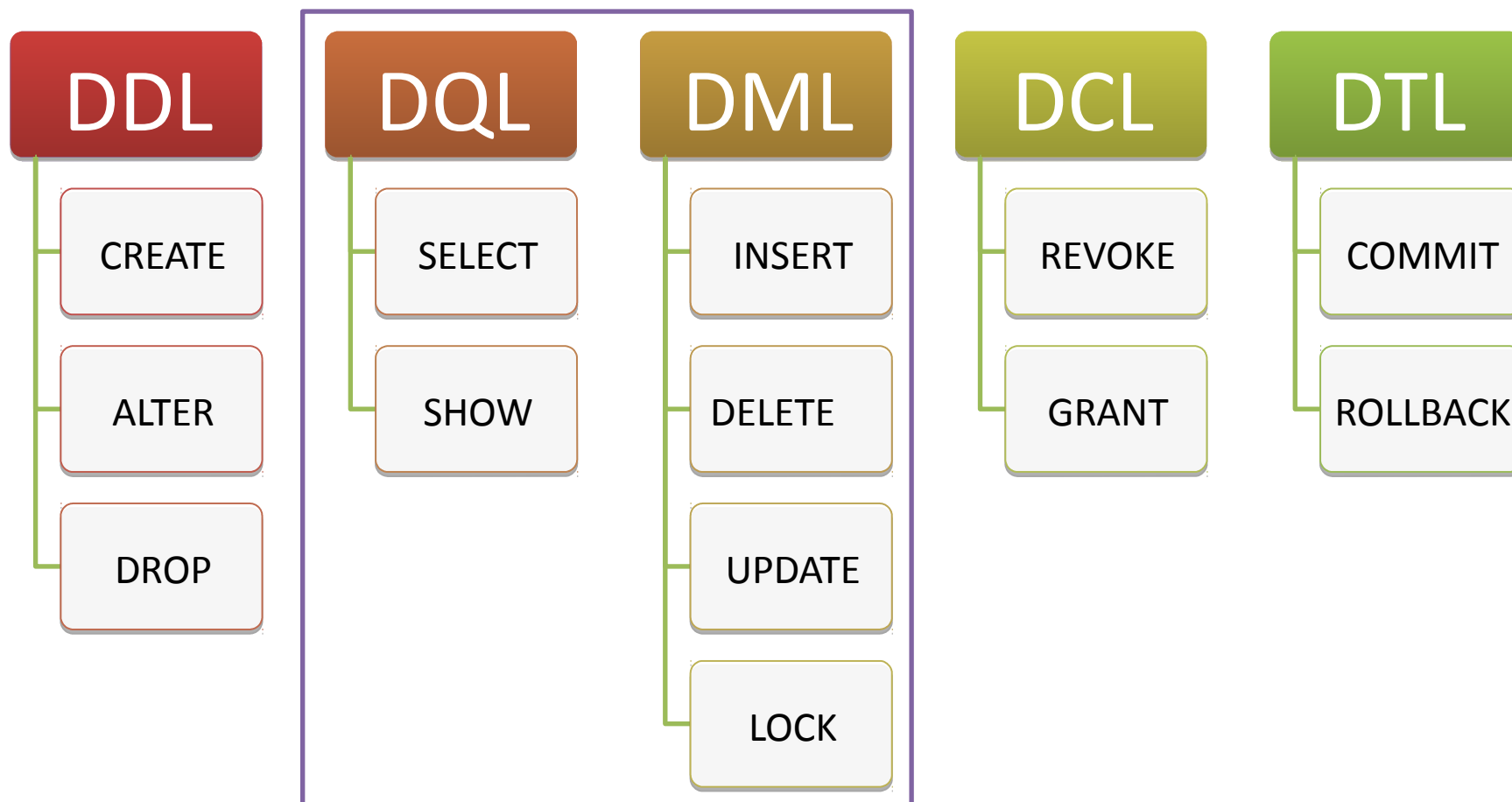


DDL, DQL, DML, DCL, DTL ?

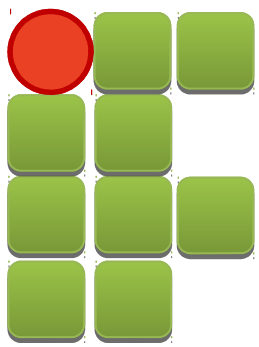
- **DDL** - Data Definition Language
- **DQL** - Data Query Language
- **DML** - Data Manipulation Language
- **DCL** - Data Control Language
- **DTL** - Data Transaction Language

***Obs.:** Estas NÃO são linguagens diferentes, mas apenas uma organização em categorias dos comandos da mesma linguagem. Serve tanto para fins didáticos, quanto requisitos necessários de um SGBD de "verdade".*

Principais Cláusulas das Categorias



Muitos autores consideram as duas apenas como DML



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA

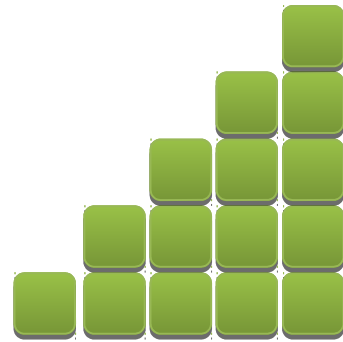
DDL (Data Definition Language)

CREATE

ALTER

DROP

Definindo os Dados



DDL

- **Linguagem de Definição de Dados** - usada para definir estruturas de dados:
- **CREATE DATABASE:** Criar uma base de dados;
- **DROP DATABASE:** Apagar uma base de dados;
- **CREATE TABLE:** Criar uma tabela;
- **DROP TABLE:** Apagar uma tabela;
- **ALTER TABLE:** Alterar estrutura de uma tabela;

DDL: Criando uma Base de Dados

```
CREATE DATABASE biblioteca_db;
```

OU

```
CREATE SCHEMA biblioteca_db;
```

- **Parâmetros adicionais:** Criando uma base de dados **somente se não existir** outra com o mesmo nome.

```
CREATE DATABASE IF NOT EXISTS biblioteca_db;
```

OU

```
CREATE SCHEMA IF NOT EXISTS biblioteca_db;
```

DDL: Apagando uma Base de Dados

```
DROP SCHEMA biblioteca_db;
```

OU

```
DROP DATABASE biblioteca_db;
```

- **Parâmetros adicionais:** Apagando uma base de dados **somente se existir** uma com o respectivo nome.

```
DROP SCHEMA IF EXISTS biblioteca_db;
```

OU

```
DROP DATABASE IF EXISTS biblioteca_db;
```


Utilizando uma Base de Dados

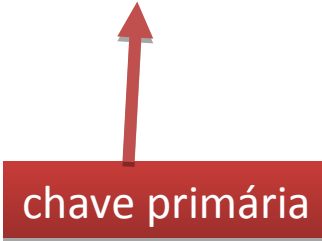
```
USE biblioteca_db;
```

OBS.: A cláusula **USE** não seria considerada como DDL mas poderia se encaixar em DML.

DDL: CREATE TABLE

- Criando uma tabela, opção 1:

```
1 • CREATE TABLE cliente(  
2     clie_id int not null auto_increment primary key,  
3     clie_nome varchar(50) not null,  
4     clie_sexo char(1) null,  
5     clie_nascimento date null,  
6     clie_telefone varchar(50) null,  
7     clie_email varchar(50) null,  
8     clie_cidade varchar(50) null,  
9     clie_estado varchar(50) null,  
10    clie_endereco varchar(100) null,  
11    clie_ativo boolean null  
12 );
```



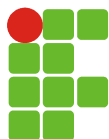
chave primária

DDL: CREATE TABLE

- Criando uma tabela, opção 2:
- A única diferença está em definir a chave primária no final do bloco:

```
1 • CREATE TABLE cliente(  
2     clie_id int not null auto_increment,  
3     clie_nome varchar(50) not null,  
4     clie_sexo char(1) null,  
5     clie_nascimento date null,  
6     clie_telefone varchar(50) null,  
7     clie_email varchar(50) null,  
8     clie_cidade varchar(50) null,  
9     clie_estado varchar(50) null,  
10    clie_endereco varchar(100) null,  
11    clie_ativo boolean null,  
12    primary key (clie_id)  
13 );
```

Outra opção para chave primária



DDL: DROP TABLE

- Apagando uma tabela:

```
DROP TABLE cliente;
```

DDL: ALTER TABLE RENAME

- Renomeando o nome da tabela:

```
ALTER TABLE cliente RENAME clientes;
```

DDL: ALTER TABLE MODIFY

- Modificando somente o tipo de um campo da tabela:

```
ALTER TABLE cliente MODIFY clie_sexo varchar(50)
```

OU

```
ALTER TABLE aluno MODIFY COLUMN alun_cpf varchar(50)
```

OU, para mudar a ordem da coluna:

```
ALTER TABLE cliente MODIFY clie_cpf varchar(50) AFTER clie_estado
```

DDL: ALTER TABLE CHANGE

- Alterando o nome de um campo da tabela:

```
ALTER TABLE cliente CHANGE clie_nascimento clie_nasc DATE NULL;
```

OU

```
ALTER TABLE cliente CHANGE COLUMN clie_nascimento clie_nasc DATE NULL;
```

OU, para mudar a ordem da coluna:

```
ALTER TABLE cliente CHANGE clie_email clie_email varchar(50) null AFTER clie_sexo;
```

DDL: ALTER TABLE ADD

- Adicionando um campo na tabela:

```
ALTER TABLE cliente ADD clie_cpf varchar(11) null;
```

OU

```
ALTER TABLE cliente ADD clie_cpf varchar(11) not null;
```

OU, especificando a ordem da coluna:

```
ALTER TABLE cliente ADD clie_cpf varchar(11) null AFTER clie_nome;
```


DDL: ALTER TABLE DROP

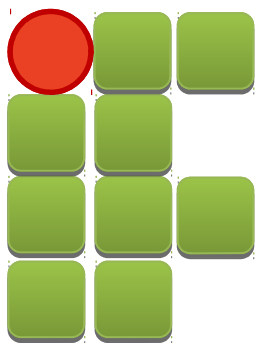
- Apagando um campo da tabela:

```
ALTER TABLE cliente DROP clie_cpf;
```

DDL: ALTER TABLE

- Resumo:

```
1 • ALTER TABLE cliente RENAME aluno;
2 • ALTER TABLE aluno ADD alun_cpf varchar(50);
3 • ALTER TABLE aluno MODIFY alun_cpf varchar(11);
4 • ALTER TABLE aluno CHANGE alun_cpf alun_cnpj varchar(11);
5 • ALTER TABLE aluno DROP alun_cnpj;
```



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA

DML (Data Manipulation Language)

INSERT

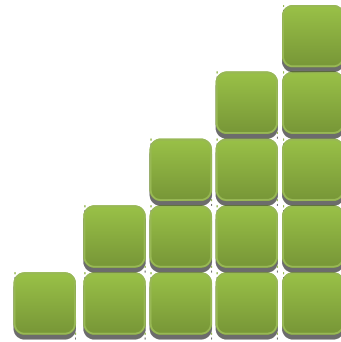
DELETE

TRUNCATE

UPDATE

LOCK

Manipulando Registros



DML: INSERT

- Inserindo apenas um registro:
- Especificando as colunas:

```
1 • INSERT INTO cliente (clie_nome, clie_cidade, clie_email)  
2   VALUES ('Eliomar Campos', 'Delmiro', 'eliomar@gmail.com');
```

OBS.: Se for inserir em todas as colunas, não precisa especificar as colunas, mas apenas passar os respectivos valores. As colunas que não receberam valor, por padrão recebem **NULL**.

DML: INSERT

- Inserindo apenas um registro:
- NÃO especificando as colunas:

```
1 • INSERT INTO cliente VALUES (0, 'Eliomar Campos', 'M', '1987-12-01',  
2                               '+55(075)99100-2261', 'eliomar@gmail.com',  
3                               'Delmiro Gouveia', 'Alagoas',  
4                               'Rua Sao Judas Tadeu', 1);
```

OBS.: Sem especificar as colunas, é obrigatório fornecer os valores de todas as colunas. Dessa forma, é necessário atribuir um **ID zero** para o auto_increment funcionar. Os valores serão inseridos nas respectivas colunas de acordo com a ordem da tabela.

DML: INSERT

- Inserindo mais de um registro:
- NÃO especificando as colunas:

```
1 • INSERT INTO cliente VALUES (0, 'Eliomar Campos', 'M', '1987-12-01',  
2                               '+55(075)99100-2261', 'eliomar@gmail.com',  
3                               'Delmiro Gouveia', 'Alagoas',  
4                               'Rua Sao Judas Tadeu', 1),  
5 (0, 'Rafaela Wanderley', 'F', '1992-09-17',  
6   '+55(075)98800-2267', 'rafaela@gmail.com',  
7   'Paulo Afonso', 'Bahia',  
8   'Rua Barcelona', 0),  
9 (0, 'Maria José', 'F', '1980-11-11',  
10 NULL, NULL,  
11 '', '',  
12 'Rua Nossa Senhora', 1);
```

OBS.: Você pode passar **NULL** ou **' '** para valores vazios. Observe que para cada registro inserido abre e fecha parênteses seguido da vírgula, e ao final ponto e vírgula.

DML: DELETE

- Deletando um registro:

```
DELETE FROM cliente WHERE clie_id = 1;
```

OBS.: Serão apagados todos os registros que satisfazem a condição WHERE. Mas nesse caso, apenas um será apagado.

DML: TRUNCATE

- Deletando TODOS os registros:

```
TRUNCATE TABLE cliente;
```

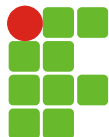
OU

```
TRUNCATE cliente;
```

OU

```
DELETE FROM cliente;
```

OBS.: Para o **DELETE FROM cliente** excluir tudo, o *safe mode* (modo seguro) precisa estar desabilitado no MySQL Workbench. Por padrão a ferramenta habilita o *safe mode*. Já o **TRUNCATE** funciona com o *safe mode* habilitado ou não.



DML: UPDATE

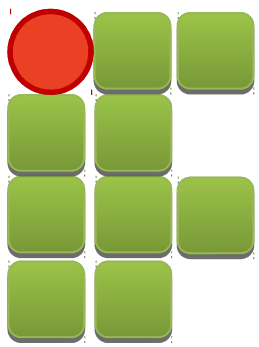
- Atualizando um registro:

```
UPDATE cliente SET clie_nome="Eliomar" WHERE clie_id = 1;
```

OU

```
1 • UPDATE cliente SET clie_nome="Eliomar", clie_email="eliomar@gmail",  
2                       clie_cidade="Paulo Afonso"  
3                       WHERE clie_id = 1;
```

OBS.: Serão atualizados todos os registros que satisfazem a condição WHERE.
Mas nesse caso, apenas um será atualizado.



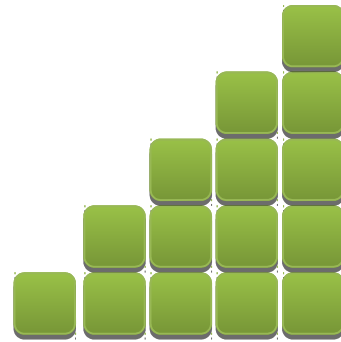
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA

DQL (Data Query Language)

SELECT

SHOW

Consultando Registros



DQL: SHOW

- Listando as bases de dados:

`SHOW DATABASES;` **ou** `SHOW SCHEMAS;`

- Listando as tabelas da base em uso:

`SHOW TABLES;`

- Descrevendo os detalhes de uma tabela:

`DESCRIBE cliente;` **ou** `DESC cliente;`

DQL: SELECT

- Consultando todos os registros com todos os campos:

```
SELECT * FROM cliente;
```

- Consultando todos os registros com campos específicos:

```
SELECT clie_id, clie_nome FROM cliente;
```

- Atribuindo **alias** ou **aliases** (pseudônimos ou apelidos) aos campos:

```
SELECT clie_id AS Código, clie_nome AS Nome FROM cliente;
```

OU sem o AS

```
SELECT clie_id Código, clie_nome Nome FROM cliente;
```

DQL: SELECT WHERE

- **Operadores relacionais (=, <, >, >=, <=, <>, !=):**
- Consultando com determinadas condições – O ID tem que ser igual a 6:

```
SELECT * FROM cliente WHERE clie_id = 6;
```

- Condição – O ID tem que ser diferente de 6:

```
SELECT * FROM cliente WHERE clie_id <> 6;
```

OU

```
SELECT * FROM cliente WHERE clie_id != 6;
```

DQL: SELECT WHERE

- **AND, OR, BETWEEN:**

- Condição – O ID tem que ser maior do que 6 e menor ou igual a 10:

```
SELECT * FROM cliente WHERE clie_id > 6 AND clie_id <= 10;
```

OU, é melhor usar o BETWEEN

```
SELECT * FROM cliente WHERE clie_id BETWEEN 7 AND 10;
```

- Condição – O ID tem que ser maior do que 8 ou igual a 6:

```
SELECT * FROM cliente WHERE clie_id > 8 OR clie_id = 6;
```

- **IN** e **NOT IN**

- Use para fazer comparações com N valores de uma única vez.

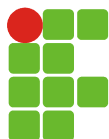
- Condição – Clientes que o ID é igual a 12 ou 6 ou 10 ou 9:

```
SELECT * FROM cliente WHERE clie_id IN (12,6,10,9);
```

- Condição – Clientes que o ID é diferente de 12 e 6 e 10 e 9:

```
SELECT * FROM cliente WHERE clie_id NOT IN (12,6,10,9);
```

OBS.: Sem o **IN** ou **NOT IN**, teríamos que fazer uso de vários **=**, **<>**, **AND**, **OR** para cada condição



DQL: SELECT WHERE

- LIKE:

- Condição – Clientes que o nome começa com a letra E:

```
SELECT * FROM cliente WHERE clie_nome LIKE 'e%';
```

- Condição – Clientes que o nome termina com S:

```
SELECT * FROM cliente WHERE clie_nome LIKE '%s';
```


DQL: SELECT WHERE

- LIKE:

- Condição – Clientes que o nome contém a letra M em qualquer posição:

```
SELECT * FROM cliente WHERE clie_nome LIKE '%m%';
```

OBS.: O símbolo de % (porcento) é um curinga no SQL. Quando não sabemos uma parte da string, podemos substituir essa parte não conhecida utilizando o % no início, no meio ou no fim dela.