Principais Conceitos da Linguagem SQL

A linguagem SQL pode ser utilizada de diversas formas, mas as duas principais são: embutida num programa de aplicação, dentro, por exemplo, de um programa escrito em uma linguagem como Java ou Python; ou diretamente codificada numa interface de execução dentro de um SGBD, como o SQLServer ou o MySQL, por exemplo.

Nos SGBD, a SQL assume um papel interessante, pois além de ser extremamente simples, pode ter enfoques diferentes e ser bastante poderosa. Com a utilização de comandos SQL, os programadores podem, por exemplo, construir no SGBD consultas complexas sem a necessidade de criação de um programa e receber de imediato as respostas que necessitam para dar continuidade ao desenvolvimento das aplicações.

O DBA, que é a pessoa responsável pela administração do BD, por sua vez, irá utilizar uma parte da SQL mais relacionada com a definição da estrutura dos dados, com o gerenciamento do acesso aos dados por parte dos usuários, com o monitoramento da utilização dos dados no dia a dia da organização, dentre outros. Ou seja, como podemos observar, a SQL não é apenas uma linguagem de consulta do banco de dados como imaginamos ao traduzir seu nome, mas sim uma linguagem muito mais abrangente que é utilizada em toda a tarefa de gerenciamento dos BD.

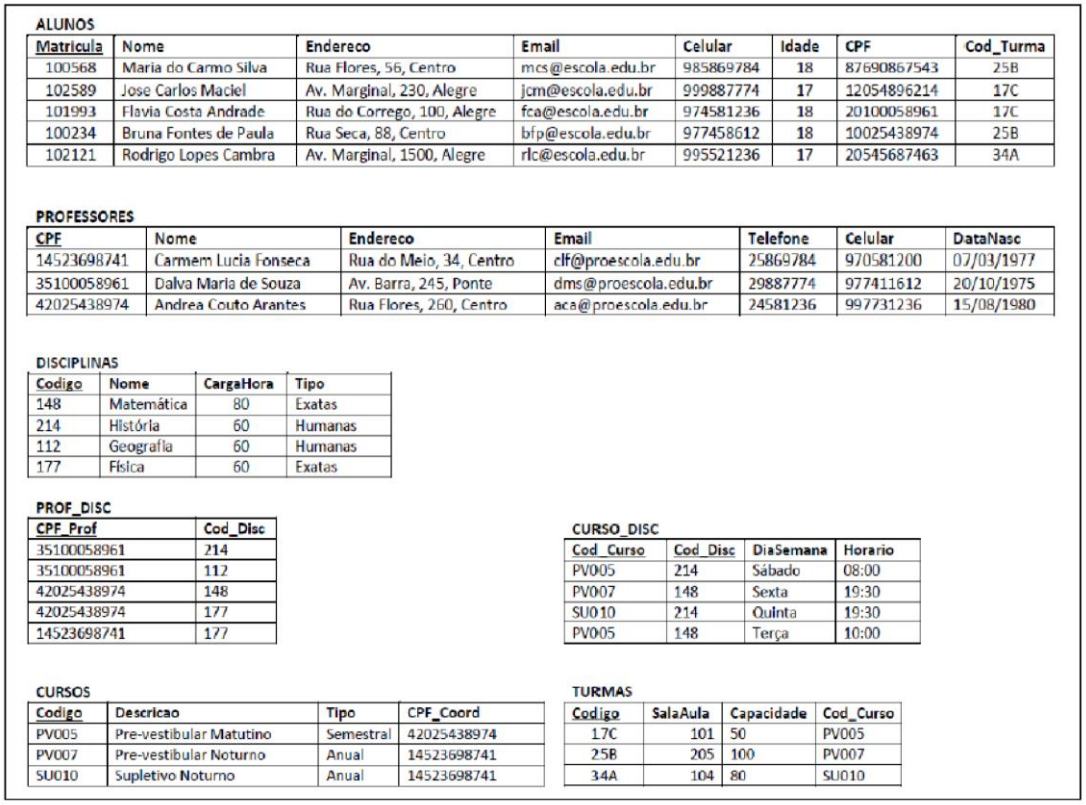
Segundo Elmasri e Navathe (2018), devido ao fato de possuir várias aplicações, a linguagem SQL provê suporte a várias funções de um SGBD e por esse motivo é dividida em algumas partes específicas. Dentre elas, vamos destacar as três partes mais importante:

- DDL, do inglês, Data Definition Language (ou linguagem de definição de dados), onde os dados a serem armazenados são definidos e estruturados;
- DML, do inglês, Data Manipulation Language (ou linguagem de manipulação de dados), que permite inclusão, remoção, seleção ou atualização de dados armazenados no banco de dados;
- DCL, do inglês, Data Control Language (ou linguagem de controle de dados), que realiza o controle de acesso, permitindo proteção dos dados de manipulações não autorizadas.

Além dessas características principais, ainda podemos citar o suporte a visões, onde são especificadas as consultas disponíveis através de tabelas virtuais e especificação de transações, para garantia do compartilhamento dos dados (ELMASRI e NAVATHE, 2018).

A linguagem SQL utiliza todos os conceitos de tabela, coluna e linha, para representar os conceitos de relação, atributo e tupla do modelo relacional. Para fins didáticos, utilizaremos ao longo deste texto o exemplo do BD Escola, já apresentado, com o objetivo de ilustrar os comandos SQL.

Relembrando o modelo, a Figura ilustra o BD Escola.



ALUNOS

Matricula	Nome	Endereco	Email	Celular	Idade	CPF	Cod_Turma
100568	Maria do Carmo Silva	Rua Flores, 56, Centro	mcs@escola.edu.br	985869784	18	87690867543	25B
102589	Jose Carlos Maciel	Av. Marginal, 230, Alegre	jcm@escola.edu.br	999887774	17	12054896214	17C
101993	Flavia Costa Andrade	Rua do Corrego, 100, Alegre	fca@escola.edu.br	974581236	18	20100058961	17C
100234	Bruna Fontes de Paula	Rua Seca, 88, Centro	bfp@escola.edu.br	977458612	18	10025438974	25B
102121	Rodrigo Lopes Cambra	Av. Marginal, 1500, Alegre	rlc@escola.edu.br	995521236	17	20545687463	34A

PROFESSORES

CPF	Nome	Endereco	Email	Telefone	Celular	DataNasc
14523698741	Carmem Lucia Fonseca	Rua do Meio, 34, Centro	clf@proescola.edu.br	25869784	970581200	07/03/1977
35100058961	Dalva Maria de Souza	Av. Barra, 245, Ponte	dms@proescola.edu.br	29887774	977411612	20/10/1975
42025438974	Andrea Couto Arantes	Rua Flores, 260, Centro	aca@proescola.edu.br	24581236	997731236	15/08/1980

DISCIPLINAS

Codigo	Nome	CargaHora	Tipo
148	Matemática	80	Exatas
214	História	60	Humanas
112	Geografia	60	Humanas
177	Física	60	Exatas

PROF_DISC

CPF_Prof	Cod_Disc
35100058961	214
35100058961	112
42025438974	148
42025438974	177
14523698741	177

CURSO_DISC

Cod_Curso	Cod_Disc	DiaSemana	Horario
PV005	214	Sábado	08:00
PV007	148	Sexta	19:30
SU010	214	Quinta	19:30
PV005	148	Terça	10:00

CURSOS

Codigo	Descricao	Tipo	CPF_Coord
PV005	Pre-vestibular Matutino	Semestral	42025438974
PV007	Pre-vestibular Noturno	Anual	14523698741
SU010	Supletivo Noturno	Anual	14523698741

TURMAS

Codigo	SalaAula	Capacidade	Cod_Curso
17C	101	50	PV005
25B	205	100	PV007
34A	104	80	SU010

Figura: Modelo Relacional do BD Escola

Criando um BD com os Comandos da DDL

Para definirmos a estrutura de um banco de dados relacional, utilizamos os comandos de definição de dados da SQL, que compõem a DDL (Data Definition Language). São três comandos principais: CREATE, ALTER e DROP.

Nem todas as versões da SQL apresentam o comando create database, mas ele é utilizado para definir um nome para o BD a ser criado. Quando ainda não existe

um banco de dados criado no SGBD, o primeiro passo é criá-lo. Se não existir esse comando, existirá outro semelhante para o mesmo fim, ou será oferecida uma interface gráfica no SGBD para criação do BD.

Por exemplo: CREATE DATABASE BD_ESCOLA;

O comando create table é utilizado para definir ou criar uma tabela, com a opção de nesse momento já definir também as suas restrições de integridade de entidade e referencial dentro do BD. Para isso, ele contém as cláusulas:

PRIMARY KEY: usada para indicar os atributos que formam a chave primária;

UNIQUE KEY: usada para indicar os atributos que formam a chave alternativa;

FOREIGN KEY: usada para indicar os atributos que formam a chave estrangeira e o nome da relação

Referida pela chave estrangeira.

Formato do comando create table:

CREATE TABLE <nome tabela>

(Nome atributo tipo_de_dado [NOT NULL], primary key (nome_atributo1, nome_atributo2,...) foreign key (nome atributo) referencies nome relação);

Antes de prosseguirmos, vale ressaltarmos que existem tipos básicos de dados predefinidos para a linguagem SQL. Esses tipos de dados básicos serão utilizados para definir os domínios dos atributos e podem variar em cada SGBD. Os tipos de dados mais utilizados são listados a seguir.

Tipo de Dados	Descrição
CHAR (N)	Conjunto de caracteres de tamanho fixo = N
VARCHAR (N)	Conjunto de caracteres de tamanho variável <= N
REAL	Número real
INTEGER	Número inteiro
DATE / TIME	Data / Hora

Por exemplo, para a criação de todas as tabelas do BD Escola, deveríamos escrever os seguintes comandos. É interessante no momento de criação de um banco de dados, salvar os comandos de criação destes, chamados de script do banco de dados, pois na fase de implantação de um sistema de BD, provavelmente o banco de

dados para ser testado, deverá ser criado, apagado e novamente criado diversas vezes. Tudo que aparece ao lado dos comandos entre parêntesis {} é apenas um comentário explicativo, não faz parte do comando.

```
CREATE TABLE ALUNOS (
Matrícula                    CHAR (6) NOT NULL,
Nome                         VARCHAR (150) NOT NULL,
Endereco                     VARCHAR (150) NOT NULL,
Email                        VARCHAR (40) NOT NULL,
Celular                          CHAR (12),
Idade                        INTEGER,
CPF                          CHAR (11) NOT NULL,
Cod_Turma  Char (3),
PRIMARY KEY (Matricula),          {este atributo é a chave primária}
UNIQUE KEY (CPF));           {este atributo é a chave alternativa}
```

Observe o comando de criação acima, nele podemos perceber que foi escolhido o nome da tabela “ALUNOS”, em seguida estão listados os atributos da tabela com seus respectivos domínios (tipos de dados), depois a cláusula que define se o atributo é obrigatório (NOT NULL) e por fim vem a definição da chave primária (PRIMARY KEY) da tabela. Pelo modelo, essa tabela possui uma chave estrangeira “Cod_Turma” da tabela TURMAS, mas nesse momento essa chave ainda não pode ser definida porque a tabela TURMAS ainda não existe no BD.

```
CREATE TABLE PROFESSORES (
CPF                  CHAR (11) NOT NULL,
Nome                 VARCHAR (150) NOT NULL,
Endereço             VARCHAR (150) NOT NULL,
E-mail               VARCHAR (40) NOT NULL,
Telefone             CHAR (12) NOT NULL,
```

Celular CHAR (12),

DataNasc DATE NOT NULL,

PRIMARY KEY (CPF); {este atributo é a chave primária}

CREATE TABLE DISCIPLINAS (

Código CHAR (5) NOT NULL,

Nome VARCHAR (30) NOT NULL,

Carga Hora INTEGER,

Tipo VARCHAR (30),

PRIMARY KEY (CODIGO); {este atributo é a chave primária}

CREATE TABLE PROF_DISC (

CPF_Prof CHAR(11) NOT NULL,

Cod_Disc CHAR (5) NOT NULL,

PRIMARY KEY (CPF_Prof, Cod_Disc), {estes atributos compõem uma chave primária}

FOREIGN KEY (CPF_Prof) REFERENCES PROFESSORES, {chave estrangeira} FOREIGN KEY (Cod_Disc) REFERENCES DISCIPLINAS); {chave estrangeira}

CREATE TABLE CURSOS (

Código CHAR (5) NOT NULL,

Descrição VARCHAR (40) NOT NULL,

Tipo VARCHAR (20) NOT NULL,

CPF_Coord CHAR (11) NOT NULL,

PRIMARY KEY (Código), {este atributo é a chave primária}

FOREIGN KEY (CPF_Coord) REFERENCES PROFESSORES); {chave estrangeira}

CREATE TABLE CIRSO_DISC (

Cod_Curso CHAR (5) NOT NULL,

Cod_Disc CHAR (5) NOT NULL,

DiaSemana VARCHAR (20) NOT NULL,

Horario TIME,

PRIMARY KEY (Cod_Curso, Cod_Disc), {estes atributos compõem uma chave primária}

FOREIGN KEY (Cod_Curso) REFERENCES CURSOS, {chave estrangeira}

FOREIGN KEY (Cod_Disc) REFERENCES DISCIPLINAS); {chave estrangeira}

CREATE TABLE TURMAS (

Código CHAR (5) NOT NULL,

Sala Aula CHAR (3) NOT NULL,

Capacidade INTEGER,

Cod_Curso CHAR (5) NOT NULL,

PRIMARY KEY (Código), {este atributo é a chave primária}

FOREIGN KEY (Cod_Curso) REFERENCES CURSOS); {chave estrangeira}

Uma vez que o banco de dados já existe e que as tabelas estão criadas, se precisarmos alterar a estrutura de uma tabela, devemos utilizar o comando alter table. Ou seja, o comando alter table permite modificar a estrutura de uma tabela já definida. Podemos adicionar uma coluna à tabela, modificar uma coluna existente, definir uma chave, dentre outras alterações.

A sintaxe do comando é mostrada a seguir:

ALTER TABLE <nome tabela> add nome coluna tipo dados; | modify nome coluna tipo dados;]

O comando a seguir altera a estrutura da tabela ALUNOS para aumentar o tamanho do campo Email:

ALTER TABLE ALUNOS MODIFY Email VARCHAR (50);

A seguir, o comando é usado para definir uma chave estrangeira na tabela ALUNOS:

ALTER TABLE ALUNOS FOREIGN KEY (Cod_Turma) REFERENCES TURMAS;

Para exclusão definitiva de uma tabela do banco de dados, usamos o comando drop table, o qual remove a tabela e todas as informações sobre ela (inclusive do catálogo do banco de dados).

Exemplo: **DROP TABLE ALUNOS**; {a tabela ALUNOS deixará de existir}

Manipulando Dados com a DML

Para manipularmos os dados nas tabelas do banco de dados, podem ser usados comandos de inserção, exclusão, alteração e consulta de tabelas. Os comandos de atualização de banco de dados com SQL são: INSERT, UPDATE e DELETE, o comando de consulta (recuperação dos dados sem alteração) é o SELECT. Uma vez que as tabelas foram criadas no banco de dados, para sua utilização nas aplicações, é necessário popular essas tabelas com dados, ou inserir dados na tabela, usando o comando insert. O comando insert insere uma linha na tabela e nesse momento observamos se todas as restrições estão sendo respeitadas. Por exemplo, se um campo é obrigatório e no momento da inserção estamos tentando colocar o valor nulo, o comando não será executado e enviará uma mensagem de erro.

A sintaxe do comando insert é a seguinte:

INSERT INTO nome tabela (<lista_de_atributos>)
VALUES <lista_de_valores>;

Por exemplo, inserindo uma linha na tabela de ALUNOS:

INSERT INTO ALUNOS (Matricula, Nome, Endereço, E-mail, Celular, Idade, CPF, Cod_Turma)

VALU S ‘ 568’, ‘Maria do Carmo Silva’, ‘ ua Flores, 56, Centro’, ‘mcs@escola.edu.br’, ‘985869784’, 8, ‘8769 867543’, ‘ 5B’ ;

Observe nesse exemplo que a ordem em que os valores dos dados foram inseridos é a mesma ordem em que os atributos foram listados. Os dados que não são numéricos estão com seus valores colocados entre aspas simples ‘ ’.

Para cada linha de dados que deverá ser inserida no BD, um comando insert deverá ser executado. Geralmente, os SGBDs oferecem interfaces gráficas para inserção de dados, sem a necessidade de digitação desse comando.

Depois que os dados já estão cadastrados no banco de dados e o usuário precisa realizar uma determinada alteração, ele deverá utilizar o comando update. Ou seja, o comando update realiza a alteração de alguma parte dos dados de uma tabela, que atenda a uma condição imposta.

A sintaxe do comando update é a seguinte:

UPDATE nome_tabela
SET alteração
[WHERE condição] {esta parte que define a condição é opcional}

Por exemplo:

Alterar a tabela de professores para corrigir o número do telefone da professora “Carmem”:

UPDATE PROFESSORES {define a tabela que será alterada}
S T Telefone = ‘ 5784563’ {define a coluna com o valor a ser alterado}
WH CPF = ‘ 45 369874 ’; {define a condição que deverá ser validada}

Alterar a tabela de turmas para diminuir a capacidade de todas as turmas em 10%:

UPDATE TURMAS
SET Capacidade = Capacidade – (Capacidade * 0,1);
{como nenhuma condição foi imposta, todas as linhas serão alteradas}

Mudar todos os alunos da turma “ 7C” para a turma “34A”:

UPDATE ALUNOS
SET Cod_Turma = “34A”
WH Cod_Turma = “ 7C”;

De forma semelhante, se os dados já estão cadastrados nas tabelas e pretendemos apagá-los, devemos utilizar o comando delete. Ou seja, o comando delete remove dados cadastrados numa tabela, que atendem a uma determinada condição imposta.

A sintaxe do comando delete é a seguinte:

```
DELETE FROM nome_tabela
[WHERE condição];          {esta parte que define a condição é opcional}
```

Por exemplo:

Remover todos os alunos do banco de dados:

DELETE FROM ALUNOS; {como não foi especificada nenhuma condição, todas as linhas da tabela serão apagadas}

Remover todas as disciplinas com carga horária maior que 60 horas:

```
    DELETE FROM DISCIPLINAS
WHERE CargaHora > 60; {somente algumas linhas serão removidas}
```

Consultas em SQL

Realizar uma consulta ou a recuperação de dados num banco de dados relacional é a tarefa mais importante, devido ao fato de ser a mais executada. Praticamente, todas as ações de manipulação de dados requerem que pelo menos uma consulta seja realizada anteriormente e/ou posteriormente para conferir os dados. Ou seja, se o usuário vai fazer a inserção de um novo dado, antes ele deverá checar se esse dado já existe e depois deverá checar se foi de fato gravado no BD. Esse mesmo tipo de conduta ocorre nos casos de alteração e remoção de dados. Sendo assim, as consultas são realmente as operações mais executadas no BD.

Alguns autores inclusive colocam o comando de recuperação de dados de fora da DML, classificando esse comando como DQL (Data Query Language), onde query é o significado de consulta em inglês. O comando de consulta em SQL é o select.

Uma característica importante sobre uma consulta na SQL é a seguinte: que toda consulta feita na SQL é baseada no modelo e na álgebra relacional e, como na álgebra relacional, sempre retorna uma tabela como resposta. Ou seja, o resultado de um select é sempre uma nova tabela e sobre ela podemos realizar novas consultas.

Outra característica é que para definir uma consulta, basta informarmos o que queremos como resultado e não como fazê-lo, pois a SQL é uma linguagem declarativa e não procedural.

A sintaxe básica do comando select é a seguinte:

SELECT <lista de atributos> FROM <lista de tabelas>

[WHERE <condição>]; {esta parte que define a condição é opcional}

Ao ser executado, o comando select irá selecionar as colunas que deverão compor a tabela resultante, de acordo com a lista de atributos indicada. No FROM deverão ser colocadas as tabelas do banco de dados que devem ser consultadas. Pelo menos, uma tabela é obrigatória em qualquer consulta. No WHERE será indicada uma ou mais condições pela qual os dados serão filtrados.

As condições especificadas devem retornar um dos valores: verdadeiro ou falso. Para cada linha da tabela, o interpretador SQL verifica se atende à condição especificada nessa cláusula e adiciona a linha na resposta caso seja verdadeira a avaliação. É uma cláusula opcional de consulta.

Algumas observações importantes a seguir:

Caso a cláusula WHERE seja omitida, todas as linhas consideram a condição como verdadeira, ou seja, todas as linhas serão retornadas;

A lista de atributos A_1, A_2, A_n pode ser substituída por um asterisco (*) para selecionar todos

Os atributos de todas as tabelas da cláusula FROM;

Para compor as condições que serão impostas os operadores lógicos AND, OR e NOT deverão

Ser utilizados; assim como os operadores de comparação: >, >=, <, <=, = , != (ou <>).

Por exemplo:

Selecione todos os dados cadastrados na tabela de alunos:

```
SELECT * FROM ALUNOS;
```

Selecione todas as disciplinas do Tipo Exatas:

```
SELECT * FROM DISCIPLINAS
```

WH Tipo = “ xatas”;

Selecione os números das salas de aula com capacidade para colocarmos 50 alunos:

```
SELECT SalaAula FROM TURMAS
WHERE Capacidade >= 50;
```

O comando select possui uma grande variedade de cláusulas alternativas, que poderão ser utilizadas a depender do que se deseja consultar. A seguir, algumas das cláusulas mais utilizadas serão apresentadas, mas vale ressaltarmos que dada à variedade de versões da SQL existentes nos dias atuais, para conhecer a maioria, será necessário um estudo mais aprofundado em materiais específicos, tais como os manuais do SGBD específico que estiver trabalhando.

A cláusula DISTINCT é bastante usada e serve para remover as duplicações de linhas no resultado da consulta. Se uma tabela possui na mesma coluna dados repetidos e essa cláusula for utilizada, os dados aparecerão apenas uma vez.

SELECT DISTINCT <atributo>
FROM <nome_tabela>

Por exemplo:

Se quisermos saber quais são os códigos das turmas que aparecem na tabela de alunos:

SELECT DISTINCT Cod_Turma FROM ALUNOS; {neste caso mesmo havendo mais de um aluno nas turmas, cada código só aparecerá uma vez}

Usando o exemplo da Figura, os dados retornados nessa consulta seriam:

Cod_Turma
25B
17C
34A

Como o comando select do modelo relacional é baseado na álgebra relacional, ele permite retornar como resultado um valor calculado de expressões aritméticas. Da mesma forma, cada atributo especificado no comando select também é uma expressão. Sendo assim, ela pode ser manipulada. Essa é uma forma de mostrar resultados calculados, que na verdade não estão armazenados no BD, são os chamados dados derivados. Para os atributos que são do tipo numérico, podemos utilizar qualquer operador matemático, tais como: +, -, *, /. E para os atributos que são do tipo caractere, podemos utilizar concatenação || (união de dados do tipo caractere).

Por exemplo:

Mostrar os nomes e as idades dos alunos no próximo ano: SELECT Nome, (Idade + 1) FROM ALUNOS;

Além dos operadores comuns, ainda existem outras funções que serão mostradas a seguir. As funções de agregação operam sobre um conjunto de linhas e realizam cálculos envolvendo os dados das tabelas.

A função COUNT é um contador de ocorrências ou contador de linhas retornadas no SELECT.

Por exemplo:

1) Conte quantos professores estão cadastrados no BD:

SELECT COUNT (*) FROM PROFESSORES; {como nenhuma condição foi imposta, todos serão contados} 2) Conte quantos alunos são maiores de idade:

SELECT COUNT (*) FROM ALUNOS

WHERE Idade >= 18;

Outras funções muito utilizadas, que realizam comparações e contas matemáticas, possuem a sintaxe semelhante: MAX (retorna o maior valor), MIN (retorna o menor valor), SUM (calcula e mostra a soma de valores), AVG (calcula e mostra a média de valores), dentre outros.

Por exemplo:

1) Mostre a menor e a maior das idades cadastradas na tabela de alunos:

SELECT MIN (Idade), MAX (Idade) FROM ALUNOS;

2) Some todas as capacidades das diferentes turmas, para saber a capacidade total da escola: SELECT SUM (Capacidade) FROM TURMAS;

O operador de comparação [NOT] LIKE permite a definição de padrões de busca, utilizando o símbolo * para definir padrões possíveis:

a Like “c*” – irá exibir valores dos atributos que iniciam com o caractere “c” e depois vem qualquer combinação de valores. b Like “*c” – irá exibir valores dos atributos que terminam com “c”.

c Like “*c*” – irá exibir valores dos atributos que possuem o caractere “c” no meio da palavra. Antes e depois pode vir qualquer sequência de letras.

Vale observarmos que devido às variações da SQL, algumas implementações usam % no lugar de *.

Por exemplo, buscar o nome de todos os professores que começam com a letra B:

SELECT Nome

FROM PROFESSORES

WH Nome LIK “B*”;

A cláusula LIKE além de ser utilizada para encontrarmos determinado padrão, pode ser usada para encontrarmos valores diferentes de um padrão, incluindo o operador NOT. Por exemplo, para buscarmos todos os cursos cujo código não inicia com as letras “PV”:

SELECT * FROM CURSOS WHERE Código NOT LIKE ‘PV*’;

Já a cláusula IS NULL faz a comparação buscando por valores nulos e se colocarmos o operador NOT, buscará por valores que não são nulos.

Por exemplo: buscar dados dos professores que não têm celular cadastrado: SELECT * FROM PROFESSORES WHERE Celular IS NULL;

Por outro lado, se quisermos buscar os dados dos professores que possuem celular (o oposto do exemplo anterior), basta acrescentarmos o operador NOT:

SELECT * FROM PROFESSORES WHERE Celular IS NOT NULL;

Quando a resposta de uma consulta precisa ser buscada em mais de uma tabela do banco de dados, então é necessário especificar quais tabelas serão usadas na consulta SQL. Essa especificação é feita através da cláusula FROM no comando SELECT.

Como a SQL é baseada no modelo relacional, que por sua vez realiza operações da álgebra relacional, a forma de juntar tabelas é através de produto cartesiano das mesmas. O importante é saber que o produto cartesiano faz a combinação uma a uma de todas as linhas das tabelas envolvidas na consulta e nem todas essas linhas combinadas farão sentido para o resultado esperado na consulta. Em outras palavras, para que o resultado faça sentido, precisamos especificar qual é o critério que vamos utilizar entre as tabelas envolvidas, que fará com que as linhas se relacionem. Geralmente, esse critério envolve algum tipo de relação entre chaves estrangeiras.

Na prática, podemos definir quantas tabelas forem necessárias para uma consulta, mas sempre lembrando de colocar a condição que iguala os valores das chaves estrangeiras, para que os dados retornados sejam verdadeiros.

Por exemplo, se quisermos usar duas tabelas, a de CURSOS e a de TURMAS, mostradas a seguir, o resultado é uma tabela que contém relacionamentos entre todos os cursos e todas as turmas. Obviamente que várias linhas geradas no resultado não são úteis, portanto, temos que especificar quais as linhas úteis através da cláusula WHERE.

Se a consulta fosse:

SELECT * FROM CURSOS, TURMAS;

O resultado seria uma tabela como a ilustrada na Figura:

CURSOS				TURMAS			
Codigo	Descricao	Tipo	CPF_Coord	Codigo	SalaAula	Capacidade	Cod_curso
PV005	Pre-vestibular Matutino	Semestral	42025438974	17C	101	50	PV005
PV005	Pre-vestibular Matutino	Semestral	42025438974	25B	205	100	PV007
PV005	Pre-vestibular Matutino	Semestral	42025438974	34A	104	80	SU010
PV007	Pre-vestibular Noturno	Anual	14523698741	17C	101	50	PV005
PV007	Pre-vestibular Noturno	Anual	14523698741	25B	205	100	PV007
PV007	Pre-vestibular Noturno	Anual	14523698741	34A	104	80	SU010
SU010	Supletivo Noturno	Anual	14523698741	17C	101	50	PV005
SU010	Supletivo Noturno	Anual	14523698741	25B	205	100	PV007
SU010	Supletivo Noturno	Anual	14523698741	34A	104	80	SU010

Figura: resultado do produto cartesiano entre as tabelas CURSOS e TURMAS do BD Escola.

Observe que nessa figura as quatro primeiras colunas correspondem às colunas da tabela de cursos e as quatro últimas são as colunas da tabela de turmas. Já as linhas estão combinadas (uma a uma) todas as linhas de ambas as tabelas:

Linha 1 = primeira linha de CURSOS, com a primeira de TURMAS

Linha 2 = primeira linha de CURSOS, com a segunda de TURMAS Linha 3 = primeira linha de CURSOS, com a terceira de TURMAS

Linha 4 = segunda linha de CURSOS, com a primeira de TURMAS

Linha 5 = segunda linha de CURSOS, com a segunda de TURMAS

Linha 6 = segundas linhas de CURSOS, com a terceira de TURMAS

Linha 7 = terceira linha de CURSOS, com a primeira de TURMAS

Linha 8 = terceira linha de CURSOS, com a segunda de TURMAS

Linha 9 = terceira linha de CURSOS, com a terceira de TURMAS

Percebeu como é feita a combinação linha a linha de ambas as tabelas?

Agora observe novamente, na Figura os dados resultantes desse produto cartesiano e perceba que entre as tabelas envolvidas há um valor comum, que é o código da turma (chave estrangeira). As linhas desse resultado que possuem o mesmo valor de código de turma (marcados em verde) significam que os dados se relacionam, já que possuem códigos distintos não fazem sentido (marcados em vermelho).

CURSOS				TURMAS			
Código	Descrição	Tipo	CPF_Coord	Codigo	SalaAula	Capacidade	Cod_curso

PV005 (verde)	Pre-vestibular Matutino	Semestral	42025438974	17C	101	50	PV005 (verde)
PV005 (vermelho)	Pre-vestibular Matutino	Semestral	42025438974	25B	205	100	PV007 (vermelho)
PV005 (vermelho)	Pre-vestibular Matutino	Semestral	42025438974	34A	104	80	SU010 (vermelho)
PV007 (vermelho)	Pre-vestibular Noturno	Anual	14523698741	17C	101	50	PV005 (vermelho)
PV007 (verde)	Pre-vestibular Noturno	Anual	14523698741	25B	205	100	PV007 (verde)
PV007 (vermelho)	Pre-vestibular Noturno	Anual	14523698741	34A	104	80	SU010 (vermelho)
SU010 (vermelho)	Supletivo Noturno	Anual	14523698741	17C	101	50	PV005 (vermelho)
SU010 (vermelho)	Supletivo Noturno	Anual	14523698741	25B	205	100	PV007 (vermelho)
SU010 (verde)	Supletivo Noturno	Anual	14523698741	34A	104	80	SU010 (verde)

Figura: resultado do produto cartesiano entre as tabelas CURSOS e TURMAS do BD Escola, com destaque nos valores de chave estrangeira.

Nesse caso, o correto seria eliminar todas as linhas que não fazem sentido usando uma cláusula WHERE no comando que gerou esse resultado. Assim, o comando ficaria da seguinte forma:

```
SELECT * FROM CURSOS, TURMAS
WHERE CURSOS. Código = TURMAS. Cod_Curso;
```

E então o resultado seria composto apenas por linhas úteis, como mostrado na Figura:

CURSOS				TURMAS			
Codigo	Descricao	Tipo	CPF_Coord	Codigo	SalaAula	Capacidade	Cod_curso
PV005	Pre-vestibular Matutino	Semestral	42025438974	17C	101	50	PV005
PV007	Pre-vestibular Noturno	Anual	14523698741	25B	205	100	PV007
SU010	Supletivo Noturno	Anual	14523698741	34A	104	80	SU010

Figura: Resultado do produto cartesiano entre as tabelas CURSOS e TURMAS do BD Escola, com igualdade na chave estrangeira.

Então vale ressaltarmos mais uma vez, que sempre que uma consulta envolve mais de uma tabela do banco de dados, o comando SELECT deverá conter uma ou mais condições de igualdade envolvendo os valores de chaves estrangeiras.

Como já ressaltamos, existe uma infinidade de variação do comando SELECT. Para conhecer outras opções de cláusulas do comando SELECT, consulte o artigo que trata de SQL avançado, disponível no site:

<Http://www.devmedia.com.br/sql-avancado/28889>. (Acesso em: maio 2019).