**ESTUDO PROVA BANCO DE DADOS**

**Structured Query Language**

**Tipos de Dados usados para criação dos atributos**

**Números inteiros**

* **SMALLINT**: Inteiro pequeno (2 bytes, -32.768 a 32.767)
* **INTEGER / INT**: Inteiro padrão (4 bytes, -2.147.483.648 a 2.147.483.647)
* **BIGINT**: Inteiro grande (8 bytes, para valores muito grandes)

**Números decimais**

* **NUMERIC(p, s)**: Precisão exata, usado para valores como dinheiro (ex: NUMERIC(10, 2))
* **DECIMAL(p, s)**: Igual ao NUMERIC
* **REAL**: Ponto flutuante (4 bytes), precisão aproximada
* **DOUBLE PRECISION**: Ponto flutuante (8 bytes), maior precisão

**Texto**

* **CHAR(n)**: Texto de tamanho fixo (ex: CHAR(10) ocupa sempre 10 caracteres)
* **VARCHAR(n)**: Texto de tamanho variável até n caracteres
* **TEXT**: Texto de tamanho ilimitado

**Data e Hora**

* **DATE**: Apenas data (formato: AAAA-MM-DD)
* **TIME**: Apenas hora (formato: HH:MM:SS)
* **TIMESTAMP**: Data e hora combinadas
* **TIMESTAMPTZ**: Data e hora com fuso horário

**Booleano**

* **BOOLEAN**: Pode armazenar TRUE, FALSE ou NULL

**Identidade (auto incremento)**

* **SMALLSERIAL**: Inteiro auto-incrementável pequeno (2 bytes)
* **SERIAL**: Inteiro auto-incrementável padrão (4 bytes)
* **BIGSERIAL**: Inteiro auto-incrementável grande (8 bytes)

**Binário**

* **BYTEA**: Armazena dados binários, como arquivos ou imagens

**Outros Especiais**

* **UUID**: Identificador único universal (ex: usado como chave primária)
* **JSON / JSONB**: Armazena dados em formato JSON. JSONB é otimizado para busca
* **ARRAY**: Armazena listas de valores (ex: TEXT[], INTEGER[])
* **ENUM**: Conjunto fixo de valores possíveis (ex: 'pequeno', 'médio', 'grande')

**DDL (Data Definition Language)**

Usado para definir e estruturar um banco de dados. Esses comandos não manipulam os dados, apenas criam ou alteram tabelas e outros objetos.

* **CREATE:** Cria tabelas, bancos de dados, índices etc.
* **ALTER**: Modifica a estrutura de tabelas (adicionar/remover colunas).
* **DROP**: Remove tabelas ou bancos de dados.
* **TRUNCATE**: Remove todos os dados de uma tabela sem apagar a estrutura.

**CREATE SCHEMA** (Cria um esquema de BD relacional)

|  |
| --- |
| CREATE SCHEMA vendas |

**DROP SCHEMA** (Remove um esquema de BD relacional)

* **CASCADE** (Remove o esquema BD incluindo todas suas tabelas e outros elementos)
* **RESTRICT** (Remove um esquema BD somente se não existir elementos definidos para esse esquema)

|  |
| --- |
| DROP SCHEMA vendas RESTRICT;  DROP SCHEMA vendas CASCADE; |

Obs: o esquema serve para agrupar as tabelas e outros comandos que pertencem à mesma aplicação

**CREATE TABLE** (Cria uma nova tabela – relação – no BD, ela não possui dados inicialmente)

|  |
| --- |
| CREATE TABLE clientes (  id SERIAL PRIMARY KEY,  nome VARCHAR(100) NOT NULL,  email VARCHAR(100) UNIQUE,  idade INTEGER,  criado\_em TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  ); |

**DROP TABLE** (remove uma tabela – relação – e todas suas instâncias do BD)

|  |
| --- |
| DROP TABLE clientes; |

**ALTER TABLE** (Altera a estrutura de uma tabela – relação – já existente no BD)

|  |
| --- |
| -- Adicionar uma coluna  ALTER TABLE clientes ADD COLUMN telefone VARCHAR(20);  -- Modificar o tipo de uma coluna  ALTER TABLE clientes ALTER COLUMN idade TYPE SMALLINT;  -- Renomear uma coluna  ALTER TABLE clientes RENAME COLUMN telefone TO celular;  -- Renomear a tabela  ALTER TABLE clientes RENAME TO consumidores;  -- Remover uma coluna  ALTER TABLE consumidores DROP COLUMN celular; |

**Restrição de Integridade**

**Valor nulo (NULL)**: Não precisa de um valor específico

|  |
| --- |
| CREATE TABLE produtos (  id SERIAL PRIMARY KEY,  nome VARCHAR(100),  descricao TEXT NULL  ); |

**Restrição não nula (NOT NULL):** Usamos quando não é permitido um valor nulo

|  |
| --- |
| CREATE TABLE categorias (  id SERIAL PRIMARY KEY,  nome VARCHAR(100) NOT NULL  ); |

**Comparações em consultas:** Usar “IS NULL” e “IS NOT NULL”

|  |
| --- |
| -- Buscar clientes sem email cadastrado  SELECT \* FROM clientes WHERE email IS NULL;  -- Buscar clientes com email cadastrado  SELECT \* FROM clientes WHERE email IS NOT NULL; |

**Cláusula DEFAULT:** Associa um valor predeterminado para um atributo caso nenhum seja especificado

|  |
| --- |
| CREATE TABLE pedidos (  id SERIAL PRIMARY KEY,  status VARCHAR(20) DEFAULT 'pendente'  ); |

**Cláusula CHECK:** Especifica um predicado que precisa ser satisfeito por todas as tuplas de uma relação

|  |
| --- |
| CREATE TABLE funcionarios (  id SERIAL PRIMARY KEY,  nome VARCHAR(100),  salario NUMERIC CHECK (salario > 0)  ); |

**Cláusula PRIMARY KEY:** Identifica os atributos da relação que formam a sua chave primária (obrigatoriamente NOT NULL)

|  |
| --- |
| CREATE TABLE departamentos (  id INTEGER PRIMARY KEY,  nome VARCHAR(50) ); |

**Cláusula UNIQUE:** Não permite valores duplicados para um determinado atributo

|  |
| --- |
| CREATE TABLE usuarios (  id SERIAL PRIMARY KEY,  nome\_usuario VARCHAR(50) UNIQUE  ); |

**Cláusula FOREIGN KEY:** Integridade referencial (dependência existente entre a chave estrangeira de uma relação e a chave primária da relação relacionada). Com essa cláusula eliminamos a possibilidade de violação da integridade referencial.

|  |
| --- |
| CREATE TABLE pedidos (  id SERIAL PRIMARY KEY,  cliente\_id INTEGER,  FOREIGN KEY (cliente\_id) REFERENCES clientes(id)  ); |

**CREATE DOMAIN** (Domínio: Conjunto de valores válidos que um atributo pode assumir, podendo ser usados dentro da criação de tabelas)

|  |
| --- |
| CREATE DOMAIN idade\_valida AS INT  CHECK (VALUE >= 0 AND VALUE <= 120);  CREATE TABLE pessoa (  nome TEXT,  idade idade\_valida  ); |

**DROP DOMAIN**

|  |
| --- |
| DROP DOMAIN idade\_valida; |

**ALTER DOMAIN**

|  |
| --- |
| -- renomear domínio  ALTER DOMAIN idade\_valida RENAME TO idade\_limitada;  -- adicionar uma restrição  ALTER DOMAIN idade\_limitada ADD CONSTRAINT idade\_minima CHECK (VALUE >= 18);  -- remover uma restrição  ALTER DOMAIN idade\_limitada DROP CONSTRAINT idade\_minima; |

Obs: As características de um domínio são globais ao BD

**CREATE INDEX** (Índice: Estrutura de dados usada para acelerar a busca de informações dentro de uma tabela)

|  |
| --- |
| -- Cria índice simples na coluna 'nome' da tabela 'cliente'  CREATE INDEX idx\_nome ON cliente(nome);  -- Cria um índice único na coluna 'email' da tabela 'cliente'  CREATE UNIQUE INDEX idx\_email\_unico ON cliente(email);  -- Cria índice em múltiplas colunas  CREATE INDEX idx\_nome\_sobrenome ON cliente(nome, sobrenome);  -- Cria índice ordenado de forma descendente  CREATE INDEX idx\_data\_desc ON pedidos(data\_compra DESC);  -- Cria índice parcial (somente para clientes ativos)  CREATE INDEX idx\_ativos ON cliente(situacao) WHERE situacao = 'ativo';  -- Cria índice com função (para buscas case-insensitive)  CREATE INDEX idx\_lower\_email ON cliente(LOWER(email)); |

**DROP INDEX**

|  |
| --- |
| DROP INDEX idx\_nome;  -- Remove índice apenas se existir (evita erro)  DROP INDEX IF EXISTS idx\_nome; |

**ALTER INDEX**

|  |
| --- |
| -- Renomeia o índice  ALTER INDEX idx\_email\_unico RENAME TO idx\_cliente\_email\_unico;  -- Altera o proprietário do índice  ALTER INDEX idx\_cliente\_email\_unico OWNER TO novo\_usuario;  -- Reconstrói o índice (útil após muitas mudanças na tabela)  REINDEX INDEX idx\_cliente\_email\_unico; |

**DML (Data Manipulation Language)**

Usado para inserir, atualizar e excluir dados dentro das tabelas.

* **INSERT:** Insere novos dados na tabela.
* **UPDATE:** Atualiza dados existentes.
* **DELETE:** Remove registros de uma tabela.
* **SELECT:** (embora seja mais considerado DQL, está dentro da manipulação).

**INSERT INTO** (Para inserir uma linha na relação – tabela)

|  |
| --- |
| -- Inserção completa (todos os campos)  INSERT INTO clientes (nome, email, cidade, ativo)  VALUES ('Ana Souza', 'ana@email.com', 'São Paulo', TRUE);  -- Inserção parcial (colunas não incluídas usarão o valor DEFAULT ou NULL)  INSERT INTO clientes (nome, email)  VALUES ('Carlos Lima', 'carlos@email.com'); |

**UPDATE** (Atualizar atributos de linhas)

|  |
| --- |
| -- Atualiza o email e a cidade do cliente com id = 1  UPDATE clientes  SET email = 'ana.souza@email.com',  cidade = 'Rio de Janeiro'  WHERE id = 1;  -- Desativa todos os clientes da cidade de São Paulo  UPDATE clientes  SET ativo = FALSE  WHERE cidade = 'São Paulo'; |

**DELETE** (Excluir linhas da relação)

|  |
| --- |
| -- Remove o cliente com id = 2  DELETE FROM clientes  WHERE id = 2;  -- Remove todos os clientes inativos  DELETE FROM clientes  WHERE ativo = FALSE; |

**DQL (Data Query Language)**

A DQL é uma subcategoria do SQL usada exclusivamente para consultas. O principal comando aqui é o SELECT, que recupera dados de uma ou mais tabelas

* **WHERE:** filtra os dados com base em condições.
* **ORDER BY:** ordena os resultados.
* **GROUP BY:** agrupa dados (usado com funções de agregação).
* **HAVING:** filtra grupos após um GROUP BY.
* **JOIN:** permite combinar dados de várias tabelas.

**Formas de fazer pesquisa/seleções de linhas e ou colunas**

**Cláusula SELECT:** Lista os atributos e/ou funções a serem exibidas no resultado da consulta

**Cláusula FROM:** Especifica as relações (tabelas) a serem examinadas na avaliação da consulta

**Cláusula WHERE**: Especifica as condições para a seleção das tuplas no resultado da consulta, as condições devem ser definidas sobre os atributos das relações que aparecem na cláusula FROM. Inclui condições de junções

* Operadores: AND, OR e NOT

Tabela

O conteúdo gerado por IA pode estar incorreto.

* **Operador LIKE:** 
  + Caractere coringa: “%” substitui qualquer string e “\_” substitui qualquer caractere
  + Sensível a letra maiúscula e minúsculas

|  |
| --- |
| -- Seleciona todos os dados da tabela  SELECT \* FROM clientes;  -- Seleciona apenas os nomes e e-mails de todos os clientes  SELECT nome, email FROM clientes;  -- Seleciona clientes que moram em 'São Paulo'  SELECT \* FROM clientes  WHERE cidade = 'São Paulo';  -- Seleciona clientes com mais de 30 anos  SELECT \* FROM clientes  WHERE idade > 30;  -- Seleciona clientes entre 18 e 25 anos  SELECT \* FROM clientes  WHERE idade BETWEEN 18 AND 25;  -- Seleciona clientes inativos  SELECT \* FROM clientes  WHERE ativo = FALSE;  -- Clientes com idade diferente de 40  SELECT \* FROM clientes  WHERE idade != 40;  -- Clientes com idade maior ou igual a 60  SELECT \* FROM clientes  WHERE idade >= 60;  -- Clientes com cidade 'São Paulo' ou 'Rio de Janeiro'  SELECT \* FROM clientes  WHERE cidade IN ('São Paulo', 'Rio de Janeiro');  -- Clientes que não estão nessas cidades  SELECT \* FROM clientes  WHERE cidade NOT IN ('Curitiba', 'Belo Horizonte');  -- Clientes cujo nome começa com 'A'  SELECT \* FROM clientes  WHERE nome LIKE 'A%';  -- Clientes cujo nome termina com 's'  SELECT \* FROM clientes  WHERE nome LIKE '%s';  -- Clientes cujo nome contém 'silva'  SELECT \* FROM clientes  WHERE nome LIKE '%silva%';  -- Clientes cujo email é do domínio gmail  SELECT \* FROM clientes  WHERE email LIKE '%@gmail.com';  -- Clientes cujo nome tem exatamente 5 letras  SELECT \* FROM clientes  WHERE nome LIKE '\_\_\_\_\_'; -- 5 underlines = 5 letras  -- Clientes ativos com mais de 25 anos  SELECT \* FROM clientes  WHERE ativo = TRUE AND idade > 25;  -- Clientes inativos ou com idade abaixo de 18  SELECT \* FROM clientes  WHERE ativo = FALSE OR idade < 18;  -- Clientes com nome contendo 'joão' e morando no RJ  SELECT \* FROM clientes  WHERE LOWER(nome) LIKE '%joão%' AND cidade = 'Rio de Janeiro'; |

**Cláusula ORDER BY:** Ordena as tuplas que aparecem no resultado da pesquisa, ASC (ascendente) ou DESC(descendente)

|  |
| --- |
| -- Ordena por valor em ordem crescente  SELECT \* FROM pedidos  ORDER BY valor;  -- Ordena por valor em ordem decrescente  SELECT \* FROM pedidos  ORDER BY valor DESC;  -- Ordena por cidade (A-Z) e, dentro da cidade, pelo valor (maior para menor)  SELECT \* FROM pedidos  ORDER BY cidade ASC, valor DESC; |

**SELECT DISTINCT / ALL:** Distinct não considera duplas duplicas e All considera todas tuplas

|  |
| --- |
| -- Lista todas as cidades únicas dos pedidos (sem repetições)  SELECT DISTINCT cidade FROM pedidos;  -- Clientes únicos que fizeram pedidos  SELECT DISTINCT cliente FROM pedidos;  -- Esse comando é igual a SELECT padrão, pois ALL é implícito  SELECT ALL cliente, cidade FROM pedidos; |

**Cláusula GROUP BY:** Permite aplicar uma função de agregação não somente a um conjunto de tuplas, mas também a um grupo de conjunto de tuplas

|  |
| --- |
| -- Soma dos pedidos por cliente  SELECT cliente, SUM(valor) AS total\_gasto  FROM pedidos  GROUP BY cliente;  -- Quantidade de pedidos por cidade  SELECT cidade, COUNT(\*) AS total\_pedidos  FROM pedidos  GROUP BY cidade; |

**Cláusula HAVING:** Permite especificar uma condição de seleção para grupos

|  |
| --- |
| -- Clientes que gastaram mais de 500 no total  SELECT cliente, SUM(valor) AS total\_gasto  FROM pedidos  GROUP BY cliente  HAVING SUM(valor) > 500;  -- Cidades com mais de 2 pedidos  SELECT cidade, COUNT(\*) AS total\_pedidos  FROM pedidos  GROUP BY cidade  HAVING COUNT(\*) > 2; |

**Funções de Agregação**

**Média:** AVG()

**Mínimo:** MIN()

**Máximo:** MAX()

**Soma:** SUM()

**Contagem:** COUNT()

|  |
| --- |
| -- Média de valores de todos os pedidos  SELECT AVG(valor) AS media\_pedidos  FROM pedidos;  -- Média de valor por cidade  SELECT cidade, AVG(valor) AS media\_por\_cidade  FROM pedidos  GROUP BY cidade;  -- Pedido mais barato  SELECT MIN(valor) AS menor\_valor  FROM pedidos;  -- Menor pedido por status  SELECT status, MIN(valor) AS pedido\_minimo  FROM pedidos  GROUP BY status;  -- Pedido mais caro  SELECT MAX(valor) AS maior\_valor  FROM pedidos;  -- Maior valor de pedido por cliente  SELECT cliente, MAX(valor) AS pedido\_mais\_caro  FROM pedidos  GROUP BY cliente;  -- Soma total de todos os pedidos  SELECT SUM(valor) AS total\_vendas  FROM pedidos;  -- Total vendido por cidade  SELECT cidade, SUM(valor) AS total\_por\_cidade  FROM pedidos  GROUP BY cidade;  -- Total de pedidos  SELECT COUNT(\*) AS total\_pedidos  FROM pedidos;  -- Total de clientes únicos  SELECT COUNT(DISTINCT cliente) AS total\_clientes  FROM pedidos;  -- Número de pedidos por status  SELECT status, COUNT(\*) AS total\_por\_status  FROM pedidos  GROUP BY status; |

**Junções**

**Cláusula ON:** usada em JOINs para especificar a condição de junção entre duas tabelas — ou seja, qual coluna em uma tabela se relaciona com qual coluna da outra.

**CROSS JOIN:** Faz o produto cartesiano: cada linha da primeira tabela é combinada com todas as linhas da segunda.

|  |
| --- |
| SELECT clientes.nome, pedidos.valor  FROM clientes  CROSS JOIN pedidos; |

**INNER JOIN:** Retorna apenas as linhas que têm correspondência em ambas as tabelas.

|  |
| --- |
| -- Mostra só os clientes que fizeram pedidos  SELECT clientes.nome, pedidos.valor  FROM clientes  INNER JOIN pedidos ON clientes.id = pedidos.cliente\_id; |

**LEFT JOIN:** Retorna todas as linhas da tabela da esquerda e os dados correspondentes da tabela da direita. Se não houver correspondência, os campos da direita vêm como NULL.

|  |
| --- |
| -- Mostra todos os clientes, mesmo os que não fizeram pedidos  SELECT clientes.nome, pedidos.valor  FROM clientes  LEFT JOIN pedidos ON clientes.id = pedidos.cliente\_id; |

**RIGHT JOIN:** Retorna todas as linhas da tabela da direita, e os dados correspondentes da esquerda. Se não houver correspondência, os campos da esquerda vêm como NULL.

|  |
| --- |
| -- Mostra todos os pedidos, mesmo os que estão sem clientes associados  SELECT clientes.nome, pedidos.valor  FROM clientes  RIGHT JOIN pedidos ON clientes.id = pedidos.cliente\_id; |

**FULL JOIN**: Retorna todas as linhas de ambas as tabelas, com correspondência onde houver. Onde não houver, preenche com NULL.

|  |
| --- |
| -- Junta todos os clientes e todos os pedidos, inclusive os que não tem  -- correspondência entre si  SELECT clientes.nome, pedidos.valor  FROM clientes  FULL JOIN pedidos ON clientes.id = pedidos.cliente\_id; |

**SELF JOIN:** É quando uma tabela faz JOIN com ela mesma.

|  |
| --- |
| SELECT A.nome AS cliente1, B.nome AS cliente2  FROM clientes A  JOIN clientes B ON A.id != B.id; |

**SELECTs aninhados**

são consultas dentro de outras consultas. Ou seja, você coloca um comando SELECT dentro de outro para utilizar o resultado de uma consulta como parte de outra. Essas subconsultas podem ser usadas em diversas partes da consulta principal, como no SELECT, FROM, WHERE, e até em HAVING.

**Exemplos:**

|  |
| --- |
| -- EXEMPLOS DE SELECTS ANINHADOS (SUBCONSULTAS)  -- Exemplo 1: Subconsulta no WHERE  -- Objetivo: Buscar os clientes que fizeram pedidos com valor maior que o maior valor do cliente com id = 1  SELECT nome  FROM clientes  WHERE id IN (  SELECT cliente\_id  FROM pedidos  WHERE valor > (  SELECT MAX(valor)  FROM pedidos  WHERE cliente\_id = 1  )  );  -- Exemplo 2: Subconsulta no FROM  -- Objetivo: Mostrar o total de pedidos apenas para clientes que fizeram mais de 2 pedidos  SELECT cliente\_id, SUM(valor) AS total\_pedidos  FROM (  SELECT cliente\_id, valor  FROM pedidos  WHERE cliente\_id IN (  SELECT cliente\_id  FROM pedidos  GROUP BY cliente\_id  HAVING COUNT(\*) > 2  )  ) AS pedidos\_filtrados  GROUP BY cliente\_id;  -- Exemplo 3: Subconsulta no SELECT  -- Objetivo: Mostrar o nome de cada cliente e o total que ele já gastou (soma dos valores de seus pedidos)  SELECT nome, (  SELECT SUM(valor)  FROM pedidos  WHERE cliente\_id = clientes.id  ) AS total\_gasto  FROM clientes;  -- Exemplo 4: Subconsulta correlacionada  -- Objetivo: Para cada pedido, mostrar se ele foi o maior feito por aquele cliente  SELECT id, cliente\_id, valor,  CASE  WHEN valor = (  SELECT MAX(valor)  FROM pedidos AS p2  WHERE p2.cliente\_id = p1.cliente\_id  )  THEN 'MAIOR PEDIDO'  ELSE 'NORMAL'  END AS status\_pedido  FROM pedidos AS p1;  -- Exemplo 5: Subconsulta simples com comparação  -- Objetivo: Mostrar os pedidos que têm valor maior que a média geral  SELECT \*  FROM pedidos  WHERE valor > (  SELECT AVG(valor)  FROM pedidos  ); |