# Banco de Dados Relacional

# SELECT (MySQL) – parte 2

## 1 Revisando

### 1.1 Filtragem - WHERE

Restringe os resultados de uma consulta com base em condições específicas.

cláusula **WHERE** - filtra resultados com base em condições específicas.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

### 1.2 Agregação – GROUP BY

A agregação em um SELECT refere-se à combinação de múltiplas linhas de dados em um único valor com base em alguma operação de agregação.

Cláusula **GROUP BY** - divide os dados em grupos com base em valores comuns em uma ou mais colunas. Isso permite que você agregue os dados em cada grupo separadamente.

## 2. Cláusula Nova: Agregação - HAVING

Cláusula **HAVING** - utilizada em conjunto com a instrução SELECT para aplicar condições a grupos de linhas, em vez de linhas individuais. Geralmente, é usado com a cláusula GROUP BY para filtrar grupos com base em valores agregados.

**SELECT nome_da_coluna(s) FROM nome_da_tabela GROUP BY nome_da_coluna HAVING condição;**

Enquanto a cláusula WHERE é usada para filtrar linhas antes do agrupamento ocorrer, a cláusula HAVING é usada para filtrar grupos depois do agrupamento ter sido feito. Em essência, HAVING é para grupos o que WHERE é para registros.

```
SELECT id_cliente, COUNT(id_pedido) AS total_pedidos
FROM pedidos
GROUP BY id_cliente
HAVING COUNT(id_pedido) > 5;
```

## 3. Junção (Inner Join)

Em SQL, junção (join) ou (inner Join, traduzindo ao pé da letra junção interna) é uma operação que combina linhas de duas ou mais tabelas com base em uma condição específica, criando um resultado que contém colunas de ambas as tabelas.

```
SELECT colunas
FROM tabela1
INNER JOIN tabela2 ON
tabela1.coluna_chave = tabela2.coluna_chave;
```

**Ou usando Alias:**

```
SELECT colunas
FROM tabela1 t1
INNER JOIN tabela2 t2 ON
t1.coluna_chave = t2.coluna_chave;
```

**Exemplos:**

```
CREATE TABLE alunos (
    id_aluno INT PRIMARY KEY,
    nome_aluno VARCHAR(100),
    id_curso INT,
    FOREIGN KEY (id_curso) REFERENCES cursos(id_curso)
);

CREATE TABLE cursos (
    id_curso INT PRIMARY KEY,
```

```
    nome_curso VARCHAR(100)
);

INSERT INTO alunos (id_aluno, nome_aluno, id_curso)
VALUES
    (1, 'Alice', 1),
    (2, 'Bob', 2),
    (3, 'Carol', 1),
    (4, 'David', 3);

INSERT INTO cursos (id_curso, nome_curso)
VALUES
    (1, 'Matemática'),
    (2, 'História'),
    (3, 'Ciências');

SELECT alunos.nome_aluno, cursos.nome_curso
FROM alunos
INNER JOIN cursos ON alunos.id_curso = cursos.id_curso;
```

```
+----------------+----------------+
| nome_aluno | nome_curso |
+----------------+----------------+
| Alice          | Matemática |
| Bob            | História       |
| Carol          | Matemática |
| David          | Ciências       |
+----------------+----------------+
```

---

## RIGHT JOIN

Um RIGHT JOIN retorna todas as linhas da tabela à direita (tabela do lado direito da junção) e as linhas correspondentes da tabela à esquerda (tabela do lado esquerdo da junção). Se não houver correspondência, as colunas da tabela à esquerda terão valores nulos. Exemplo:

```
CREATE TABLE alunos (
    id_aluno INT PRIMARY KEY,
    nome VARCHAR(100)
);

CREATE TABLE notas (
    id_aluno INT,
    disciplina VARCHAR(100),
    nota DECIMAL(5,2)
);

INSERT INTO alunos (id_aluno, nome) VALUES
    (1, 'Alice'),
    (2, 'Bob'),
    (3, 'Carol');

INSERT INTO notas (id_aluno, disciplina, nota) VALUES
    (1, 'Matemática', 8.5),
    (2, 'História', 7.0);
```

Uso do RIGHT JOIN para listar todas as notas dos alunos, incluindo aqueles que não têm notas registradas:

```
SELECT alunos.nome AS nome_aluno, notas.disciplina, notas.nota
FROM alunos
RIGHT JOIN notas ON alunos.id_aluno = notas.id_aluno;
```

Resultado:

```
+-----------------+----------------+--------+
| nome_aluno  | disciplina      |   nota |
+-----------------+----------------+--------+
| Alice             | Matemática |     8.5 |
| Bob               | História       |     7.0 |
| Carol             | NULL           | NULL |
+-----------------+----------------+--------+
```

```
SELECT alunos.nome AS nome_aluno, notas.disciplina, notas.nota
FROM alunos
RIGHT JOIN notas ON alunos.id_aluno = notas.id_aluno
WHERE notas.nota IS NOT NULL;
```

Resultado:

```
+-----------------+----------------+--------+
| nome_aluno  | disciplina      |   nota |
+-----------------+----------------+--------+
| Alice             | Matemática |     8.5 |
| Bob               | História       |     7.0 |
```

```
+-----------------+----------------+--------+
```

---

## CROSS JOIN

CROSS JOIN é usado para combinar cada linha de uma tabela com cada linha de outra tabela, criando um produto cartesiano entre elas. Suponha que temos duas tabelas: "cores" e "tamanhos". A tabela "cores" possui três registros: "vermelho", "azul" e "verde". A tabela "tamanhos" possui dois registros: "pequeno" e "grande". Um CROSS JOIN entre essas tabelas irá gerar todas as combinações possíveis de cores e tamanhos.

sql

```
CREATE TABLE cores (
    cor VARCHAR(20)
);

CREATE TABLE tamanhos (
    tamanho VARCHAR(20)
);

INSERT INTO cores (cor) VALUES
    ('vermelho'),
    ('azul'),
    ('verde');

INSERT INTO tamanhos (tamanho) VALUES
    ('pequeno'),
    ('grande');
```

```
SELECT cores.cor, tamanhos.tamanho
FROM cores
CROSS JOIN tamanhos;
```

Resultado:

```
+------------+-------------+
| cor        | tamanho     |
+------------+-------------+
| vermelho   | pequeno     |
| vermelho   | granden     |
| azul       | pequeno     |
| azul       | grande      |
| verde      | pequeno     |
| verde      | grande      |
+------------+-------------+
```

---

## FULL JOIN

No MySQL, a cláusula FULL JOIN não é diretamente suportada. No entanto, se pode simular um FULL JOIN usando UNION com LEFT JOIN e RIGHT JOIN.
Suponha duas tabelas: "tabela_a" e "tabela_b":

```
CREATE TABLE tabela_a (
    id INT PRIMARY KEY,
    nome VARCHAR(100)
```

```
);

CREATE TABLE tabela_b (
    id INT PRIMARY KEY,
    nome VARCHAR(100)
);

INSERT INTO tabela_a (id, nome) VALUES
    (1, 'Alice'),
    (2, 'Bob'),
    (3, 'Carol');

INSERT INTO tabela_b (id, nome) VALUES
    (2, 'David'),
    (3, 'Eve'),
    (4, 'Frank');

SELECT a.id, a.nome AS nome_tabela_a, b.nome AS nome_tabela_b
FROM tabela_a a
LEFT JOIN tabela_b b ON a.id = b.id
UNION
SELECT b.id, a.nome AS nome_tabela_a, b.nome AS nome_tabela_b
FROM tabela_b b
LEFT JOIN tabela_a a ON a.id = b.id
WHERE a.id IS NULL;
```

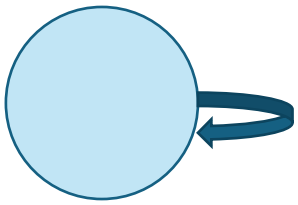
Neste exemplo, a primeira parte da consulta usa um LEFT JOIN entre "tabela_a" e "tabela_b", enquanto a segunda parte usa um LEFT JOIN entre "tabela_b" e "tabela_a". A cláusula WHERE a.id IS NULL é usada na segunda parte para filtrar apenas as linhas onde "tabela_a" não tem correspondência com "tabela_b".

O resultado será algo como:

```
+----+--------------------+--------------------+
| id   | nome_tabela_a | nome_tabela_b |
+----+--------------------+--------------------+
|  1  | Alice                | NULL               |
|  2  | Bob                  | David              |
|  3  | Carol                | Eve                |
|  4  | NULL                 | Frank              |
+----+--------------------+--------------------+
```

---

## SELF JOIN – Auto Junção



Auto junção é uma Junção em que uma tabela é combinada consigo mesma. Isso é útil quando você precisa comparar linhas dentro da mesma tabela. Por exemplo, um relatório de nome de pessoa e o nome do respectivo conjugê.

```
CREATE TABLE pessoa (
    id_pessoa INT PRIMARY KEY,
```

```
    nome VARCHAR(100),
    id_conjuge INT,
    FOREIGN KEY (id_conjuge) REFERENCES pessoa(id_pessoa)
);

INSERT INTO pessoa (id_pessoa, nome, id_conjuge)
VALUES
    (1, 'Alice', 2),
    (2, 'Bob', 1),
    (3, 'Carol', 4),
    (4, 'David', 3);

SELECT p1.nome AS pessoa, p2.nome AS conjuge
FROM pessoa p1
LEFT JOIN pessoa p2 ON p1.id_conjuge = p2.id_pessoa;
```

```
+----------+------------+
| pessoa |  conjuge |
+----------+------------+
| Alice      | Bob       |
| Bob       | Alice      |
| Carol     | David    |
| David    | Carol     |
+----------+------------+
```

**Considerando este insert (sem o David):**

```
INSERT INTO pessoa (id_pessoa, nome, id_conjuge)
VALUES
    (1, 'Alice', 2),
```

```
    (2, 'Bob', 1),
    (3, 'Carol', 4),

SELECT p1.nome AS pessoa, p2.nome AS conjuge
FROM pessoa p1
LEFT JOIN pessoa p2 ON p1.id_conjuge = p2.id_pessoa;

+----------+------------+
| pessoa | conjuge |
+-----------+-----------+
| Alice     | Bob        |
| Bob       | Alice      |
| Carol     | NULL       |

SELECT p1.nome AS pessoa, p2.nome AS conjuge
FROM pessoa p1
RIGHT JOIN pessoa p2 ON p1.id_conjuge = p2.id_pessoa;


pessoa | conjuge
-------|--------
Alice  | Bob
Bob    | Alice
```

## 4 Revisão

Essas são as novas palavras reservadas no MySQL. Você lembra a utilidade de cada uma? Caso não se lembre, revise!

1. **HAVING**

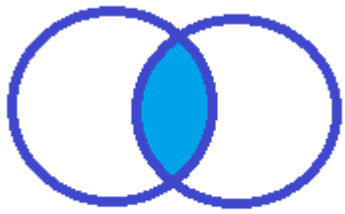
2. **ON**

3. **IS NOT NULL**

4. **IS NULL**

---

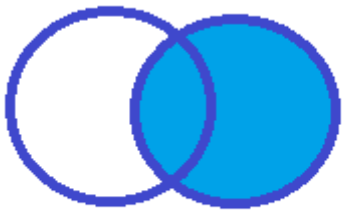
### 1. INNER JOIN - Junção



```
SELECT *
FROM A
INNER JOIN B ON
A.KEY = B.KEY;
```

---

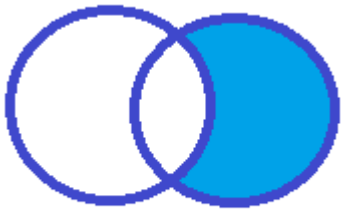
## 2. RIGHT JOIN – Junção à direita



```
SELECT *
FROM A
RIGHT JOIN B ON
A.KEY = B.KEY;
```

---

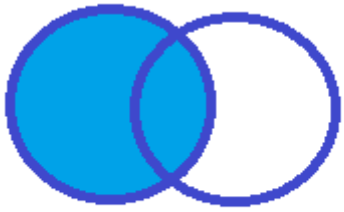
## 3. RIGHT JOIN NULL - Junção à direita nula



```
SELECT *
FROM A
RIGHT JOIN B ON
A.KEY WHERE B.KEY IS NULL;
```

---

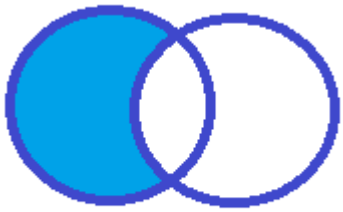
## 4. LEFT JOIN - Junção à esquerda



```
SELECT *
FROM A
LEFT JOIN B ON
A.KEY = B.KEY;
```

---

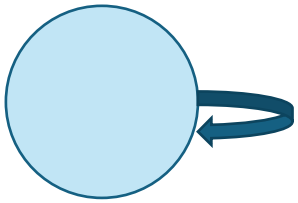
## 5. LEFT JOIN NULL - Junção à esquerda nula



```
SELECT *
FROM A
LEFT JOIN B ON
A.KEY WHERE B.KEY IS NULL;
```

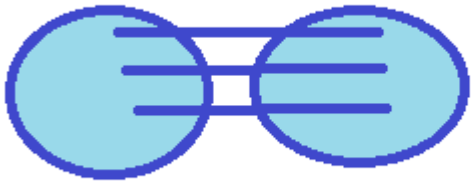
---

## 6. SELF JOIN – Auto Junção



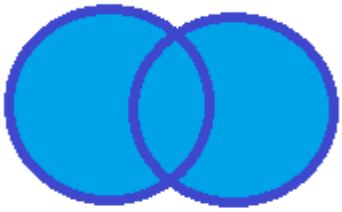
---

## 7. CROSS JOIN - Junção cruzada



---

## 8. FULL JOIN - Junção completa



---

## 9. FULL JOIN NULL - Junção completa nula

---