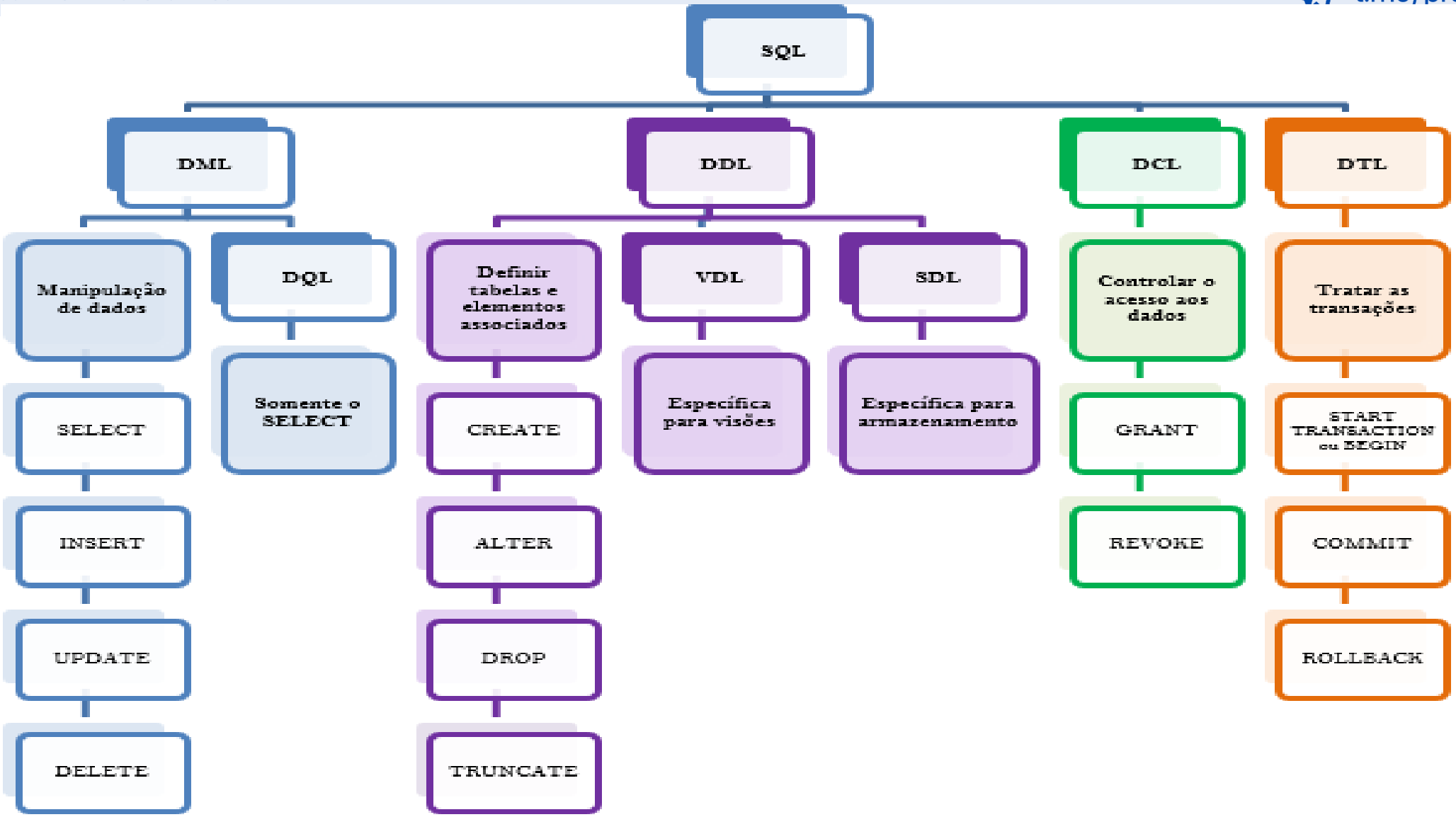
SQL (DML)
(Revisão)

Prof. Ramon Souza

SQL - Subdivisões

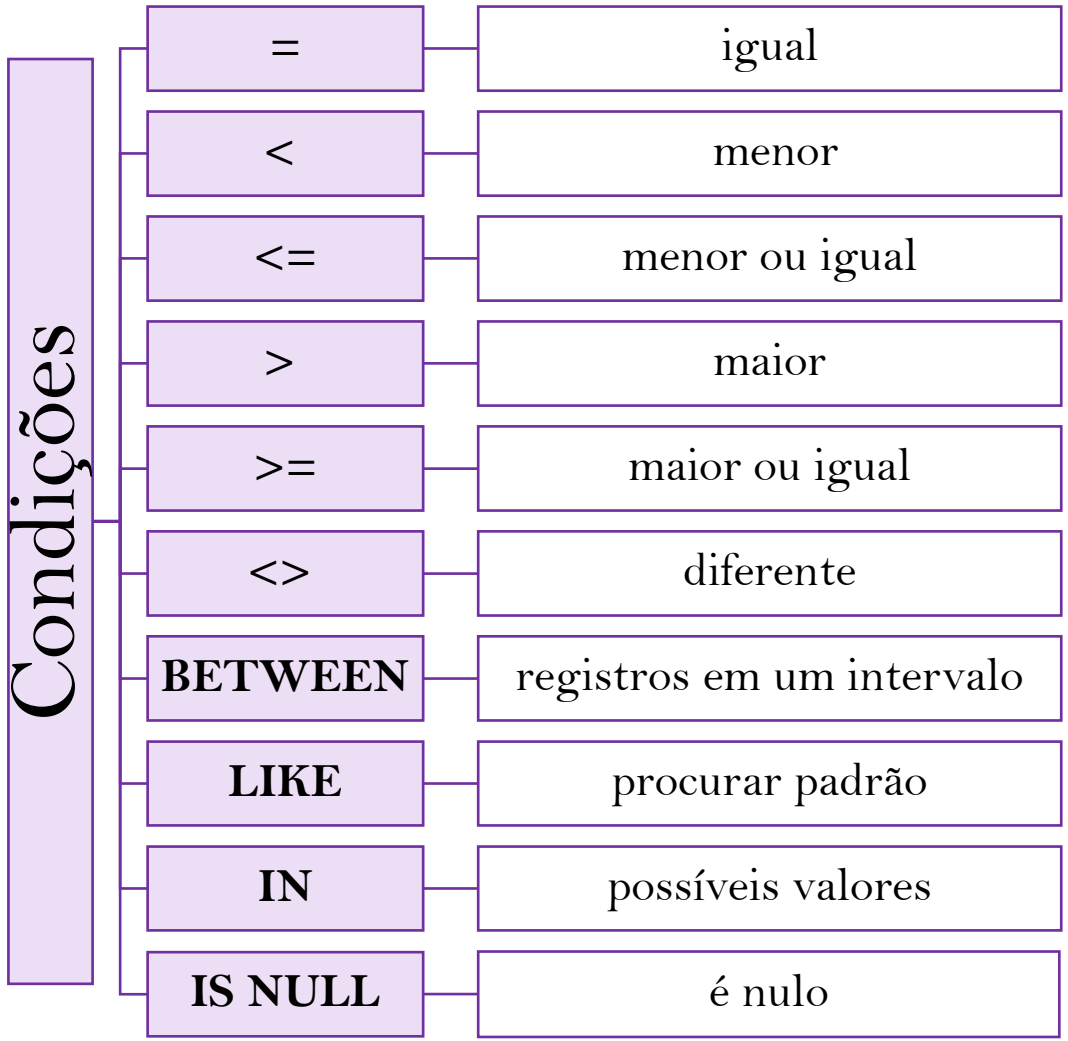


SELECT

A sintaxe básica de uma instrução SELECT é da seguinte forma:

```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE condição;
```

Condições em SELECT



Operador BETWEEN

O operador **BETWEEN** recupera os **registros que estão em um determinado intervalo**. Os valores podem ser números, texto ou datas e tanto os valores de início como o de fim são incluídos.

```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE coluna BETWEEN valor1 AND valor2;
```

Operador IN

O operador **IN** permite especificar **múltiplos valores para uma condição**, isto é, a condição será testada com base na lista de valores indicada

```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE coluna IN (valor1, valor2, ...);
```

Operador LIKE

O operador **LIKE** é utilizado para **procurar um padrão em uma coluna**. Este operador permite a comparação com parte de uma cadeia de caracteres. Este operador é usado em conjunto com dois elemenos curinga (wildcard):

- **%** substitui um número qualquer de 0 ou mais caracteres.
- _ substitui um único caractere.

```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE coluna LIKE padrão;
```

Operador LIKE

Operador LIKE	Procurar padrão em uma coluna
%	Substitui um número qualquer de 0 ou mais caracteres.
_	Substitui um único caractere.
LIKE 'A%'	Qualquer string que inicie com A.
LIKE '%A'	Qualquer string que termine com A.
LIKE '%A%'	Qualquer string que tenha A em qualquer posição.
LIKE 'A_'	String de dois caracteres que tenha a primeira letra A e o segundo caractere seja qualquer outro.
LIKE '_A'	String de dois caracteres cujo primeiro caractere seja qualquer um e a última letra seja a letra A.
LIKE '_A_'	String de três caracteres cuja segunda letra seja A, independentemente do primeiro ou do último caractere.
LIKE '%A_'	Qualquer string que tenha a letra A na penúltima posição e a última seja qualquer outro caractere.
LIKE '_A%'	Qualquer string que tenha a letra A na segunda posição e o primeiro caractere seja qualquer outro caractere.
LIKE '_ _ _'	Qualquer string com exatamente três caracteres.
LIKE '_ _ _%'	Qualquer string com pelo menos três caracteres.
LIKE '%"%'	Qualquer string que tenha o caractere " em qualquer posição.

Operador IS NULL

O operador **IS NULL** **testa se um valor é NULO**, isto é, se o atributo não possui um valor específico.

```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE coluna IS NULL;
```

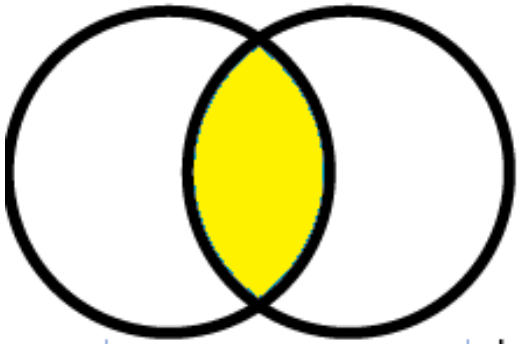
Caso queira os não nulos, então basta usar **IS NOT NULL.**

```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE coluna IS NOT NULL;
```

Operador AND

Exibe os registros em que **todas as condições** são verdadeiras.

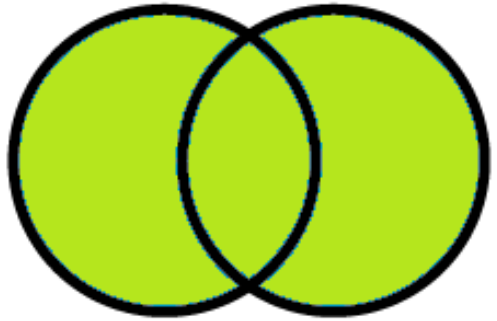
SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição1 **AND** condição2 **AND** condição3 ...;



Operador OR

Exibe os registros em que **pelo menos uma condição** é verdadeira.

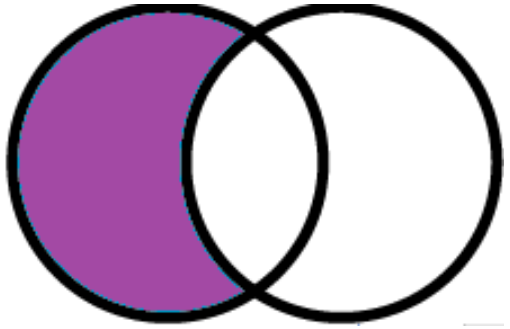
```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE condição1 OR condição2 OR condição3 ...;
```



Operador NOT

Exibe os registros que **não satisfazem** uma condição.

```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE NOT condição;
```



Definição de alias (apelido)

SELECT coluna1 **AS** nova, coluna2 **FROM** nome_da_tabela **WHERE** condição;

OU

SELECT coluna1 nova, coluna2 **FROM** nome_da_tabela **WHERE** condição;

SELECT coluna1, coluna2... **FROM** nome_da_tabela **AS** nova **WHERE** condição;

OU

SELECT coluna1, coluna2... **FROM** nome_da_tabela nova **WHERE** condição;

Ordenação com SELECT

A ordem padrão está em ordem crescente de valores. A palavra-chave **DESC** pode ser usada para ordenar os resultados em **ordem decrescente** de valores. A palavra-chave **ASC** pode ser usada para especificar a **ordem crescente** explicitamente.

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição **ORDER BY** coluna **ASC/DESC**;

Ordenação com SELECT

É importante destacar que é possível **ordenar por mais de uma coluna**, bastando indicar as colunas e a ordem desejada.

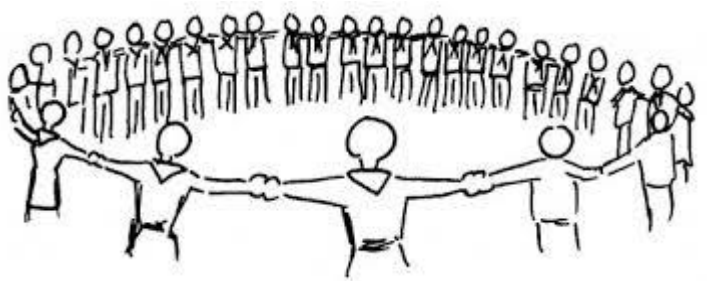
```
SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE condição
ORDER BY coluna1, coluna2 ASC;

SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE condição
ORDER BY coluna1 ASC coluna2 DESC;
```

Funções de agregação

As **funções de agregação** são usadas para **resumir informações de várias tuplas em uma síntese de tupla única**. Existem diversas funções de agregação embutidas no SQL: **COUNT, SUM, MAX, MIN e AVG**.

.

SELECT FUNCAO(coluna1) **FROM** nome_da_tabela **WHERE** condição; em que **FUNCAO** é qualquer uma das funções de agregação.



Funções de agregação

FUNÇÃO	RETORNO
MIN	Menor valor de uma coluna.
MAX	Maior valor de uma coluna.
COUNT	Número de linhas que atendem a um critério.
AVG	Média dos valores de uma coluna numérica.
SUM	Soma dos valores de uma coluna numérica.

Variações de COUNT

A cláusula **COUNT** pode ser usada com o nome da coluna, * ou com 1:

- **COUNT(nome_da_coluna)**: retorna o número de linhas excluindo-se da contagem as linhas que possuem nulo para a coluna desejada.

- **COUNT(*) ou COUNT(1)**: retorna o número total de linhas, independentemente de valores nulos registrados para qualquer campo.

Variações de SUM

A cláusula **SUM** pode ser usada com o nome da coluna ou com um número indicativo da quantidade a ser somada:

- **SUM(nome_da_coluna)**: retorna o somatório dos valores presentes em nome_da_coluna.

- **SUM(1)**: retorna um somatório, sendo somado 1 para cada registro encontrado. Resultado similar a COUNT(*) ou COUNT(1).

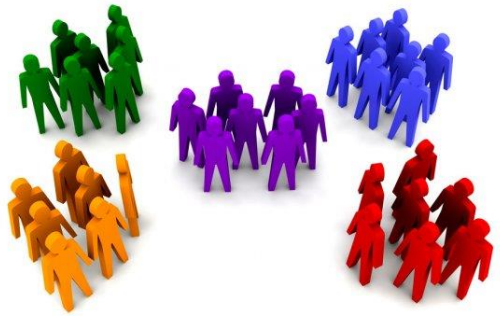
- **SUM(2)**: retorna um somatório, sendo somado 2 para cada registro encontrado.

- **SUM(N)**: retorna um somatório, sendo somado N para cada registro encontrado.

Agrupamentos

A linguagem SQL tem uma cláusula **GROUP BY** para **aplicar agrupamentos**. Esta cláusula especifica os atributos de agrupamento, que também devem aparecer na cláusula SELECT, de modo que o valor resultante da aplicação de cada função de agregação a um grupo de tuplas apareça junto com o valor do(s) atributo(s) de agrupamento.

SELECT colunas **FROM** nome_da_tabela **WHERE** condição **GROUP BY** coluna;



Condição em um agrupamento

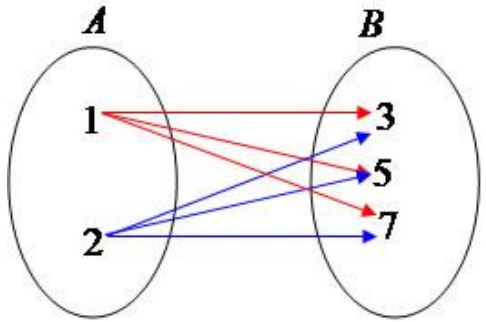
A cláusula **HAVING** pode ser usada para **definir uma condição para um agrupamento** com GROUP BY.

SELECT colunas **FROM** nome_da_tabela **WHERE** condição **GROUP BY** coluna **HAVING** condição;

Produto cartesiano

O **Produto Cartesiano** seleciona **todos os pares de linhas das duas relações de entrada** (independentemente de ter ou não os mesmos valores em atributos comuns). A nova relação possui todos os atributos que compõem cada uma das relações que fazem parte da operação.

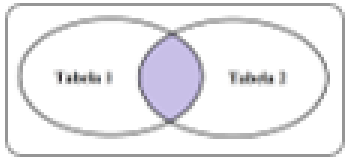
SELECT tabela1.coluna1, tabela2.coluna2., ... **FROM** tabela1, tabela2 **WHERE** condição;



Junções (Joins)

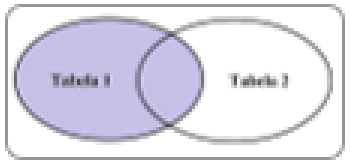
As **tabelas de junção**, que **combinam duas ou mais tabelas baseando-se em colunas relacionadas**. As tabelas de junção são especificadas segundo a cláusula **JOIN**.

Junções (Joins)



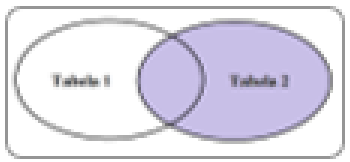
INNER JOIN (ou simplemente JOIN)

- Retorna somente os registros que possuem valores relacionados em ambas as tabelas, isto é, as intersecções.



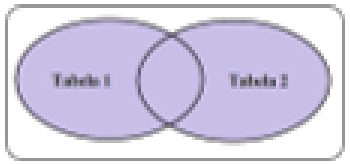
LEFT JOIN (ou LEFT OUTER JOIN)

- Retorna todos os registros da tabela da esquerda, e os registros relacionados da tabela da direita.
- Preenche campos não relacionados na tabela da direita com NULL.



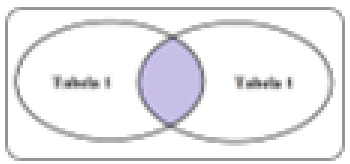
RIGHT JOIN (ou RIGHT OUTER JOIN)

- Retorna todos os registros da tabela da direita, e os registros relacionados da tabela da esquerda.
- Preenche campos não relacionados na tabela da esquerda com NULL



FULL OUTER JOIN

- Retorna todos os registros, independente de relação.
- Preenche campos não relacionados em qualquer das tabelas com NULL.



SELF JOIN

- União de uma tabela com ela mesma.

Junções (Joins)

A sintaxe básica para consultas com junções é:

```
SELECT colunas FROM tabela1 JOIN tabela2 ON tabela1.coluna = tabela2.coluna;
```

Para o INNER JOIN, se as colunas em ambas as tabelas tiverem o mesmo nome, podemos usar a cláusula USING:

```
SELECT colunas FROM tabela1 INNER JOIN tabela2 USING (coluna);
```

Operadores de conjuntos

OPERADOR	RETORNO
UNION	Todas as linhas pertencentes as consultas envolvidas, sem as repetições.
UNION ALL	Todas as linhas pertencentes as consultas envolvidas, incluindo as repetições.
INTERSECT	Linhas que estão tanto na primeira quanto na segunda consulta. Intersecção, sem repetições.
EXCEPT	Linhas que estão na primeira, mas não estão na segunda, sem repetições.

Consultas aninhadas

Em alguns casos, precisamos realizar uma consulta que é comparada com o resultado de outra consulta. Aqui teremos o uso de uma **consulta dentro de outra consulta** ou **consulta aninhada**. A consulta que é realizada dentro de outra é chamada subconsulta.

Consultas aninhadas

As subconsultas podem ser comparadas com a consulta externa com o uso de operadores IN (ou NOT IN) ou EXISTS (ou NOT EXISTS), além dos operadores básicos =, <, <=, >, >=, <>.

EXISTS

A cláusula **EXISTS** faz uma **verificação se existe algum resultado para a subconsulta informada**. **Caso haja**, o **resultado da consulta principal é exibido**. É muito comum sua utilização quando se deseja trazer resultados onde um valor específico existe dentro de outra tabela.

SELECT colunas **FROM** tabela **WHERE EXISTS (SELECT** colunas **FROM** tabela **WHERE** condicão);

DELETE

A instrução básica para **deletar registros existentes de uma tabela** é a instrução **DELETE**.

```
DELETE FROM nome_da_tabela WHERE condição;
```

A instrução básica para **atualizar os registros de uma tabela** é a instrução **UPDATE**.

```
UPDATE nome_da_tabela SET coluna1 = valor1, coluna2 = valor2 ... WHERE condição;
```

INSERT INTO

A instrução básica para **inserir novos registros em uma tabela** é a instrução **INSERT INTO**.

INSERT INTO nome_da_tabela (coluna1, coluna2, coluna3, ...) **VALUES** (valor1, valor2, valor3, ...);

OU

INSERT INTO nome_da_tabela **VALUES** (valor1, valor2, valor3, ...);

SQL (DML)
(Revisão)

Prof. Ramon Souza