UVV

Melhor Universidade Particular do Brasil *(MEC)*

THE Times Higher Education

Pacto Nacional pela Primeira Infância

universidades empreendedoras

QS WORLD UNIVERSITY RANKINGS

AD Scientific Index

Prêmio Sebrae de Educação Empreendedora

CENTELHA

QS

THE IMPACT RANKINGS

DESIGN E DESENVOLVIMENTO DE BANCO DE DADOS

UNIDADE VI - DQL’s

Prof. Msc. Gustavo Nunes Rocha

Definições Preliminares

A instrução SELECT é uma das mais cruciais e frequentemente usadas no SQL, permitindo recuperar dados de um banco de dados MySQL. Ela permite especificar quais colunas você deseja recuperar e aplicar várias opções de filtragem e ordenação ao conjunto de resultados. Este guia abrangente abordará tudo o que você precisa saber sobre o uso da instrução SELECT no MySQL.

Sintaxe básica

A sintaxe básica da instrução SELECT é a seguinte:

SELECT coluna1, coluna2, ...

FROM nome_da_tabela

coluna1, coluna2, ...: As colunas das quais você deseja recuperar dados.

nome_da_tabela: O nome da tabela da qual você deseja recuperar dados.

Sintaxe básica

Recuperando Todas as Colunas: Para recuperar todas as colunas de uma tabela, você pode usar o asterisco *:

SELECT * FROM nome_da_tabela;

Sintaxe básica

Recuperando Colunas Específicas: Para recuperar colunas específicas, liste-as após a palavra-chave SELECT:

SELECT coluna1, coluna2 FROM nome_da_tabela;

Sintaxe básica: DISTINCT

A cláusula DISTINCT é utilizada em consultas SQL para retornar apenas valores únicos de uma coluna específica de uma tabela. Ela elimina duplicatas dos resultados da consulta, garantindo que cada valor retornado seja único. Esta cláusula é frequentemente utilizada em conjunto com a instrução SELECT para obter uma lista de valores distintos de uma determinada coluna.

SELECT DISTINCT coluna1 FROM nome_da_tabela;

Sintaxe básica: DISTINCT

A cláusula DISTINCT é utilizada em consultas SQL para retornar apenas valores únicos de uma coluna específica de uma tabela. Ela elimina duplicatas dos resultados da consulta, garantindo que cada valor retornado seja único. Esta cláusula é frequentemente utilizada em conjunto com a instrução SELECT para obter uma lista de valores distintos de uma determinada coluna.

SELECT DISTINCT categoria FROM produtos;

Funções de agregação

As funções de agregação realizam cálculos em um conjunto de valores e retornam um único valor. Exemplos incluem:

- COUNT(): Retorna o número de linhas em um conjunto de resultados.
- SUM(): Retorna a soma dos valores em uma coluna.
- AVG(): Retorna a média dos valores em uma coluna.
- MIN(): Retorna o valor mínimo em uma coluna.
- MAX(): Retorna o valor máximo em uma coluna.

Funções de agregação

- COUNT( ): Retorna o número de linhas em um conjunto de resultados. Pode ser usado para contar todas as linhas ou contar as linhas que atendem a uma condição específica. Exemplos:
- SELECT COUNT(*) FROM orders; -- Conta todas as linhas na tabela de pedidos
- SELECT COUNT(*) FROM customers WHERE age > 30; -- Conta os clientes com idade acima de 30

Funções de agregação

- SUM( ): Retorna a soma dos valores em uma coluna numérica.
- SELECT SUM(quantity) FROM order_items; -- Soma a quantidade de itens pedidos
- SELECT product_id, SUM(quantity * price) FROM order_items; -- Calcula o total de vendas de produtos
- SELECT SUM(salary) FROM employees; -- Calcula a soma dos salários de todos os funcionários

Funções de agregação

- AVG( ): Retorna a média dos valores em uma coluna numérica. Exemplo:
- SELECT AVG(price) FROM products; -- Calcula a média dos preços dos produtos
- SELECT AVG(age) FROM customers; -- Calcula a média de idade dos clientes
- SELECT AVG(score) FROM exam_results; -- Calcula a média de pontuação nas avaliações

Funções de agregação

- MIN( ): Retorna o valor mínimo em uma coluna. Exemplo:
- SELECT MIN(salary) FROM employees; -- Retorna o salário mínimo entre os funcionários
- SELECT MIN(order_date) FROM orders; -- Encontra a data mais antiga de um conjunto de registros
- SELECT MIN(price) FROM products; -- Encontra o menor preço entre os produtos

Funções de agregação

- MAX( ): Retorna o valor máximo em uma coluna. Exemplo:
- SELECT MAX(score) FROM exam_results; -- Retorna a pontuação máxima nas avaliações
- SELECT MAX(order_date) FROM orders; -- Encontra a data mais recente de um conjunto de registros
- SELECT MAX(salary) FROM employees; -- Encontra o maior salário entre os funcionários

Operador Where

Filtrando Dados com a Cláusula WHERE: Você pode usar a cláusula WHERE para filtrar linhas com base em condições específicas:

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


Operador Where

Filtrando Dados com a Cláusula WHERE: Você pode usar a cláusula WHERE para filtrar linhas com base em condições específicas:

```
SELECT nome, sobrenome
FROM funcionarios
WHERE departamento = 'Vendas';
```


Operador Where – valores nulos

Para filtrar valores nulos em uma consulta SQL, você pode usar a cláusula WHERE em conjunto com a condição IS NULL ou IS NOT NULL, dependendo se você deseja selecionar os registros que têm valores nulos ou não têm valores nulos, respectivamente.

```
SELECT nome, sobrenome
FROM funcionarios
WHERE salario IS NOT NULL;
```


Funções Lógicas

O MySQL, assim como muitos outros sistemas de gerenciamento de bancos de dados relacionais (RDBMS), oferece uma variedade de funções lógicas que podem ser usadas em consultas SQL para manipular e analisar dados. Aqui estão algumas das funções lógicas comuns no MySQL:

Funções Lógicas

Função	Descrição	Exemplo
IF()	Retorna um valor com base em uma condição verdadeira/falsa	**IF(coluna1 > 10, 'Maior que 10', 'Menor ou igual a 10')**
CASE	Avalia várias condições e retorna um valor correspondente	**CASE WHEN coluna1 > 10 THEN 'Maior que 10' ELSE 'Menor que 10' END**
AND	Operador lógico que combina duas condições com "E"	**coluna1 > 10 AND coluna2 < 20**
OR	Operador lógico que combina duas condições com "OU"	**coluna1 = 10 OR coluna2 = 20**
NOT	Operador lógico que nega uma condição	**NOT(coluna1 = 10)**
IN()	Verifica se um valor está em um conjunto de valores	**coluna1 IN (1, 2, 3)**
NOT IN()	Verifica se um valor não está em um conjunto de valores	**coluna1 NOT IN (1, 2, 3)**
BETWEEN	Verifica se um valor está dentro de um intervalo	**coluna1 BETWEEN 10 AND 20**
EXISTS	Verifica se uma subconsulta retorna algum resultado	**EXISTS (SELECT * FROM tabela2 WHERE tabela1.id = tabela2.id)**
LIKE	Verifica se um valor corresponde a um padrão com curingas	**coluna1 LIKE 'abc%'**

Funções Lógicas

-- Operador AND:

```
SELECT * FROM produtos WHERE preco > 10 AND quantidade < 20;
```

-- Operador OR:

```
SELECT * FROM clientes WHERE nome = 'João' OR idade > 30;
```

-- Operador NOT:

```
SELECT * FROM funcionarios WHERE NOT cargo = 'Gerente';
```


Funções Lógicas

-- Função IN():

```
SELECT * FROM pedidos WHERE status IN ('Em andamento', 'Atrasado');
```

-- Função NOT IN():

```
SELECT * FROM produtos WHERE categoria NOT IN ('Eletrônicos',
'Roupas');
```

-- Função BETWEEN:

```
SELECT * FROM vendas WHERE data BETWEEN '2024-01-01' AND '2024-
03-31';
```


Funções Lógicas

-- Função EXISTS:

```
SELECT * FROM clientes WHERE EXISTS (SELECT * FROM pedidos
WHERE clientes.id = pedidos.cliente_id);
```

-- Função LIKE:

```
SELECT * FROM produtos WHERE nome LIKE 'Camiseta%';
```


MySQL Wildcards

Em MySQL, os wildcards são caracteres especiais que você pode usar em consultas para fazer correspondência parcial em cadeias de caracteres. Os wildcards mais comuns são **%** e _ .

% : Representa zero, um ou vários caracteres. Pode ser usado no início, no meio ou no final de uma cadeia de caracteres.

_ : Representa um único caractere. Pode ser usado no lugar de um único caractere em uma cadeia de caracteres.

MySQL Wildcards

Usando % para correspondência parcial:

Para encontrar todos os clientes cujo nome começa com "Jo":

```
SELECT * FROM clientes WHERE nome LIKE 'Jo%';
```

Para encontrar todos os clientes cujo nome termina com "son":

```
SELECT * FROM clientes WHERE nome LIKE '%son';
```

Para encontrar todos os clientes cujo nome contém "ohn":

```
SELECT * FROM clientes WHERE nome LIKE '%ohn%';
```


MySQL Wildcards

Usando _ para corresponder a um único caractere:

Encontrar todos os clientes cujo nome tem exatamente cinco letras e começa com "J":

```
SELECT * FROM clientes WHERE nome LIKE 'J____';
```

Encontrar todos os clientes cujo nome tem três letras e termina com "y":

```
SELECT * FROM clientes WHERE nome LIKE '___y';
```

Encontrar todos os clientes cujo nome tem exatamente 6 letras e tem um "r" na terceira posição:

```
SELECT * FROM clientes WHERE nome LIKE '__r___';
```

EXERCÍCIOS

BDEX1_HOSPITAL:

1. Quais são os nomes e datas de nascimento de todos os pacientes cadastrados?
2. Qual é o número total de médicos cadastrados?
3. Qual é a idade média dos pacientes cadastrados?

EXERCÍCIOS

BDEX2_PASSAGENS_AEREAS:

1. Quais são os nomes e cidades de todos os aeroportos cadastrados?
2. Quantos voos estão cadastrados para cada companhia aérea?
3. Qual é a capacidade média das aeronaves cadastradas?

EXERCÍCIOS

BDEX3_REDESOCIAL:

1. Quais são os nomes e sobrenomes de todos os usuários cadastrados?
2. Quantos grupos existem na rede social?
3. Qual é o número médio de comentários por postagem?

EXERCÍCIOS

BDEX4_ESTOQUE:

1. Quais são os nomes e preços de todos os produtos cadastrados?
2. Quantos produtos diferentes cada fornecedor fornece?
3. Qual é o número total de produtos em estoque?

EXERCÍCIOS

BDEX5_EVENTOS:

1. Quais são os nomes e datas dos eventos cadastrados?
2. Quantas atividades estão planejadas para cada evento?
3. Qual é o número médio de participantes por evento?

EXERCÍCIOS

BDEX6_CINEMA:

1. Quais são os nomes e capacidades de todas as salas de cinema cadastradas?
2. Quantos espectadores frequentaram cada sessão de cinema?
3. Qual é o horário médio de início das sessões de cinema?

EXERCÍCIOS

BDEX7_TRANSACOESBANCARIAS:

1. Quais são os nomes e CPFs de todos os usuários cadastrados?
2. Qual é o total de transações realizadas por cada usuário?
3. Qual é o valor médio das transações bancárias?

Operador Group by

O operador Group By, é utilizado quando queremos agrupar valores de uma coluna. Grande parte de relatórios que vemos com totalizadores de alguma informação está utilizando ele. Um ponto importante sobre o uso deste operador; sempre devemos utilizá-lo junto com alguma função de agregação.

```
SELECT coluna1, coluna2, função(...)

FROM nome_da_tabela

GROUP BY nome_da_coluna;
```


Operador Group by

O operador Group By, é utilizado quando queremos agrupar valores de uma coluna. Grande parte de relatórios que vemos com totalizadores de alguma informação está utilizando ele. Um ponto importante sobre o uso deste operador; sempre devemos utilizá-lo junto com alguma função de agregação.

```
SELECT nome, sobrenome, AVG(salario) AS media_salario

FROM funcionarios

WHERE departamento = 'Vendas'

GROUP BY nome, sobrenome;
```


Operador Having

Este operador é muito similar ao Where, no entanto, ele é somente utilizado em conjunto com o operador Group By. Enquanto o operador Where aplica uma condição para cada registro retornado na consulta, o Having aplica a condição na consulta depois que os registros são agrupados.

```
SELECT coluna1, coluna2, função(...)

FROM nome_da_tabela

GROUP BY nome_da_coluna;

HAVING funcaodoagrupamento(...)
```


Operador Having

Este operador é muito similar ao Where, no entanto, ele é somente utilizado em conjunto com o operador Group By. Enquanto o operador Where aplica uma condição para cada registro retornado na consulta, o Having aplica a condição na consulta depois que os registros são agrupados.

```
SELECT cliente_id, SUM(quantidade) AS quantidade_total

FROM pedidos

GROUP BY cliente_id

HAVING SUM(quantidade) > 10;
```


Operador Order by

Ordenando Dados com a Cláusula ORDER BY: A cláusula ORDER BY é usada para ordenar o conjunto de resultados em ordem ascendente (ASC) ou descendente (DESC):

```
SELECT coluna1, coluna2
FROM nome_da_tabela
ORDER BY nome_da_coluna ASC|DESC;
```


Operador Order by

Ordenando Dados com a Cláusula ORDER BY: A cláusula ORDER BY é usada para ordenar o conjunto de resultados em ordem ascendente (ASC) ou descendente (DESC):

```
SELECT nome_produto, preço
FROM produtos
ORDER BY preço DESC;
```


Operador Limit

A cláusula LIMIT em SQL é usada para limitar o número de linhas retornadas por uma consulta. É especialmente útil quando você deseja visualizar apenas uma parte dos resultados, ou quando você está paginando resultados em uma interface de usuário.

```
SELECT coluna1, coluna2, ...
FROM nome_da_tabela
LIMIT quantidade_de_linhas;
```


Operador Limit

A cláusula LIMIT em SQL é usada para limitar o número de linhas retornadas por uma consulta. É especialmente útil quando você deseja visualizar apenas uma parte dos resultados, ou quando você está paginando resultados em uma interface de usuário.

```
SELECT *
FROM clients
LIMIT 5;
```


Ordem para uso das cláusulas em um consulta:

SELECT nome_coluna(s)

FROM nome_tabela

WHERE condição

GROUP BY nome_coluna(s)

HAVING condição

ORDER BY nome_coluna(s)

LIMIT condição

EXERCÍCIOS

BDEX1_HOSPITAL:

1. Quais são os pacientes ordenados por idade?
2. Quantos pacientes existem em cada departamento?
3. Qual é a média de idade dos pacientes por departamento?

EXERCÍCIOS

BDEX2_PASSAGENS_AEREAS:

1. Quais são os voos ordenados por data e hora de partida?
2. Quantos passageiros estão reservados para cada voo?
3. Qual é a média de passageiros reservados por voo?

EXERCÍCIOS

BDEX3_REDESOCIAL:

1. Quais são as postagens ordenadas por data e hora de criação?
2. Quantos comentários cada postagem possui?
3. Qual é o número médio de comentários por postagem?

EXERCÍCIOS

BDEX4_ESTOQUE:

1. Quais são os produtos ordenados por preço?
2. Quantos produtos existem em cada armazém?
3. Qual é o valor total de estoque por fornecedor?

EXERCÍCIOS

BDEX5_EVENTOS:

1. Quais são os eventos ordenados por data e hora?
2. Quantas atividades cada evento possui?
3. Qual é o número médio de participantes por atividade?

EXERCÍCIOS

BDEX6_CINEMA:

1. Quais são os filmes ordenados por nome?
2. Quantas sessões cada filme possui?
3. Qual é o horário mais popular para as sessões de cinema?

EXERCÍCIOS

BDEX7_TRANSACOESBANCARIAS:

1. Quais são as transações bancárias ordenadas por valor?
2. Quantas transações ocorreram em cada conta?
3. Qual é o valor médio das transações por usuário?

ALIAS

Em MySQL, um alias é um nome alternativo que você pode atribuir a uma tabela ou a uma coluna em uma consulta SQL. Isso permite que você se refira a essa tabela ou coluna usando o alias em vez do nome original, tornando suas consultas mais legíveis e concisas.

SELECT **c**.nome **AS cliente_nome**

FROM clientes **AS c**

Você pode definir um alias usando a palavra-chave AS ou simplesmente colocando o alias diretamente após o nome da tabela ou coluna na sua consulta.

ALIAS

Em MySQL, um alias é um nome alternativo que você pode atribuir a uma tabela ou a uma coluna em uma consulta SQL. Isso permite que você se refira a essa tabela ou coluna usando o alias em vez do nome original, tornando suas consultas mais legíveis e concisas.

SELECT **c**.nome **AS cliente_nome**

FROM clientes **AS c**

Você pode definir um alias usando a palavra-chave AS ou simplesmente colocando o alias diretamente após o nome da tabela ou coluna na sua consulta.

JOINS

Em MySQL, assim como em outros sistemas de gerenciamento de banco de dados relacionais, os joins são utilizados para combinar registros de duas ou mais tabelas com base em uma condição relacionada entre elas. Existem diferentes tipos de joins que você pode usar, sendo os mais comuns:

- INNER JOIN: Retorna registros que têm correspondência em ambas as tabelas, com base na condição de junção especificada.
- LEFT JOIN: Retorna todos os registros da tabela da esquerda e os registros correspondentes da tabela da direita. Se não houver correspondência na tabela da direita, são retornados valores nulos.

JOINS

RIGHT JOIN: Retorna todos os registros da tabela da direita e os registros correspondentes da tabela da esquerda. Se não houver correspondência na tabela da esquerda, são retornados valores nulos.

FULL JOIN: Retorna todos os registros quando há uma correspondência em uma das tabelas. Os registros não correspondentes em ambas as tabelas também são incluídos na saída.

JOINS: INNER JOIN

Este exemplo retorna o nome do cliente e o valor do pedido para os clientes que fizeram pedidos, combinando registros das tabelas "clientes" e "pedidos" com base na correspondência das chaves primárias e estrangeiras.

```
SELECT clientes.nome, pedidos.valor

FROM clientes

INNER JOIN pedidos ON clientes.id = pedidos.cliente_id;
```


JOINS: INNER JOIN

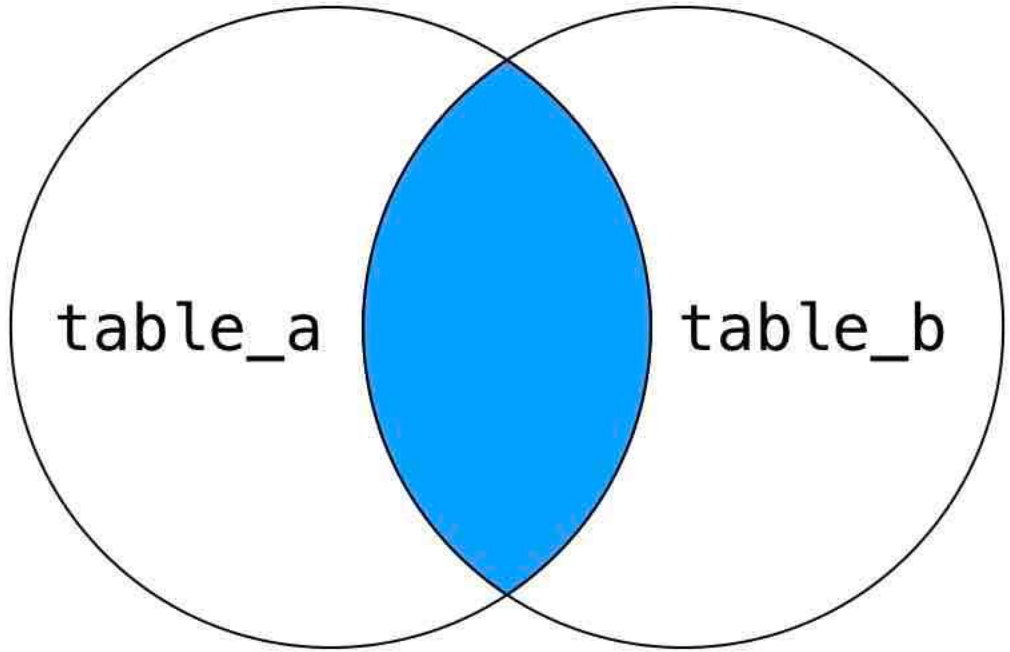
Se você usar apenas JOIN sem especificar o tipo (INNER, LEFT, RIGHT, etc.), o MySQL irá interpretá-lo como um INNER JOIN. Isso ocorre porque INNER JOIN é o tipo de join mais comum e frequentemente o mais útil em consultas SQL.

```
SELECT clientes.nome, pedidos.valor

FROM clientes

JOIN pedidos ON clientes.id = pedidos.cliente_id;
```


INNER JOIN




```
SELECT column_name
FROM table_a
INNER JOIN table_b
ON table_a.col_name = table_b.col_name;
```

JOINS: LEFT JOIN

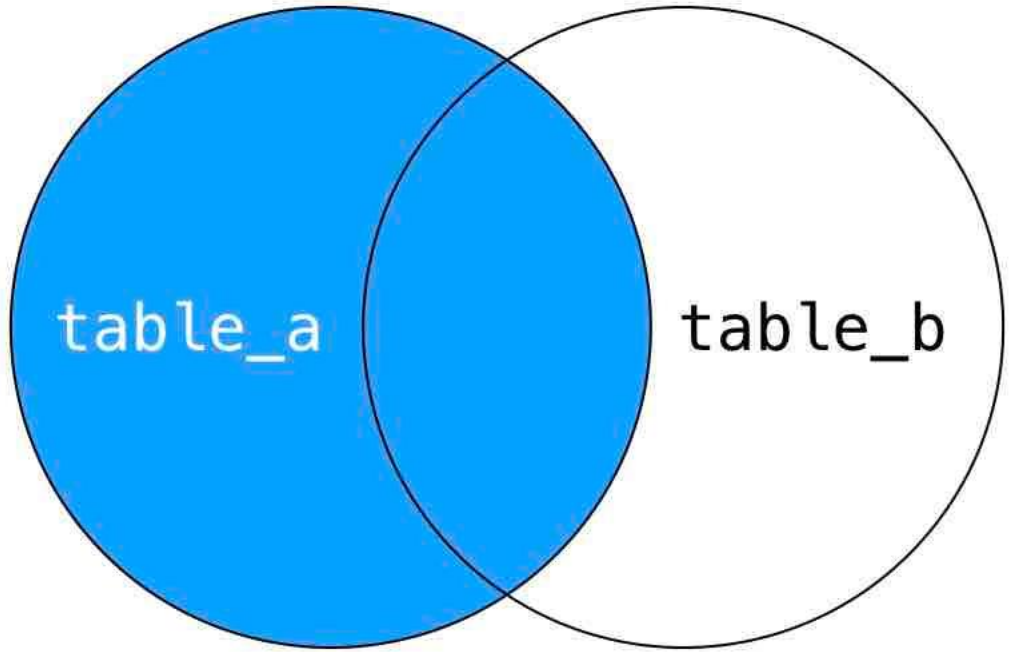
Este exemplo retorna o nome do cliente e o valor do pedido para todos os clientes, incluindo aqueles que não fizeram pedidos. Os valores de pedido para os clientes que não fizeram pedidos serão NULL.

```
SELECT clientes.nome, pedidos.valor

FROM clientes

LEFT JOIN pedidos ON clientes.id = pedidos.cliente_id;
```


LEFT JOIN




```
SELECT column_name
FROM table_a
LEFT JOIN table_b
ON table_a.col_name = table_b.col_name;
```

JOINS: RIGHT JOIN

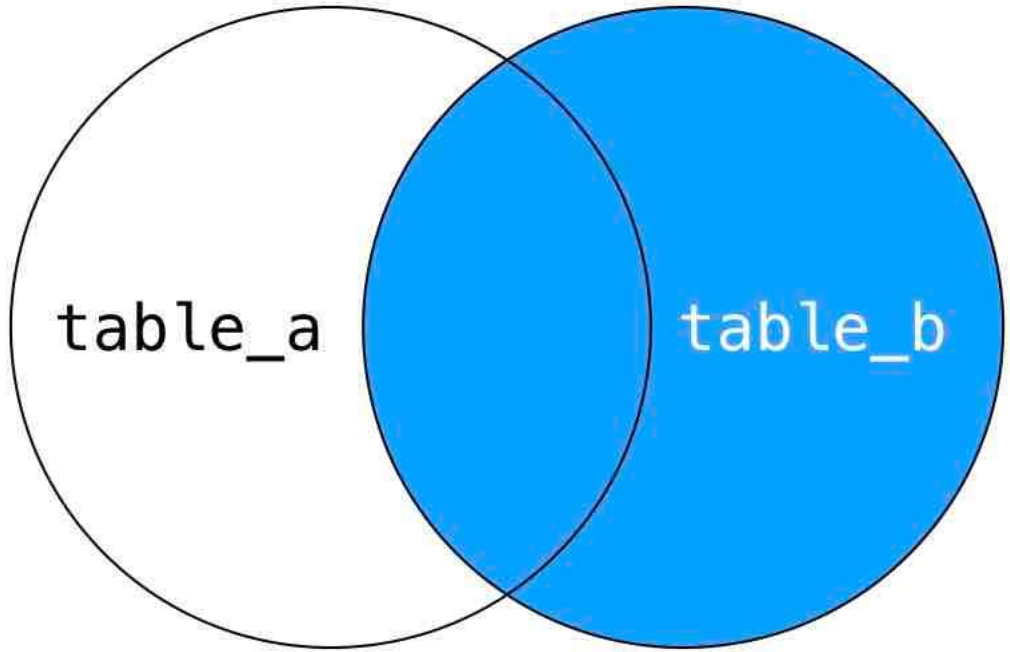
Este exemplo retorna o nome do cliente e o valor do pedido para todos os pedidos, incluindo aqueles que não têm correspondência com um cliente. Os valores de cliente para os pedidos que não têm um cliente correspondente serão NULL.

```
SELECT clientes.nome, pedidos.valor

FROM clientes

RIGHT JOIN pedidos ON clientes.id = pedidos.cliente_id;
```


RIGHT JOIN




```
SELECT column_name
FROM table_a
RIGHT JOIN table_b
ON table_a.col_name = table_b.col_name;
```

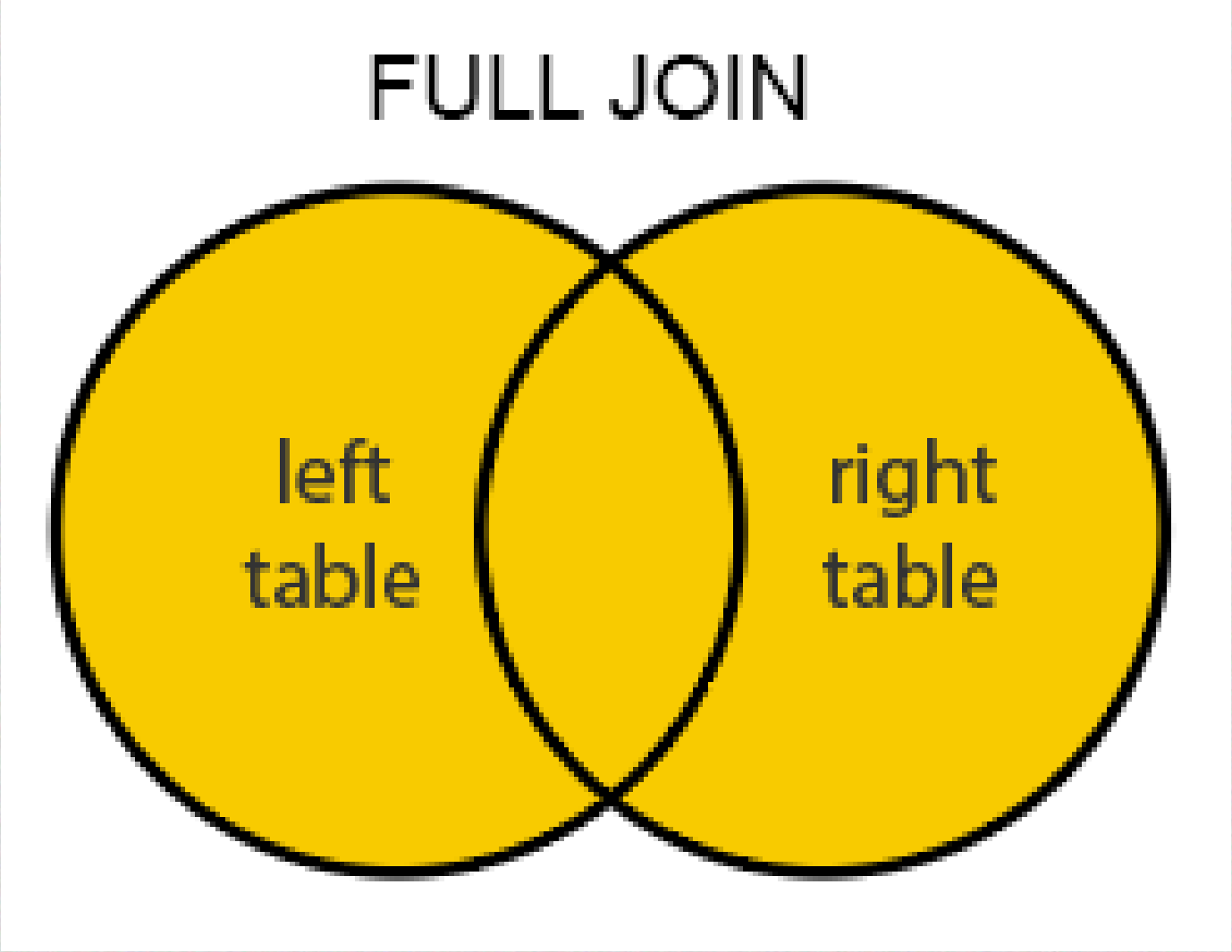
JOINS: FULL JOIN

O MySQL não possui suporte direto para FULL JOIN, no entanto, você pode simular um FULL JOIN combinando um LEFT JOIN com um RIGHT JOIN e usando a cláusula UNION para unir os resultados.

- A primeira parte da consulta executa um LEFT JOIN, que retorna todos os registros de "tabela1" e os registros correspondentes de "tabela2".
- A segunda parte da consulta executa um RIGHT JOIN, que retorna todos os registros de "tabela2" e os registros correspondentes de "tabela1".
- A cláusula WHERE tabela1.chave IS NULL na segunda parte garante que apenas os registros de "tabela2" que não têm correspondência em "tabela1" sejam incluídos no resultado.

JOINS: FULL JOIN

```
SELECT *
FROM tabela1
LEFT JOIN tabela2 ON tabela1.chave = tabela2.chave
UNION
SELECT *
FROM tabela1
RIGHT JOIN tabela2 ON tabela1.chave = tabela2.chave
WHERE tabela1.chave IS NULL;
```



FULL JOIN
left
table
right
table

EXERCÍCIOS

BDEX1_HOSPITAL:

1. Quais são os pacientes agendados para consulta com um determinado médico, incluindo informações do médico, em uma determinada data?
2. Quais são os médicos e seus respectivos departamentos, incluindo aqueles que não estão associados a nenhum departamento?
3. Quais são os pacientes que já passaram por consulta com um determinado médico, incluindo informações do médico, mesmo que não tenham sido atendidos?

EXERCÍCIOS

BDEX2_PASSAGENS_AEREAS:

1. Quais são os voos disponíveis de um determinado aeroporto para outro em uma determinada data, incluindo informações dos aeroportos de partida e chegada?
2. Quais são os passageiros reservados em um determinado voo, incluindo aqueles que ainda não reservaram?
3. Quais são os voos operados por uma determinada companhia aérea, incluindo aqueles que não têm operações?

EXERCÍCIOS

BDEX3_REDESOCIAL:

1. Quais são os usuários que pertencem a um determinado grupo, incluindo informações dos grupos e dos usuários?
2. Quais são as postagens de um usuário específico, incluindo o texto da postagem e as informações do usuário?
3. Quais são os usuários que realizaram transações, incluindo informações das transações e dos usuários?

EXERCÍCIOS

BDEX4_ESTOQUE:

1. Quais são os produtos fornecidos por um determinado fornecedor, incluindo informações do fornecedor e dos produtos?
2. Quais são os produtos disponíveis em um determinado armazém, incluindo informações dos produtos e dos armazéns?
3. Quais são os fornecedores que fornecem um determinado produto, incluindo aqueles que não têm fornecimentos registrados?

EXERCÍCIOS

BDEX5_EVENTOS:

1. Quais são as atividades planejadas para um determinado evento, incluindo informações dos eventos e das atividades?
2. Quais são os participantes inscritos em uma determinada atividade, incluindo aqueles que não têm inscrição registrada?
3. Quais são os eventos que ocorrerão em um determinado local, incluindo aqueles que não têm localização definida?

EXERCÍCIOS

BDEX6_CINEMA:

1. Quais são os espectadores que assistiram a um determinado filme em uma determinada data, incluindo informações dos espectadores e dos filmes?
2. Quais são os filmes em exibição em uma determinada sala, incluindo aqueles que não têm sessões programadas?
3. Quais são as sessões disponíveis para um determinado filme, incluindo informações das sessões e dos filmes?

EXERCÍCIOS

BDEX7_TRANSACOESBANCARIAS:

1. Quais são as transações realizadas por um determinado usuário, incluindo informações das transações e dos usuários?
2. Quais são as transações de débito realizadas em uma determinada conta, incluindo aquelas que não têm registros de débito?
3. Quais são os saldos registrados em uma determinada agência e conta, incluindo aqueles que não têm registros de saldo?

UVV

universidadevilavelha

uvvoficial

universidadevilavelha