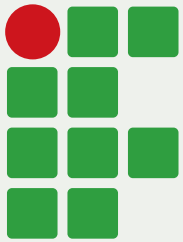
# Linguagem SQL

BCD29008 – Engenharia de Telecomunicações

Prof.  Emerson Ribeiro de Mello

mello@ifsc.edu.br

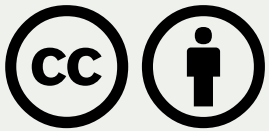
05 de maio de 2022



INSTITUTO FEDERAL
Santa Catarina

Câmpus
São José

# Licenciamento



# Revisão: modelo relacional – restrição de integridade

- **Integridade de domínio**
  - Valor de um campo deve respeitar a definição de valores permitidos
- **Integridade de vazio**
  - Indica se os valores de uma coluna podem ou não ser vazios (NULL)
- **Integridade de chave**
  - Valores das chaves primárias devem ser únicos
- **Integridade referencial**
  - Valores que aparecem uma chave estrangeira devem aparecer na chave primária da tabela referenciada

# *Structured Query Language* – SQL

- **Linguagem de Consulta Estruturada**, baseada em álgebra relacional e permite a definição e controle de acesso aos dados; manipulação de dados
- Desenvolvida dentro da IBM na década de 70, sendo hoje a mais usada em SGBD relacional
- **Existem padrões ANSI e ISO, porém um código SQL geralmente não é portável** e pequenos ajustes são necessários para cada SGBD
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008, SQL:2011

# *Structured Query Language* – SQL

- **Linguagem de Consulta Estruturada**, baseada em álgebra relacional e permite a definição e controle de acesso aos dados; manipulação de dados
- Desenvolvida dentro da IBM na década de 70, sendo hoje a mais usada em SGBD relacional
- **Existem padrões ANSI e ISO, porém um código SQL geralmente não é portável** e pequenos ajustes são necessários para cada SGBD
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008, SQL:2011

Os códigos apresentados nessa aula foram escritos para o MySQL

# Linguagem SQL consiste de

- **Data Definition Language – DDL**
  - Especificação do esquema relacional, indicando restrições
  - Cria, altera, exclui tabelas
- **Data Manipulation Language – DML**
  - Consulta, insere, modifica e exclui tuplas das tabelas
- **Data Control Language – DCL**
  - Controle de acesso e manipulação sobre os dados
- **Data Transaction Language – DTL**
  - Para especificar início e término de uma transação

A linguagem não é sensível a caixa (alta ou baixa)

- Ex: CREATE, create

# SQLite

# SQLite

- Biblioteca que implementa um banco de dados relacional
  - Tamanho menor que 500Kb
  - Tabelas, índices, gatilhos, visões
  - Tabela com até 32mil colunas e número imilitado de linhas
  - Subconsultas e funções padrões do SQL
- Não possui processo servidor separado
- Banco de dados em um único arquivo
  - Arquivo pode ser usado sem problemas em arquiteturas de 32bit, 64bit, com ordenação de bytes big-endian ou little-endian
- Transações serializadas (garante ACID)



# Onde usar ou não usar o SQLite?

- **Adequado para**
  - Dispositivos embarcados e IoT
  - Formato de arquivo para aplicações (alternativa ao CSV, XML, etc)
  - Website com pouco ou médio tráfego (400k pedidos / dia)
  - Análise de dados
  - Ensino e treinamento
- **Não seria muito adequado para**
  - Aplicações cliente/servidor (acesso concorrente ao sistema de arquivos poderia gerar problemas)
  - Sites com grande volume de acesso
  - Grande conjunto de dados (SQLite está limitado a 140TB, mas o limite do sistema de arquivos pode ser mais restritivo)



# Onde usar ou não usar o SQLite?

## Use cliente/servidor

- A aplicação faz uso da rede para acessar os dados?
- Muita escrita concorrente?
- Big data?

## Cenário ideal para SQLite

- Sistema locais
- Baixa concorrência de escrita
- Limite de 1Tb

# Ferramentas

- Biblioteca SQLite no Linux

```
sudo apt install sqlite3

sqlite3 lab01.db
```

- DB Browser for SQLite[1]

```
sudo apt install sqlitebrowser
```

- Instalador para Windows no site oficial[2]

[1] https://sqlitebrowser.org/
[2] https://www.sqlite.org

# SQLite

## Sintaxe da linguagem

# Tipo de afinidade das colunas

Qualquer coluna (com exceção de `INTEGER PRIMARY KEY`) pode armazenar dados de qualquer classe (NULL, inteiro, texto, real, blob)

- Tipo das colunas[3]
  - **TEXT** – conversão de número para texto
  - **NUMERIC** – feita coerção de tipo (inteiro ou real) de forma que não se tenha perda de informação
  - **INTEGER** – coerção para inteiro
  - **REAL** – coerção para real
  - **BLOB** – armazena dados binários

[3] https://www.sqlite.org/datatype3.html

# Boolean e DATETIME

- Valores **booleanos** são armazenados como inteiros (0 ou 1)
- Funções de data e hora do SQLite[4] permite guardar esse tipo de informação como
  - **TEXT** – string (“YYYY-MM-DD HH:MM:SS.SSS”)
  - **REAL** – número de dias desde o meio-dia em Greenwich em 24 de novembro de 4714 a.C. de acordo com o calendário gregoriano proléptico
  - **INTEGER** – número de segundos desde a era Unix – 1970-01-01 00:00:00 UTC

[4] https://www.sqlite.org/lang_datefunc.html

# Comandos sqlite3

- Criando ou conectando em um banco

```
sqlite3 banco.db
```

- Listando os bancos conectados na sessão atual

```
sqlite> .databases
```

- Listando as tabelas e a instrução usada para criar uma tabela

```
sqlite> .tables
sqlite> .schema NomeDaTabela
```

- Exportando o resultado de uma consulta para um arquivo CSV

```
sqlite> .mode csv
sqlite> .output arquivo-saida.csv
sqlite> SELECT * FROM Aluno
```



# CREATE TABLE

```
1 CREATE TABLE Disciplina(
2    codigo INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
3    nome TEXT NOT NULL,
4    cargaHoraria INTEGER NOT NULL);
```

- Toda tabela possui uma coluna especial ROWID que identifica unicamente cada linha
  - `AUTOINCREMENT` evita o reuso de códigos do ROWID diante da exclusão de linhas
- Se houver uma coluna `INTEGER PRIMARY KEY`, então essa será um “apelido” para a coluna ROWID
- É possível criar uma tabela sem a coluna ROWID
  - Veja documentação oficial para conhecer as vantagens e desvantagens



# Comandos sqlite3

- Mudando a forma de apresentação dos resultados

```
sqlite> .header on
sqlite> .mode column
sqlite> SELECT * FROM disciplinas;

codigo      nome                              cargaHoraria
----------  --------------------------------  ------------
POO29004    Programação Orientada a Objetos   72
STD29006    Sistemas Distribuídos             54
BCD29008    Banco de Dados                    54
```

- Importando conteúdo de arquivo CSV para tabela

```
sqlite> .mode csv
sqlite> .import 'arquivo.csv' aluno
```

# Criando relacionamentos entre tabelas

```
1  -- configurando a integridade referencial
2  sqlite> PRAGMA foreign_keys = ON;
3
4  -- Criando tabela pessoa
5  CREATE TABLE Pessoa(
6    idPessoa    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
7    nome  TEXT
8  );
9  -- Criando tabela Telefone
10 CREATE TABLE Telefone(
11   idTel     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
12   rotulo   TEXT,
13   numero TEXT,
14   pessoa INTEGER,
15   FOREIGN KEY(pessoa) REFERENCES Pessoa(idPessoa)
16 );
17 -- Inserindo uma pessoa
18 INSERT INTO Pessoa (nome) VALUES ('Juca');
```



# ALTER TABLE

- Permite alterar o nome da tabela ou adicionar novas colunas
- Não é possível modificar, renomear ou excluir uma coluna existente
  - É necessário renomear a tabela
  - Criar uma nova tabela com as colunas desejadas
  - Copiar dados para nova tabela

```
1  -- No MySQL
2  ALTER TABLE Funcionarios ADD situacao TEXT;
3  -- No SQLite
4  PRAGMA foreign_keys=off;
5  BEGIN TRANSACTION;
6  ALTER TABLE Funcionario RENAME TO _funcionario_antigo;
7  CREATE TABLE ....
8  INSERT INTO Funcionario (...) SELECT (...) from _funcionario_antigo;
9  COMMIT;
10 PRAGMA foreign_keys=on;
```



# Demais funções do SQLite

- https://www.sqlite.org/lang_corefunc.html
- https://www.sqlite.org/lang_datefunc.html

```sql
SELECT date('now');
-- 2018-08-03

SELECT time('now');
-- 07:05:51

SELECT datetime('now');
-- 2018-08-03 07:05:51

SELECT strftime('%Y-%m-%d','now');
-- 2018-08-03
```



# Linguagem SQL - dialeto MySQL

# Alguns tipos de domínio para o MySQL

| Tipo | Descrição |
|---|---|
| `CHAR(M)` | Cadeia de caracteres de tamanho fixo. $0 \geq M \leq 255$ |
| `VARCHAR(M)` | Cadeia de caracteres de tamanho variável. $0 \geq M \leq 65.535$. UTF-8 requer 3 bytes por caractere, então tamanho máximo de 21.844 |
| `SMALLINT(M)` | Inteiro de $-32.768$ a $32.767$ |
| `INT(M)` | Inteiro de $-2.147.483.648$ a $2.147.483.648$ |
| `BIGINT(M)` | Inteiro de $-9.223.372.036.854.775.808$ a $9.223.372.036.854.775.807$ |
| `FLOAT(M,D)` | Real sendo $M$ o total de dígitos para a parte inteira e $D$ o total de digital para a parte decimal |
| `DOUBLE(M,D)` | Float com o dobro de precisão |
| `BOOLEAN` | Tipo booleano (TRUE ou FALSE) |
| `JSON` | Para guardar documentos JSON |



# Criando tabelas

https://dev.mysql.com/doc/refman/5.7/en/create-table.html

- Esquema de uma tabela
  - nome da tabela, atributos e seus tipos de domínio

```
1 CREATE TABLE Aluno(
2   matricula INT,
3   nome VARCHAR(80),
4   email VARCHAR(80));
```



# Criando tabelas

https://dev.mysql.com/doc/refman/5.7/en/create-table.html

- Esquema de uma tabela
  - nome da tabela, atributos e seus tipos de domínio

```
1 CREATE TABLE Aluno(
2   matricula INT,
3   nome VARCHAR(80),
4   email VARCHAR(80));
```

- Campo matrícula como chave primária e seu valor incrementado automaticamente para cada nova tupla inserida na tabela

```
1 CREATE TABLE Aluno(
2   matricula INT NOT NULL AUTO_INCREMENT,
3   nome VARCHAR(80),
4   email VARCHAR(80),
5   PRIMARY KEY (matricula));
```



# Criando tabelas

- Nome das colunas não precisam fazer referência ao nome da tabela
  - Ex: nomeAluno, emailAluno
- Para a chave primária é desejado que faça referência ao nome da tabela, pois esse campo poderá ser chave estrangeira em outra tabela
  - Ex: matriculaAluno



# Apagando e alterando tabelas

## Apagando

```sql
DROP TABLE Aluno;
```

## Alterando

```sql
ALTER TABLE Aluno ADD COLUMN tel INT;

ALTER TABLE Aluno CHANGE COLUMN tel telefone VARCHAR(40);

ALTER TABLE Aluno MODIFY COLUMN telefone VARCHAR(25);

ALTER TABLE Aluno DROP COLUMN telefone;
```



# Data Manipulation Language – DML

- **Inserindo**

```
1 INSERT INTO Aluno (nome, email, telefone)
2   VALUES ('Joao', 'j@email.co.br', '48-1234');
```

- **Apagando todas as linhas da tabela**

```
1 DELETE FROM Aluno;
```

- **Atualizando valores de todas as linhas da tabela**

```
1 UPDATE Aluno SET curso := 'Telecomunicações';
```

- **Recuperando (listando) todos os alunos**

```
1 SELECT * FROM Aluno
```



# Data Control Language – DCL

- Atribuindo privilégio de apenas consulta

```
1 GRANT SELECT ON academico.NOTAS TO 'appwebuser'@'localhost'
      identified by 'supersenha';
```

- Revogando todos privilégios

```
1 REVOKE ALL PRIVILEGES ON academico.* FROM 'bibliotecauser'@'%';
```



# Data Transaction Language – DTL

- `START TRANSACTION` – inicia uma nova transação
- `COMMIT` – efetiva uma transação
- `ROLLBACK` – desfaz a transação atual, cancelando qualquer mudança feita
- `SET autocommit` – habilita ou desabilita o mode de COMMIT automático

```
1 START TRANSACTION;
2   -- guarde na variável A o resultado da soma de todas as linhas da
      coluna salario e cujo cargo tem valor igual a 1
3   SELECT @A:=SUM(salario) FROM Salarios WHERE cargo=1;
4
5   -- Atualize o valor da coluna folha para o valor da variável A
6   UPDATE Financeiro SET folha=@A WHERE cargo=1;
7 COMMIT;
```

- Garantirá que se houver atualização da tabela Salarios, isso não irá influenciar a atualização da tabela Financeiro



# Operadores

| Operador | Descrição |
|---|---|
| `AND` , `&&` | E lógico |
| `OR` , `\|\|` | OU lógico |
| `NOT` , `!` | Negação |
| `!=` , `<>` | Diferente |
| `>`, `>=` | Maior, maior ou igual |
| `<`, `<=` | Menor, menor ou igual |
| `:=` | Atribuição |
| `=` | Atribuição para as instruções `SET` e `UPDATE`; igualdade para os demais contextos |
| `LIKE` | Busca por padrão |
| `IS [NOT] NULL` | Se [não] é NULO |
| `BETWEEN ...AND ...` | Valor dentro de uma faixa |

# Data Manipulation Language – DML

- **Cláusulas**
  - `FROM` – para especificar tabela
  - `WHERE` – para especificar condições
  - `GROUP BY` – para agrupar linhas
  - `HAVING` – condição por grupo
  - `ORDER BY` – para ordenar linhas
  - `DISTINCT` – selecionar dados sem repetição
  - `UNION` – para combinar duas consultas
- **Funções de agregação**
  - `AVG` – calcular média
  - `COUNT` – contar número de linhas
  - `SUM` – somar todos valore de um campo
  - `MAX` – maior valor em um campo



# SELECT – consulta SQL (query)

- Resultado de uma consulta SQL é uma tabela

```
1 SELECT A1, A2, ..., An
2 FROM T1, T2, ..., Tn
3 WHERE P
```

  - **A** – Atributo
  - **T** – Tabela
  - **P** – predicado da consulta

- Recuperando todas as colunas e linhas uma tabela

```
1 SELECT * FROM Aluno;
```

- Recuperando as colunas *nome* e `email` de todas as linhas

```
1 SELECT nome, email FROM Aluno;
```

- Recuperando todas disciplinas cursadas por um aluno e removendo duplicatas

```
1 SELECT DISTINCT disciplina FROM Aluno;
```



- Recuperando todos os dados de todos os alunos do curso de Telecomunicações e que moram em São José

```sql
SELECT *  FROM Aluno
WHERE curso = 'Telecomunicações' AND cidade = 'São José';
```

- Todos funcionários com salário maior que R$ 1.000,00

```sql
SELECT *  FROM Funcionarios
WHERE salario > 1000;
```

- Usando operadores aritméticos

```sql
SELECT 5 * 2 AS resultado;

SELECT horaExtra * valorHora * 2 as valorHE FROM funcionario;
```

# SELECT

- Ordenando o resultado

```sql
SELECT *  FROM Aluno
ORDER BY nome, matricula;
```

- Buscando por padrões em cadeias de caracteres
  - % – qualquer substring
  - _ – qualquer caracter

```sql
SELECT *  FROM Aluno  WHERE nome LIKE 'João%';

SELECT *  FROM Aluno  WHERE nome LIKE '%Silva%';

SELECT *  FROM Aluno  WHERE nome LIKE 'Jo_o %';
```

# SELECT: Funções de agregação

- Com exceção do `COUNT`, todos os demais ignoram tuplas que tenham o valor NULL nos atributos agregados

```
1 SELECT COUNT(*) AS totalDeAlunos FROM Aluno;
2
3 SELECT AVG(salario) FROM Funcionario;
```

- `GROUP BY` – Obtendo salário médio dos funcionários por departamento

```
1 SELECT departamento, AVG(salario)
2   FROM Funcionario GROUP BY departamento;
```

  - Atributos na instrução `SELECT` que estejam fora das funções de agregação devem obrigatoriamente aparecer na lista do `GROUP BY`



# SELECT

- Obtendo nomes dos departamentos cujo salário médio de seus funcionários seja maior que 1000

```
1 SELECT departamento, AVG(salario) FROM Funcionario
2   GROUP BY departamento
3     HAVING AVG(salario) > 1000;
```

- Retornando no máximo as 100 primeiras linhas

```
1 SELECT nome FROM Aluno LIMIT 100;
2 SELECT nome FROM Aluno LIMIT 0,100;
```

- Retornando no máximo 20 linhas a partir da linha 100

```
1 SELECT nome FROM Aluno LIMIT 100,20;
```



# Linguagem SQL - dialeto MySQL

## Funções para manipulação de datas

# MySQL: Tipos de dados para data e hora

| Tipo | Descrição |
|---|---|
| DATE | Para valores que só possuem data – faixa: '1000-01-01' até '9999-12-31'. |
| DATETIME | Para valores que possuem data e hora – faixa: '1000-01-01 00:00:00' até '9999-12-31 23:59:59' |
| TIMESTAMP | Para valores que possuem data e hora – faixa: '1970-01-01 00:00:01' UTC até '2038-01-19 03:14:07' UTC |

## Funções do MySQL para manipular data e hora

https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html



# MySQL: Funções DATE_FORMAT e TIME_FORMAT

| Máscara | Descrição | Exemplo |
|---|---|---|
| %m | mês | 01, . . . , 12 |
| %c | mês | 1, . . . , 12 |
| %M | mês por extenso | 'janeiro', . . . , 'dezembro' |
| %d | dia do mês | 01, . . . , 31 |
| %e | dia do mês | 1, . . . , 31 |
| %Y | ano - 4 dígitos | 2012, 2013, . . . |
| %y | ano - 2 dígitos | 12, 13, . . . |
| %h | hora - 12h | 01, . . . , 12 |
| %H | hora - 24h | 00, . . . , 23 |
| %i | minuto | 0, . . . , 59 |
| %s | segundo | 0, . . . , 59 |
| %W | dia da semana | 'domingo', . . . , 'sábado' |

```
1 mysql> select @@lc_time_names; -- verificando atual localização
2 mysql> set lc_time_names = 'pt_BR'; -- definindo localização para pt_BR
3 mysql> charset utf8; -- definindo codificação para utf8
```



# MySQL: Funções DATE_FORMAT e TIME_FORMAT

```sql
SELECT DATE_FORMAT('2017-03-31', '%e %M %Y'); --  31 março 2017

SELECT DATE_FORMAT('2017-03-29', '%d/%m/%Y'); -- 29/03/2017

SELECT TIME_FORMAT('15:40:00', '%Hh %im %ss'); -- 15h 40m 00s

SELECT CURRENT_DATE(); -- 2017-03-29

SELECT CURRENT_TIME(); -- 15:40:00

SELECT NOW(); -- 2017-03-29 15:40:00

SELECT DATEDIFF(CURRENT_DATE(), '2017-02-08'); -- valor em dias
```



# MySQL: Funções DATE_ADD e DATE_SUB

- Função para operações de soma e subtração com datas

```
DATE_ADD( data, INTERVAL expr unidade) ou DATE_SUB(...)
-- unidades: MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, YEAR
```

- Exemplos

```
SELECT DATE_ADD('2017-03-31 15:40', INTERVAL 2 HOUR); -- ... 17:40:00

SELECT DATE_SUB('2017-03-31 15:40', INTERVAL 1 YEAR); -- 2016-03-31 ...

SELECT DATE_ADD('2017-03-31 15:40:00', INTERVAL '1 2' DAY_HOUR); --
    2017-04-01 17:40:00

SELECT Nome FROM Aluno
WHERE CURRENT_DATE() > (DATE_ADD(DIngresso, INTERVAL 9 YEAR));
```



# Linguagem SQL - dialeto MySQL

## Manipulação de documentos JSON

# Manipulação de documentos JSON

```sql
CREATE TABLE cidade (idCidade INT, info JSON);

INSERT INTO cidade VALUES (JSON_OBJECT("populacao", "10000"));

INSERT INTO cidade VALUES ('{"populacao" : "2000"}');

SELECT info FROM cidade;

SELECT info->>"$.populacao" FROM cidade;
```

- https://dev.mysql.com/doc/refman/5.7/en/json.html