
FRANCA
DIGITAL

Iniciativa: Formação Digital

Curso:
Iniciação à Aprendizagem Computacional

FRANCA DIGITAL

SQL e Banco de Dados

Apresentação

Disciplina:

SQL e Banco de Dados Relacional

Professor: André Morais

Contato: andre.morais@luizalabs.com

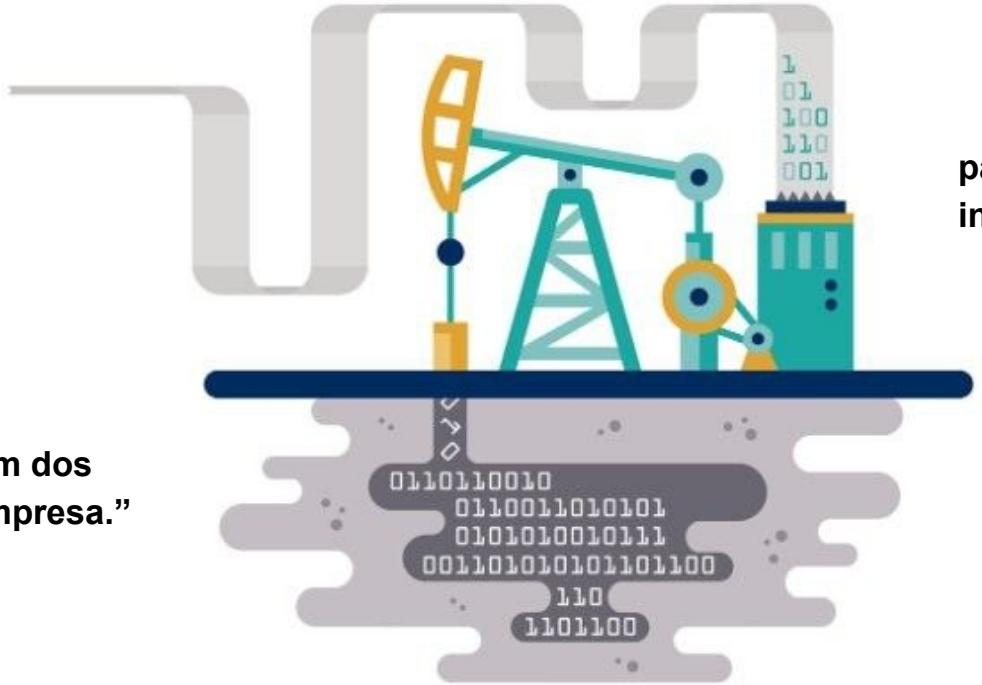
FRANCA DIGITAL

Conteudo das aulas

- **Banco de dados e arquitetura SGBD**
  - Banco de dados, banco de dados Relacional, SGBD, Arquitetura e Principais objetos
  - Ferramentas de Trabalho: MySql e MySql WorkBench
- **Linguagem SQL e subconjunto de comandos**
  - O que é SQL e seus subconjuntos: DQL, DML, DDL, DCL e DTL
- **Modelagem Relacional e Joins**
  - Modelo relacional, relacionamentos, joins e tipos de joins
- **Restrições de Integridade**
  - Constraints, restrições e tipos de integridade
- **Agrupamento e Agregação dos Dados**
  - Agregação dos dados, funções de grupo, cláusulas Group By e Having
- **Parte prática**
  - Prática usando as ferramentas durante as aulas e assuntos discutidos

FRANCA DIGITAL

Dados são o novo Petróleo?



“Dados são insumo para a construção de inteligência.”

“Cuide bem dos dados da sua empresa.”

SQL e Banco de Dados

Banco de dados e Arquitetura SGBD

Banco de dados

O que é um banco de dados?

Em tecnologia, um **banco de dados** é um repositório sistêmico de informações relacionadas a alguma coisa, aplicações ou sistemas. Podem ser, por exemplo, dados de clientes de um comércio, dados internos de uma empresa, nome e email de usuários cadastrados em uma rede social, e muitos outros. Eles também são conhecidos como **Database**.

Para que serve um banco de dados?

Um banco de dados serve, justamente, para que todas estas informações possam ser registradas e armazenadas de maneira **segura**, **organizada** e **padronizada**.

FRANCA DIGITAL

Bancos de Dados relacionais

Estes são os bancos de dados clássicos, baseados em **tabelas** e seus **relacionamentos**. Também conhecidos como **bancos de dados SQL**.

Isso significa que os dados armazenados neste tipo de banco de dados são mostrados ao usuário em formato de **tabelas**, suas **relações** e **restrições**.

Existem ainda os bancos de dados não relacionais, também conhecidos como **NoSQL**. Porém isso é papo para outro assunto… ;)

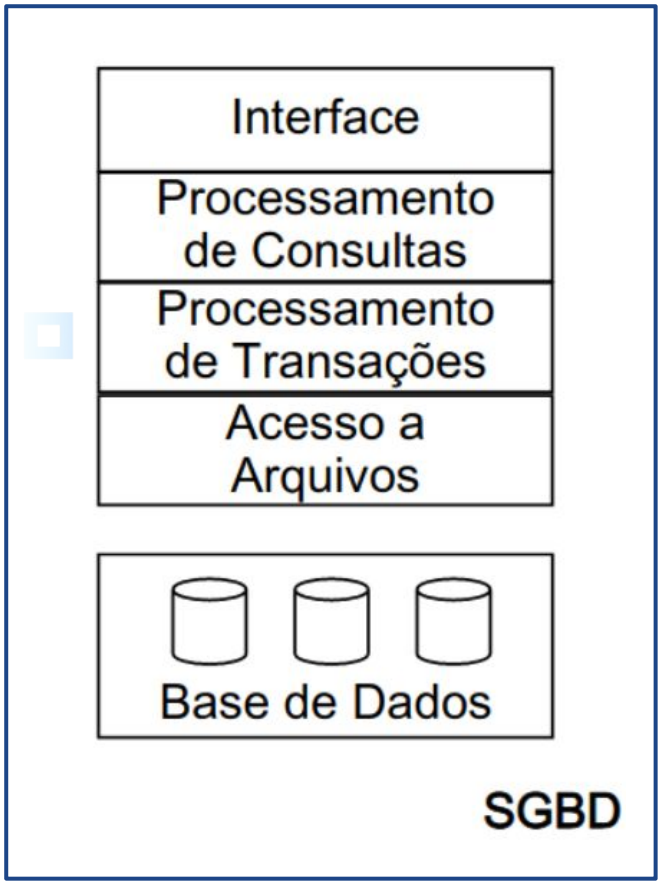
Bancos de Dados relacionais

Existem vários sistemas de bancos de dados relacionais, também chamados de **DBMS** - **D**ata**b**ase **M**anagement **S**ystems, ou **SGBD** - **S**istemas de **G**erenciamento de **B**ancos de **D**ados, em português.

Alguns exemplos de bancos de dados:

- Oracle
- **MySQL**
- SQL Server
- Postgre
- E muitos outros

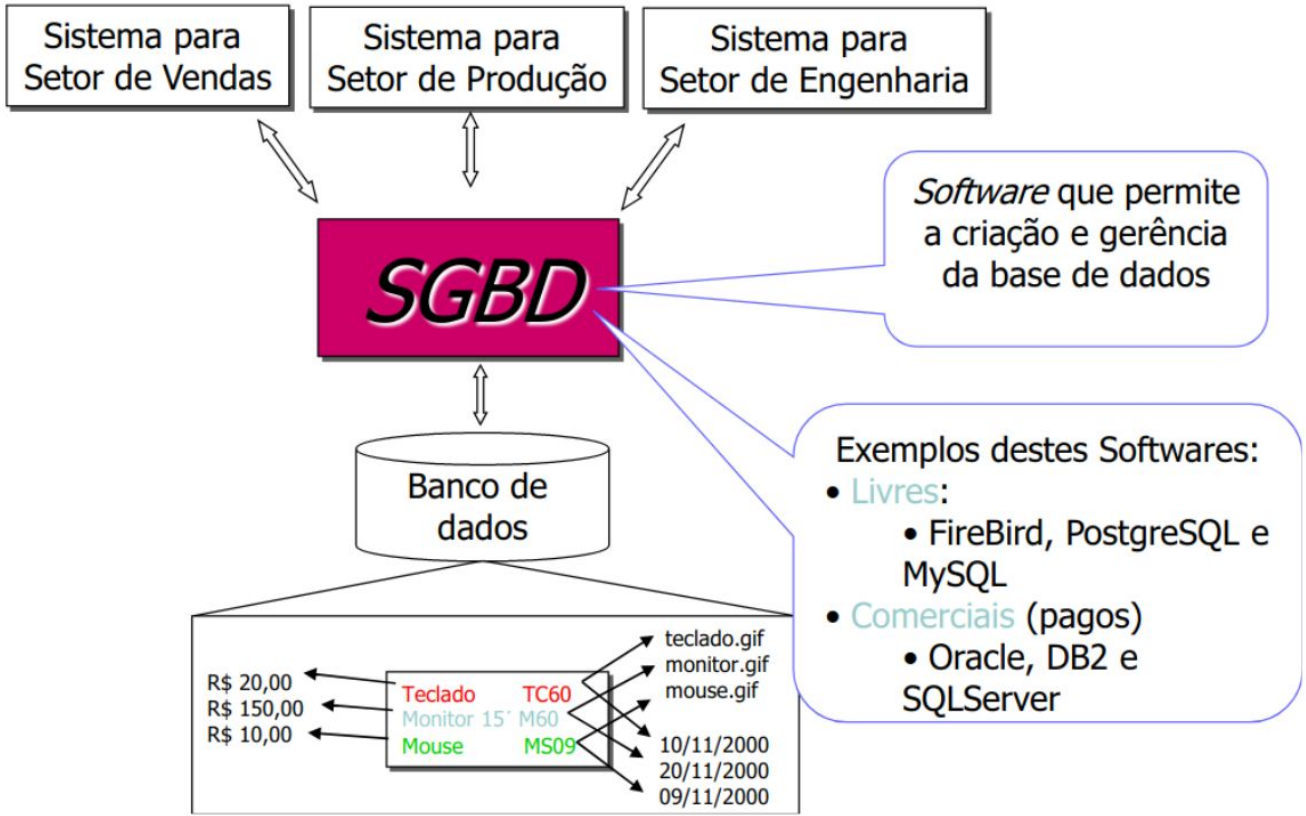
Arquitetura de um SGBD



Modelagem de dados
(Projeto Estrutural)

FRANCA DIGITAL

Aplicação de um SGBD



FRANCA DIGITAL

Principais objetos do Bancos de Dados Relacional

Schemas - Representa a configuração lógica de um banco de dados e coleção de objetos

- **Tabelas** - Conjunto de dados dispostos em colunas e linhas (ou tuplas)
  - **Chave Primária** - A PK é composta por uma ou mais colunas que identificam um registro da tabela como único.
  - **Chave Estrangeira** - A FK é uma ou mais colunas da tabela que caracteriza o relacionamento com a PK de outra tabela.
  - **Índices** - Estrutura de otimização para aumento de performance nas consultas em uma tabelas. São compostos de uma colunas ou mais contendo valores únicos ou não.
  - **Triggers** - É um bloco de código executado dependendo do evento ocorrido na tabela, inclusão, alteração ou deleção.

Tabelas - Conceito de forma prática

Tabela: Estrutura na qual os dados de um banco de dados são armazenados. Para imaginar melhor, é como se fosse uma super planilha de excel. A tabela é composta por **linhas** (tuplas ou registros) e **colunas**. Também conhecido por **Table**.

Exemplo

Imagine um sistema de contas a receber. Teremos um banco de dados que é responsável pelo armazenamento, acesso, controle dos dados. Dentro deste banco de dados teremos **várias tabelas**, cada uma responsável por armazenar a informação em uma determinada estrutura: “cliente” pode ser uma tabela, “contas_a_receber” outra e assim por diante… Esta estrutura das tabelas é a razão do termo dados estruturados.

Data Types - Conceito de forma prática

Toda coluna em uma tabela de banco de dados deve ter um nome, e também um **tipo de dados**, também conhecido como **Data Types** ou **Column Types**.

Um desenvolvedor **SQL** deve decidir que tipo de dados serão armazenados dentro de cada coluna **ao criar** uma tabela. O **tipo de dados** é uma diretriz para o SQL entender que tipo de dados é esperado dentro de cada coluna, e também identificar como o SQL irá interagir com os dados armazenados.

Link de Referência:

https://www.w3schools.com/sql/sql_datatypes.asp

FRANCA DIGITAL

Bancos de Dados MySql

O MySQL é um banco de dados relacional que pertence à Oracle. Uma das características mais marcantes desse modelo é o fato de se tratar de um **Open Source**. Utiliza a linguagem SQL, e funciona com as licenças de software comercial e livre.

O MySQL se destaca pelo seu fácil uso e uma estrutura de segurança e confiabilidade. Permitiu que empresas e aplicativos baseados na internet utilizassem seus recursos. Dentre os principais usuários estão o **Magalu**, Google, Facebook, Youtube, Twitter e NASA.

MySql Workbench

É uma ferramenta que permite fazer administração e manutenção em um banco de dados MySql. Através do **MySQL Workbench**, pode-se executar consultas SQL, administrar o sistema, modelar, criar e manter a base de dados através de um ambiente integrado. O **MySQL Workbench** está disponível para Windows, Linux e Mac OS.

FRANCA DIGITAL

SQL e Banco de Dados

Linguagem SQL e seus Subconjuntos de Comandos

Linguagem SQL

O **SQL** é uma linguagem padrão para **manipulação de registros** em banco de dados relacionais. A sigla **SQL** vem dos termos em inglês “**S**tructured **Q**uery **L**anguage”, que podem ser traduzidos para o português como “Linguagem de Consulta Estruturada”.

O **SQL** é usado para **criar**, **inserir**, **atualizar**, **excluir** e **consultar** as informações armazenadas na base, além de outras diversas funções mais complexas que permitem a administração do banco.

FRANCA DIGITAL

SQL - Conceito de forma prática

SQL é a linguagem de programação com a qual conseguimos nos comunicar com os dados e interagir com um banco de dados.

Exemplo:

Consultar o nome, telefone e endereço na tabela “cliente”, do sistema de contas a receber, passando como filtro seu código identificador:

```
SELECT nome, telefone, endereco
  FROM cliente
 WHERE cliente_id = 401220
```

SQL - Conceito de forma prática

Além de utilizar o **SQL** para consultar dados em um banco de dados, também podemos utilizá-lo para diversas outras funções, alguns exemplos são:

- Gerenciamento de permissões
- Criação de tabelas e outros objetos
- Manipulação de dados
  - Inserção
  - Atualização
  - Deleção
- E muitas outras! Veremos…

Link de Referência: https://www.w3schools.com/sql/default.asp

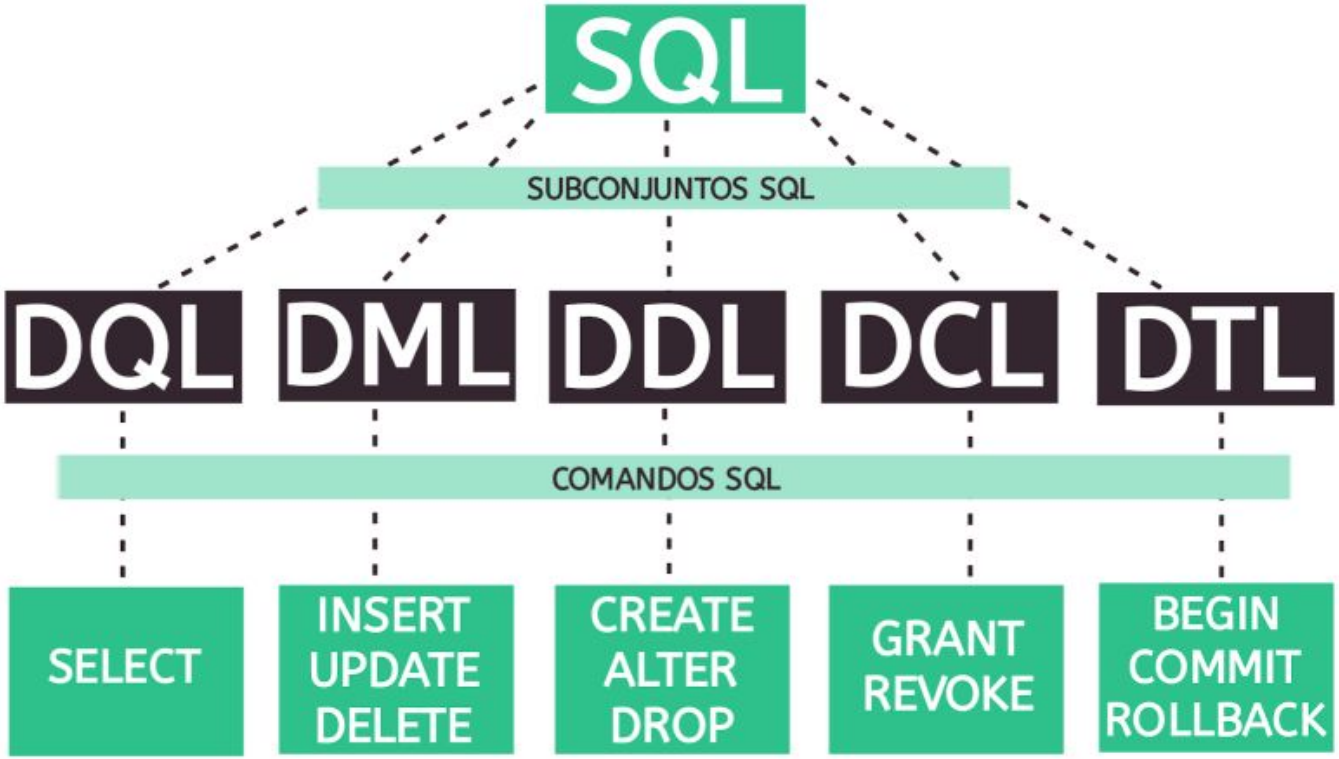
FRANCA DIGITAL

SQL - Conceito de forma prática

Segue alguns dos comandos mais importantes do SQL:

- SELECT - Consulta dados de um banco de dados
- UPDATE - Atualiza dados em um banco de dados
- DELETE - Exclui dados de um banco de dados
- INSERT INTO - Insere novos dados em um banco de dados
- CREATE DATABASE - Cria um novo banco de dados
- ALTER DATABASE - Modifica um banco de dados
- CREATE TABLE - Cria uma nova tabela
- ALTER TABLE - Modifica uma tabela
- DROP TABLE - Exclui uma tabela
- CREATE INDEX - Cria um índice (chave de pesquisa)
- DROP INDEX - Exclui um índice

Subconjuntos da Linguagem SQL



DQL: linguagem de consulta de dados

O **DQL** (**D**ata **Q**uery **L**anguage) é o subconjunto **SQL** que define o comando mais popular da linguagem, o **SELECT**. Esse comando é essencial para que possamos consultar os dados que armazenamos em nosso banco. A consulta também é conhecida por **Query**.

Consultando um cliente na respectiva tabela pelo seu código:

```
SELECT * FROM cliente WHERE cliente_id = 128221
```

DQL: linguagem de consulta de dados

CLÁUSULA **SELECT**

É a primeira sessão da Query onde se define os campos da tabela que queremos consultar e também se aplicamos algum operação matemática ou lógica nos campos:

- **`SELECT *`** : essa expressão retorna todas as colunas de uma tabela;
- **`SELECT coluna_a, coluna_b`**: essa expressão retorna as colunas a e b de uma tabela;
- **`SELECT count(coluna_a)`**: essa expressão, conta todos os valores não nulos da **coluna_a**;

No Select também podemos realizar operações e renomear colunas:

- **`SELECT coluna_a AS col_a`** : seleciona a **coluna_a** e renomeia para **col_a;**
- **`SELECT (coluna_a + coluna_b) AS soma`**: seleciona a soma da **coluna_a** e a **coluna_b**, e coloca o nome da coluna resultante como **soma.**

FRANCA DIGITAL

DQL: linguagem de consulta de dados

CLÁUSULA **FROM**

É a segunda sessão da Query, onde se define **a tabela ou tabelas** que queremos consultar. Mais adiante vamos ver como funciona a consulta de mais de uma tabela e a relação de uma com a outra…

- SELECT * **FROM tabela_a**: essa expressão retorna todas as colunas da tabela a;
- SELECT coluna_a, coluna_b from **FROM tabela_a**: essa expressão retorna as colunas a e b da tabela a;
- SELECT count(coluna_a) **FROM tabela_a**: essa expressão, conta todos os valores não nulos da coluna_a da tabela_a;

FRANCA DIGITAL

DQL: linguagem de consulta de dados

CLÁUSULA **WHERE**

É a terceira sessão da Query, onde se define **o(s) filtro(s)** que queremos aplicar sobre os dados da tabela. Para isso devemos utilizar o que chamamos de operadores, para validar se uma condição é verdadeira ou não. Como por exemplo, se 1 = 1 (Verdadeiro) ou se 1 é maior que 2 (Falso).

Os operadores são:

- **=** (igual) : verifica igualdade
- **>** (maior) : verifica se o número a esquerda é maior que o número da direita
- **<** (menor) : verifica se o número a direita é maior que o número da esquerda
- **>=** (maior igual) e **<=** (menor igual) : fazem a mesma coisa que os anteriores, porém também verificam a igualdade além se um número é maior que outro
- **in** : checa se um valor está dentro de uma lista.
- **between** : checa se um valor está entre um intervalo
- **like** : checa se um texto é parecido com outro através de parte do valor
    - O like é poderoso mas pode custar performance para sua query
    - Você consegue utilizar símbolos especiais com ele, como por exemplo o **%**, que indica qualquer coisa

DQL: linguagem de consulta de dados

CLÁUSULA **WHERE**

Para adicionarmos mais de um filtro, utilizamos operadores especiais:

- **AND** - Usamos quando queremos que **TODAS** as condições **SEJAM** verdadeiras.
- **OR** - Usamos quando queremos que uma condição **OU** outra **SEJA** verdadeira.

Exemplo:

A query abaixo retorna todas as colunas da tabela “categoria”, porém apenas as linhas que possuem o “perc_parceiro” maior que 2 e menor ou igual a 5:

```
SELECT *
  FROM categoria
 WHERE perc_parceiro > 2 AND perc_parceiro <= 5
```

FRANCA DIGITAL

DML: linguagem de manipulação de dados

O **DML** (**D**ata **M**anipulation **L**anguage) é o subconjunto SQL que define os comandos usados para **manipular** os dados armazenados em um banco. Esse é um dos conjuntos mais utilizados, pois ele fornece operadores que nos permitem **inserir**, **excluir** e **alterar** os registros de uma tabela, por exemplo. Os comandos mais importantes desse subconjunto são: **INSERT**, **DELETE** e **UPDATE**.

Comando **INSERT**:

```
INSERT INTO cliente (cliente_id, nome, status)
VALUES (128222, 'JOSE MARQUES', 'ATIVO'
```

FRANCA DIGITAL

DML: linguagem de manipulação de dados

Comando **UPDATE**:

```
UPDATE cliente
SET status = 'INATIVO'
WHERE cliente_id = 128222
```

Comando **DELETE**:

```
DELETE cliente
WHERE status = 'INATIVO'
```

FRANCA DIGITAL

DDL: linguagem de definição de dados

O **DDL** (**D**ata **D**efinition **L**anguage) é o subconjunto SQL que apresenta comandos usados para **gerenciar as estruturas** do banco de dados. Com ele, podemos **criar, atualizar e remover objetos** da base, como **tabelas** e **índices**. Os comandos definidos pelo DDL são: **CREATE**, **DROP** e **ALTER**..

Comando **CREATE**:

```
CREATE TABLE cliente
(cliente_id  INT PRIMARY KEY
,nome        VARCHAR(100)
,status      VARCHAR(30));

CREATE INDEX idx_nome ON Cliente(nome);
```

FRANCA DIGITAL

DCL: linguagem de controle de dados

O **DCL** (**D**ata **C**ontrol **L**anguage) é o subconjunto no qual encontramos comandos para **controlar o acesso** aos dados da nossa base. Utilizando esse conjunto, conseguimos estabelecer **restrições e permissões** para quem acessa o banco por meio dos comandos **GRANT** e **REVOKE**. Muito importante para garantir a segurança no banco de dados.

Dando permissão de criação de objetos e de consulta para o usuário

```
GRANT CREATE, SELECT ON *. * TO  'admin'@'localhost';
```

Revogando todos os acessos e privilégios do usuário da tabela “cliente”

```
REVOKE ALL PRIVILEGES ON curso.cliente FROM
'admin'@'localhost' ;
```

FRANCA DIGITAL

DTL ou TCL: linguagem de transação de dados

O **DTL** (**D**ata **T**ransaction **L**anguage), também conhecido como **TCL** (**T**ransaction **C**ontrol **L**anguage), é o subconjunto **SQL** que define comandos que utilizamos quando é necessário **gerenciar transações** feitas no banco. Isso significa que eles permitem iniciar, confirmar e desfazer determinadas alterações de comandos **DML**. Os comandos estabelecidos pelo conjunto são:

- **COMMIT -** Consolida uma transação no banco;
- **ROLLBACK -** Desfaz uma transação no banco;
- **BEGIN -** Inicia um bloco de comandos para controle na mesma transação. É o **START TRANSACTION** do MySql;

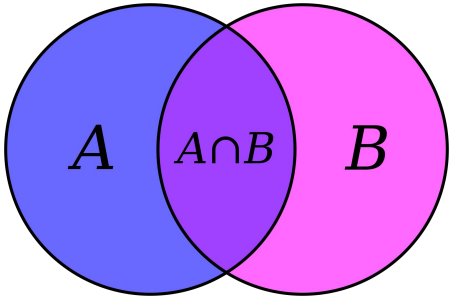
FRANCA DIGITAL

SQL e Banco de Dados

Modelagem Relacional e Joins

Modelo Relacional

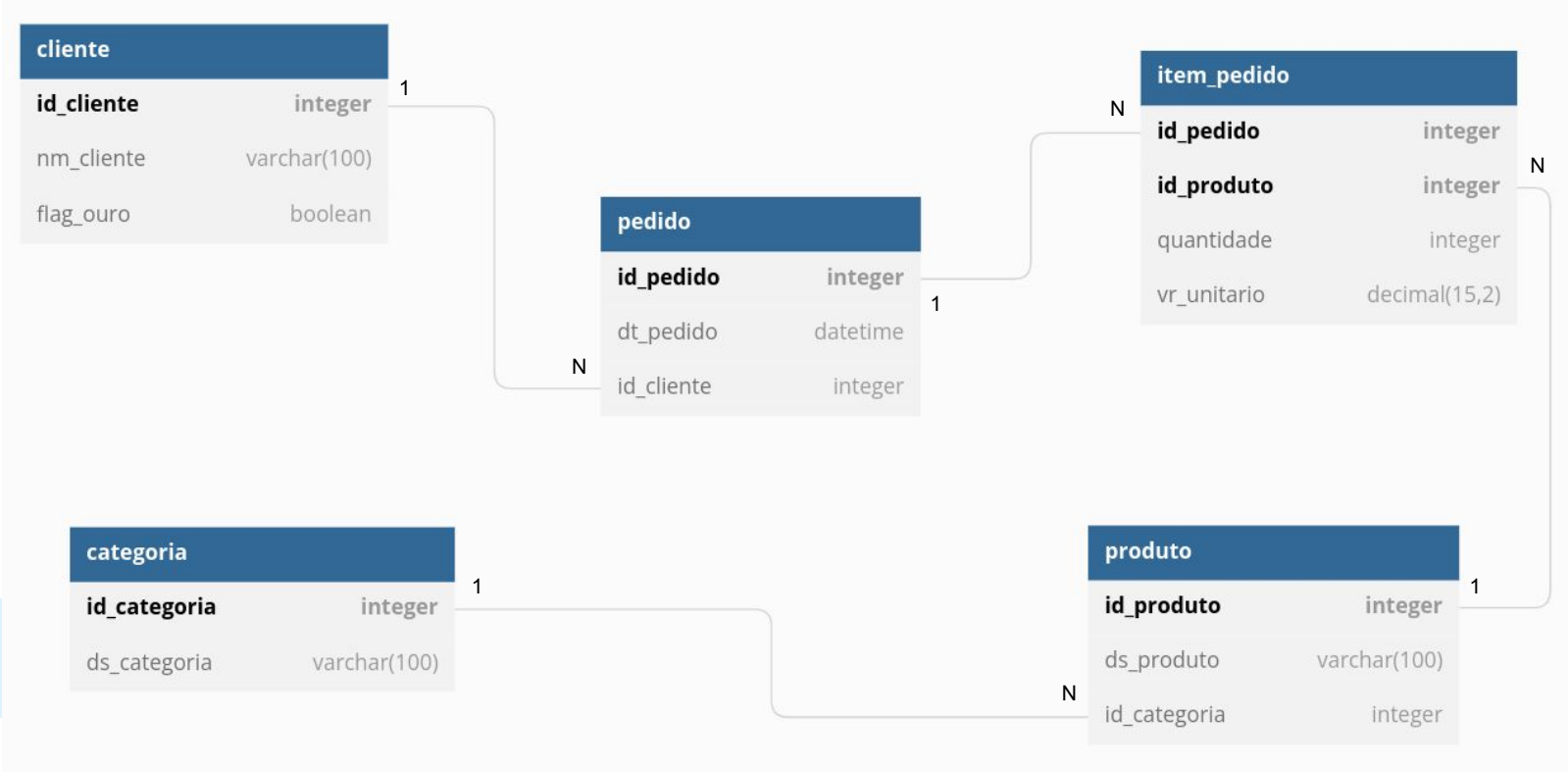
O **modelo relacional** é um modelo de dados representativo que se baseia no princípio em que **todos os dados estão guardados em tabelas relacionadas entre si**. Podemos nos basear na teoria dos conjuntos quando selecionamos e relacionamos elementos de uma tabela A com a tabela B.



O modelo relacional é composto de **entidades** e **relacionamentos**. Uma **entidade** pode ser uma tabela física ou uma visão de diversas tabelas.

FRANCA DIGITAL

Modelagem Relacional



FRANCA DIGITAL

Estrutura Relacional

Determinados os **relacionamentos**, temos que verificar o número de referências de uma entidade em outra, ou seja, verificar a **CARDINALIDADE** dos relacionamentos. Vejamos as possibilidades:

- **Relacionamento Um-Para-Um (1:1)** - Um registro da tabela A relaciona-se a um registro da tabela B
- **Relacionamento Um-Para-Vários (1:N)** - Um registro da tabela A relaciona-se a vários registros da tabela B
- **Relacionamento Vários-Para-vários (N:M)** - Vários registros da tabela A relacionam-se a vários registros da tabela B

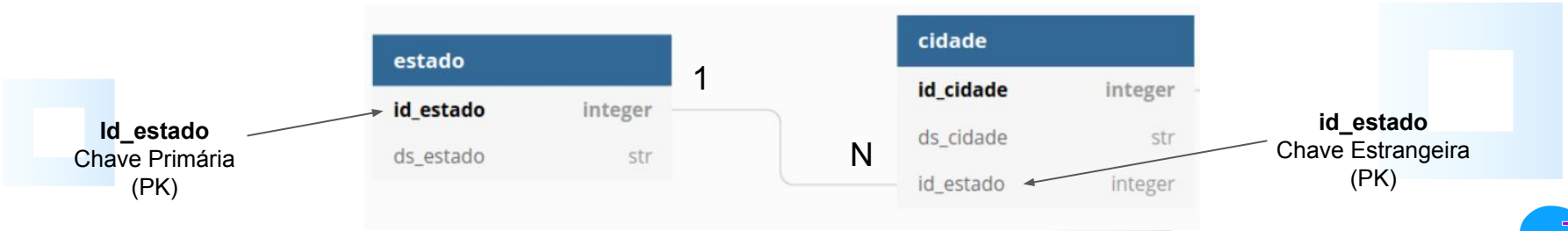
FRANCA DIGITAL

Estrutura Relacional

Na relação entre a tabela **Estado** e **Cidade** é possível dizer que ambas possuem um **relacionamento 1:N**, pois um 1 estado possui N cidades.

Esse relacionamento ocorre pela **chave** “estado_id”. Se filtrarmos o estado de SP na tabela cidade, ela nos mostrará todas as cidades do estado, ex: São Paulo, Campinas, Jundiaí, Franca, etc…

Olhando para a teoria dos conjuntos, é correto dizer que Franca e Jundiaí estão contidas no estado de SP. Dizemos então, que a tabela “Estado” é a tabela “**Pai**” e a tabela “Cidade” é a tabela “**Filha**” na relação.



Join entre as Tabelas

Em **SQL**, os relacionamentos entre as tabelas são representados através da cláusula **JOIN**, que corresponde a uma operação de **junção**, combinando colunas de uma ou mais tabelas em um banco de dados relacional.

```
SELECT cli.id_cliente
      ,cli.nome
      ,ped.data_pedido
      ,ped.valor_pedido
  FROM pedido as ped
       INNER JOIN cliente as cli
       ON (ped.id_cliente = cli.id_cliente)
 WHERE data_pedido >= '2022-07-01'
```

FRANCA DIGITAL

Join entre as Tabelas

Geralmente o **JOIN** se dá pela coluna chave primária (**PK**) da tabela “Pai” e a chave estrangeira (**FK**), que identifica a entidade, na tabela “Filha”.

Porém, o **JOIN** pode ser feito por mais de um campo que represente a relação entre as tabelas (chaves compostas).

O SQL padrão ANSI especifica cinco tipos de JOIN :

- **INNER** JOIN
- **LEFT** JOIN
- **RIGHT** JOIN
- **FULL** JOIN
- **CROSS** JOIN

Inner Join

A cláusula **INNER JOIN** compara cada linha da **tabela A** com as linhas da **tabela B** para encontrar **todos** os pares. Se a condição de junção for **TRUE**, os valores correspondentes das tabelas A e B serão combinados em uma nova linha e incluído no resultado.

```
SELECT <select_list>
  FROM Tabela A
 INNER JOIN Tabela B
    ON (A.Key = B.Key)
```

FRANCA DIGITAL

Left Join

A cláusula **LEFT JOIN** retorna **todos** os registros da tabela **esquerda** e os registros **correspondentes** da tabela **direita** no resultado.

Com isso, os registros **não** encontrados na tabela da **direita,** o resultado de suas colunas serão **NULL**.

```
SELECT <select_list>
  FROM Tabela A
  LEFT JOIN Tabela B
    ON (A.Key = B.Key)
```

FRANCA DIGITAL

Right Join

Ao contrário da cláusula Left Join, o **RIGHT JOIN** retorna **todos** os registros da tabela **direita** e os registros **correspondentes** da tabela **esquerda** no resultado.

Com isso, os registros **não** encontrados na tabela da **esquerda,** o resultado de suas colunas serão **NULL**.

```
SELECT <select_list>
  FROM Tabela A
  RIGHT JOIN Tabela B
    ON (A.Key = B.Key)
```

Full Join

A cláusula **FULL JOIN** é um combinado entre o **Right Join** e o **Left Join**, pois retorna **todos** os registros da tabela **direita** e todos da tabela **esquerda** no resultado do relacionamento.

Com isso, os registros **não** encontrados na tabela da **esquerda** ou **direita** pela chave do relacionamento, terão o resultado de suas colunas como **NULL**.

```
SELECT <select_list>
  FROM Tabela A
  FULL JOIN Tabela B
    ON (A.Key = B.Key)
```

Cross Join

O **CROSS JOIN** retorna todas as linhas das tabelas por cruzamento, ou seja, para cada linha da tabela esquerda queremos todas as linhas da tabelas direita ou vice-versa. Também chamado de produto cartesiano entre duas tabelas.

Repare que esse tipo de relacionamento é bem **específico**, e **não compara** colunas entre as tabelas, apenas realiza o **cartesiano** entre as linhas das mesmas..

```
SELECT <select_list>
  FROM Tabela A
  CROSS JOIN Tabela B
```

FRANCA DIGITAL

Exemplo Prático

Resolver a quantidade de pedidos por cidades localizadas no estado de São Paulo. Observe a consulta SQL:

```
SELECT cid.ds_cidade
      ,count(ped.id_pedido) as qtde_pedido
  FROM pedido as ped
 INNER JOIN filial as fil
    ON (ped.id_filial = fil.id_filial)
 INNER JOIN cidade as cid
    ON cid.id_cidade = fil.id_cidade
 INNER JOIN estado as est
    ON cid.id_estado = est.id_estado
 WHERE est.ds_estado =  "SP"
 GROUP BY cid.ds_cidade
```

SQL e Banco de Dados

Restrições de Integridade

SQL Constraints

As **restrições SQL** são usadas para especificar regras para os dados em uma tabela.

As restrições são usadas para limitar o tipo de dados que podem entrar em uma tabela. Isso garante a precisão e confiabilidade dos dados na tabela. Se houver alguma violação entre a restrição e a ação de dados, a ação será abortada com erro.

Elas podem ser em nível de coluna ou nível de tabela. As restrições de nível de coluna se aplicam à coluna e as restrições de nível de tabela se aplicam à tabela toda.

SQL Constraints

As seguintes **restrições** são comumente usadas em **SQL**:

- NOT NULL - Garante que uma coluna não pode ter um valor NULL
- UNIQUE - Garante que todos os valores em uma coluna sejam diferentes
- PRIMARY KEY - Uma combinação de NOT NULL e UNIQUE. Identifica exclusivamente cada linha em uma tabela
- FOREIGN KEY - Evita ações que destruiriam relacionamento entre tabelas
- CHECK - Garante que os valores em uma coluna satisfaçam uma condição específica
- COLUMN TYPE - Tipo definido para a coluna no momento de sua criação
- DEFAULT - Define um valor padrão para uma coluna se nenhum valor for especificado

Restrição de Integridade

A **integridade de dados** se refere a acurácia e consistência dos dados armazenados em um sistema de banco de dados relacional (ou outro sistema). A **restrição de integridade** garante que os dados armazenados possam ser armazenados, consultados e utilizados com confiabilidade, sendo assim dados íntegros.

Tipos de restrições de integridade:

- Integridade Referencial
- Integridade de Domínio
- Integridade de Vazio
- Integridade de Chave
- Integridade Definida pelo Usuário

Integridade Referencial

Essa restrição assegura que valores de uma coluna em uma tabela são válidos baseados nos valores em uma outra tabela relacionada.

Cada valor de uma chave estrangeira na tabela “Filha” deve corresponder a um valor de uma chave primária existente na tabela “Pai”, e refere-se ajustamento ao relacionamento entre as entidades

Caso alguma inclusão, alteração e deleção que não respeitar a relação entre as tabelas “pai” e “filha” tomará um erro de restrição.

Garantido pela Constraint abaixo:

- FOREIGN KEY - Evita ações que destruiriam relacionamento entre tabelas

Integridade de Domínio

Os valores inseridos em uma **coluna** devem sempre obedecer à definição dos **valores** que são **permitidos** para essa coluna. São chamados valores do ***domínio***.

Exemplo 1: Em uma coluna “preço”, os valores admitidos são do domínio numérico, ou seja, apenas números. Não é permitido preços usando letras em sua representação. A própria definição do tipo da coluna é uma restrição.

Exemplo 2: Em uma coluna de situação de cadastro, aceita-se apenas “A” (ativo) e “I” (inativo). O domínio da coluna são os valores “A” e “I”, não é permitido mais nenhum valor.

Garantido pelas Constraints abaixo:

- CHECK - Garante que os valores em uma coluna satisfaçam uma condição específica
- COLUMN TYPE - Tipo definido para a coluna no momento da criação da tabela

Integridade de Vazio

Este tipo de integridade informa se a coluna é obrigatória ou opcional, ou seja, se é possível **não inserir um valor** na coluna.

Uma coluna de chave primária, por exemplo, sempre deve ter dados inseridos, e nunca pode estar vazia, para nenhum registro.

Garantido pelas Constraints abaixo:

- NOT NULL - Garante que uma coluna não pode ter um valor NULL
- PRIMARY KEY - Uma combinação de NOT NULL e UNIQUE. Identifica exclusivamente cada linha em uma tabela
- DEFAULT - Define um valor padrão para uma coluna se nenhum valor for especificado

Integridade de Chave

Os valores inseridos na coluna de chave primária (PK) devem ser sempre únicos, não admitindo-se repetições nesses valores. Desta forma, os registros serão sempre distintos.

Os valores de chave primária também não podem ser nulos.

Garantido pelas Constraints abaixo:

- PRIMARY KEY - Uma combinação de NOT NULL e UNIQUE. Identifica exclusivamente cada linha em uma tabela
- UNIQUE - Garante que todos os valores em uma coluna sejam diferentes

Integridade definida pelo usuário

Diz respeito a regras de negócio específicas que são definidas pelo usuário do banco de dados. Por exemplo, pode-se definir que uma coluna somente aceitará um conjunto restrito de valores.

Exemplo 1: Apenas as siglas dos estados da federação serão aceitos, e mais nenhum outro valor diferente.

Exemplo 2: Um campo “idade” de uma tabela “estudante” aceitará somente “`idade<=18`”

Garantido pelas Constraints abaixo:

- CHECK - Garante que os valores em uma coluna satisfaçam uma condição específica

SQL e Banco de Dados

Agrupamentos e agregações dos dados

Agregação dos dados

A **agregação dos dados** diz respeito ao processo de reunião e tratamento de dados, para os apresentar sob a forma de um relatório resumido, a fim de facilitar a obtenção de determinados resultados. Através do **agrupamento dos dados** em uma **consulta SQL** podemos obter um resultado analítico.

Quando queremos extrair **métricas** de uma tabela, como por **exemplo**, uma **contagem** de registros ou a **soma** do valor dos pedidos, devemos utilizar as **funções de agregação**.

As funções de agregação são muito utilizadas em linguagem SQL para **promover a possibilidade de criação de diversos tipos de relatórios com dados agrupados**. São importantes recursos que programadores usam para implementar relatórios em sistemas de informação, web sites e aplicativos para celulares.

Funções de Agregação

As **Funções de agregação** são utilizadas juntamente com as colunas na cláusula **SELECT** da **query**. Principais funções de agregação:

- **COUNT** : Conta valores não nulos de uma coluna
- **SUM** : Soma valores numéricos não nulos de uma coluna
- **AVG** : Tira a média de valores numéricos de uma coluna
- **MIN** : Pega o menor valor de uma coluna
- **MAX** : Pega o maior valor de uma coluna

Exemplo:

```
SELECT COUNT(*) as qtde_clientes
      ,AVG(idade) as media_idade
  FROM cliente
```

FRANCA DIGITAL

Cláusula Group By

Utilizamos cláusula **GROUP BY** para extrair métricas por alguma **granularidade** específica, por exemplo:

- Contagem de Pedidos por Seller
- Valor total de venda por Dia, etc…

Devemos especificar a **coluna de granularidade** no **SELECT** e também na cláusula **GROUP BY**, que vem logo após a cláusula **FROM** ou **WHERE** na Query.

Exemplo:

```
SELECT data_pedido
      ,SUM(valor_pedido) as vr_venda_dia
  FROM pedido
 GROUP BY data_pedido
```

Cláusula Having

A cláusula **HAVING** é utilizada em consultas de agregação, após o **GROUP BY**, quando se deseja filtrar por uma condição ou valor referente ao resultado de uma função de agregação, como o **COUNT(*)** por exemplo.

- Exemplo: Valor total de vendas por Dia, **desde que** o valor da soma seja maior que 100,00.

Para esse tipo de filtro **não** é possível utilizar a cláusula **WHERE**, pois a engine de consulta a interpreta antes do agrupamento.

Exemplo:

```
SELECT data_pedido
      ,SUM(valor_pedido) as vr_venda_dia
  FROM pedido
 WHERE data_pedido >= '2022-07-01'
 GROUP BY data_pedido
HAVING SUM(valor_pedido)>100
```

FRANCA DIGITAL

SQL e Banco de Dados

Obrigado!!

FRANCA DIGITAL

“Compartilhar conhecimento não faz você melhor do que os outros, pelo contrário, todo professor é aluno duas vezes.”

Felipe Angelo

“Lembre-se que as pessoas podem tirar tudo de você, menos seu conhecimento”

Albert Einstein