

BANCO DE DADOS

Parte 8: SQL (Funções , Triggers e Joins)

Prof. Luiz Fernando Fernandes Miranda

Junho / 2017

Objetivos

- Entender o que são Funções
- Entender o que são Triggers e sua importância prática na montagem de rotinas após comandos de inserção, atualização ou remoção de dados
- Entender a cláusula JOIN e suas derivações



AGENDA

➤ FUNÇÕES

➤ Conceitos

➤ Tipos de Funções

➤ Exemplos

➤ TRIGGERS

➤ Conceitos

➤ Exemplos

➤ JOINS

➤ INNER JOIN

➤ LEFT e RIGHT JOINS



FUNÇÕES

- Assim como ocorre com os Store Procedures, é possível ter uma sequência de comandos SQL encapsulados em estruturas denominadas funções.
- Tanto os Stored Procedures como as Funções são procedimentos executados **diretamente no servidor SQL Server**, porém o Stored Procedure não tem obrigação de retornar valores para o usuário. Já a Função sim, ela deve retornar sempre um determinado dado ou até mesmo um conjunto de dados
- Funções não podem criar tabelas, nem inserir, excluir ou alterar dados em nenhuma tabela;



Funções (continuação)

- As funções internas já foram exploradas anteriormente (AVG, MIN, SUM...). Aqui as funções são definidas de acordo com o que é necessário para o usuário, com parâmetros de entrada e variáveis locais.
- Uma função é executada como uma parte de uma expressão.
- Construção sintática:

```
CREATE FUNCTION nome_da_funcao (parâmetros de entrada)
    RETURNS tipo_de_retorno
    BEGIN
        comandos em SQL
    RETURN valor_de_retorno
END;
```



TIPOS DE FUNÇÕES

É possível construir dois tipos de funções:

- **Funções escalares** – estruturas semelhantes a funções internas, que retornam um único valor.
- **Funções com valor de tabela (Não suportada pelo MySQL)** – estruturas semelhantes a visões com a diferença de aceitarem parâmetros de entrada e que retornam uma tabela como resultado do seu processamento.

As funções podem ser utilizadas do mesmo modo que uma função interna, sendo mais usada como:

- uma expressão na lista de um comando SELECT;
- uma expressão em uma cláusula WHERE ou HAVING;
- uma expressão em uma cláusula ORDER BY ou GROUP BY;
- uma expressão na cláusula SET de um comando UPDATE;
- uma expressão na cláusula VALUES de um comando INSERT.



Para termos o retorno da função por meio de uma variável, é necessário declará-la como variável local no MySQL:

```
DECLARE nome_da_variavel tipo_de_dado;
```

Exemplo prático:

```
DELIMITER $$
```

```
CREATE FUNCTION fun_totalpassageiros()
```

```
RETURNS int
```

```
BEGIN
```

```
    DECLARE total_passag int;
```

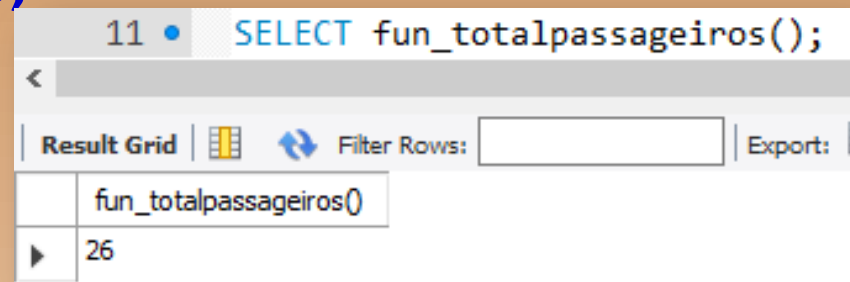
```
    SELECT SUM(voopassag_embarcou) INTO total_passag  
    FROM voo_passageiro;
```

```
    RETURN total_passag;
```

```
END $$
```

```
DELIMITER ;
```

```
SELECT fun_totalpassageiros();
```



The screenshot shows a MySQL query window with the query `SELECT fun_totalpassageiros();` executed. The result is displayed in a table with one row and one column, showing the value 26. A red arrow points from the text `SELECT fun_totalpassageiros();` in the code block to the result table.

fun_totalpassageiros()
26



Outro exemplo prático:

```
CREATE FUNCTION fun_totalpassageiros2(pr_numvoo int)
RETURNS int
BEGIN
    DECLARE total_passag2 int;
    SELECT SUM(voopassag_embarcou) INTO total_passag2 FROM
        voo_passageiro
        WHERE voopassag_numero = pr_numvoo;
    RETURN total_passag2;
END $$
DELIMITER ;
```

A função acima é parecida com a anterior, com a diferença que esta contém um parâmetro de entrada do tipo inteiro. É feita uma consulta (SELECT) da soma de todos os passageiros que embarcaram no voo passado pelo parâmetro

```
SELECT fun_totalpassageiros2(1627);
```



Luiz Fernando F Miranda

Result Grid	
	fun_totalpassageiros2(1627)
▶	4

Abaixo, a utilização da mesma função anterior, desta vez para listar a quantidade de passageiros embarcados em cada vôo cadastrado:

```
SELECT voopassag_numero,  
fun_totalpassageiros2(voopassag_numero) AS  
Total FROM voo_passageiro GROUP BY  
voopassag_numero;
```



	voopassag_numero	Total
▶	1627	4
	1628	4
	1731	3
	1732	3
	2264	4
	2335	4
	4996	4

Para excluir uma função do banco de dados, você deve utilizar o comando **DROP FUNCTION**

```
DROP FUNCTION fun_totalpassageiros;
```



TRIGGERS (Gatilhos)

- Procedimentos invocados quando um comando DML é executado (INSERT, UPDATE ou DELETE)
- Usos de um Trigger:
 - Validação e Integridade de dados
 - Rastreamento de logs de tabelas
 - Arquivamento de registros históricos
- Um Trigger é associado a uma única tabela
- É executado automaticamente (diferente dos Stored Procedures e Funções)



Triggers (continuação)

Sintaxe:

```
CREATE TRIGGER <nome-do-trigger>  
BEFORE | AFTER <comando DML>  
ON <nome-da-tabela>  
FOR EACH ROW <comando>
```

BEFORE | AFTER : O comando será executado antes ou depois da ação do comando DML

<comando DML>: INSERT, UPDATE ou DELETE



Exemplo prático

Será calculado um preco de passagem promocional (desconto de 5%) antes de cada inserção de um novo registro na tabela precovoo.

```
INSERT INTO cadvoo VALUES (2003, 'GOL', 'REC', 'GRU') ;
```

Criação do TRIGGER:

```
CREATE TRIGGER trig_promocao BEFORE INSERT  
ON precovoo FOR EACH ROW SET  
NEW.precovoo_promocao = (NEW.precovoo_valor*0.95) ;
```

Inserção de um novo registro:

```
SELECT * FROM precovoo WHERE precovoo_numero = 2003;
```



1. O preço promocional foi calculado antes da criação do registro na tabela precovoo
2. Os dados (inclusive o calculado) foram inseridos na tabela

```
SELECT * FROM precovoo  
WHERE precovoo_numero = 2003;
```



precovoo_numero	precovoo_data	precovoo_valor	precovoo_promocao
2003	2017-06-07	460.00	437.00
NULL	NULL	NULL	NULL



Outro exemplo de TRIGGER:

Deseja-se guardar o histórico de alterações de preços dos vôos. Para isso vamos criar outra tabela no sistema (**hist_precovoo**):

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;  
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,  
    FOREIGN_KEY_CHECKS=0;  
SET @OLD_SQL_MODE=@@SQL_MODE,  
SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
```

Tabela com histórico dos preços dos voos

```
CREATE TABLE IF NOT EXISTS controlevoos2.hist_precovoo  
(hprecovoo_numero INT NOT NULL,  
 hprecovoo_data DATE NOT NULL,  
 hprecovoo_valor DECIMAL(8,2) NOT NULL,  
 hprecovoo_promo DECIMAL(8,2) NULL,  
 CONSTRAINT fk_hprecovoo_numero  
 FOREIGN KEY (hprecovoo_numero)  
 REFERENCES precovoo (precovoo_numero)  
 ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE = InnoDB;
```



Antes...

precovoo

hist_precovoo

precovoo_numero	precovoo_data	precovoo_valor	precovoo_promocao
2264	2017-04-01	379.00	NULL
NULL	NULL	NULL	NULL

hprecovoo_numero	hprecovoo_data	hprecovoo_valor	hprecovoo_promo
NULL	NULL	NULL	NULL

DELIMITER \$\$

```
CREATE TRIGGER trig_historico_precos BEFORE UPDATE ON precovoo
FOR EACH ROW
BEGIN
```

```
    INSERT INTO hist_precovoo SET
        hprecovoo_numero = OLD.precovoo_numero,
        hprecovoo_data    = OLD.precovoo_data,
        hprecovoo_valor   = OLD.precovoo_valor,
        hprecovoo_promo   = OLD.precovoo_promocao;
```

END \$\$

DELIMITER ;

```
UPDATE precovoo SET PRECOVOO_DATA = current_date, precovoo_valor =
precovoo_valor * 1.10 where precovoo_numero = 2264;
```



precovoo_numero	precovoo_data	precovoo_valor	precovoo_promocao
2264	2017-06-07	416.90	NULL
NULL	NULL	NULL	NULL

hprecovoo_numero	hprecovoo_data	hprecovoo_valor	hprecovoo_promo
2264	2017-04-01	379.00	NULL
NULL	NULL	NULL	NULL

Se executarmos o comando UDPATE acima mais uma vez...



precovoo_numero	precovoo_data	precovoo_valor	precovoo_promocao
2264	2017-06-07	458.59	NULL
NULL	NULL	NULL	NULL

hprecovoo_numero	hprecovoo_data	hprecovoo_valor	hprecovoo_promo
2264	2017-04-01	379.00	NULL
2264	2017-06-07	416.90	NULL
NULL	NULL	NULL	NULL



Exemplo de procedimentos de auditoria:

Ações de atualização em banco de dados registrados em uma tabela própria de auditoria, através do uso de Triggers.

Para isso, criaremos uma tabela para auditoria:

```
CREATE TABLE  audit_aeronave
(aud_numordem INT NOT NULL AUTO_INCREMENT,
 aud_prefixo  VARCHAR(5) NOT NULL,
 aud_acao     VARCHAR(6) NOT NULL,
 aud_ciaaerea VARCHAR(10) NOT NULL,
 aud_datahora DATETIME NOT NULL,
 aud_usuario  VARCHAR(30) NOT NULL,
 PRIMARY KEY (aud_numordem) )
ENGINE = InnoDB;
```

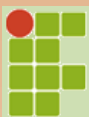


procedimentos de auditoria (continuação)

Criação de triggers para registrar dados sobre quem e quando os dados foram inseridos ou deletados da tabela de aeronaves

Trigger para inserção:

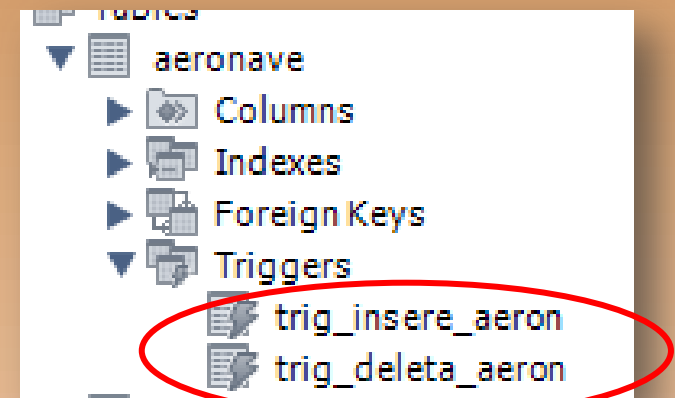
```
DELIMITER $$
CREATE TRIGGER trig_inserere_aeron
AFTER INSERT ON aeronave FOR EACH ROW
BEGIN
    INSERT INTO audit_aeronave SET
        aud_acao      = 'INSERT',
        aud_prefixo   = NEW.aeron_prefixo,
        aud_ciaaerea  = NEW.aeron_ciaaerea,
        aud_datahora  = NOW(),
        aud_usuario   = CURRENT_USER();
END $$
DELIMITER ;
```



Procedimentos de auditoria (continuação):

Trigger para deleção:

```
DELIMITER $$
CREATE TRIGGER trig_deleta_aeron
AFTER DELETE ON aeronave FOR EACH ROW
BEGIN
    INSERT INTO audit_aeronave SET
        aud_acao = 'DELETE',
        aud_prefixo = NEW.aeron_prefixo,
        aud_ciaaerea = NEW.aeron_ciaaerea,
        aud_datahora = NOW(),
        aud_usuario = CURRENT_USER();
END $$
DELIMITER ;
```



Inserção de duas novas aeronaves:

```
INSERT INTO aeronave VALUES ('PRQWA', 'A320-200', 2, 'AZL');  
INSERT INTO aeronave VALUES ('PRQWB', 'A320-200', 2, 'AZL');
```

aeronave

aeron_prefixo	aeron_tipo	aeron_config	aeron_ciaaerea
PRQWA	A320-200	2	AZL
PRQWB	A320-200	2	AZL

audit_aeronave

aud_numordem	aud_prefixo	aud_acao	aud_ciaaerea	aud_datahora	aud_usuario
1	PRQWA	INSERT	AZL	2017-06-07 17:55:01	root@localhost
2	PRQWB	INSERT	AZL	2017-06-07 17:55:06	root@localhost

Deleção de uma das aeronaves cadastradas antes:

```
DELETE FROM aeronave WHERE aeron_prefixo='PRQWB';
```

Verificando a tabela de auditoria:

```
SELECT * FROM audit_aeronave;
```

aud_numordem	aud_prefixo	aud_acao	aud_ciaaerea	aud_datahora	aud_usuario
1	PRQWA	INSERT	AZL	2017-06-07 17:55:01	root@localhost
2	PRQWB	INSERT	AZL	2017-06-07 17:55:06	root@localhost
3	PRQWB	DELETE	AZL	2017-06-07 17:56:26	root@localhost
NULL	NULL	NULL	NULL	NULL	NULL



JOIN

Cláusula utilizada para combinar dados provenientes de duas ou mais tabelas, baseado no relacionamento entre colunas destas tabelas

Há duas categorias de JOINS:

- **INNER JOIN** – retorna linhas (registros) quando houver pelo menos uma correspondência em ambas as tabelas
- **OUTER JOIN** – retorna linhas (registros) mesmo quando **NÃO** houver pelo menos uma correspondência em uma ou em ambas as tabelas. Esta divide-se em **LEFT JOIN**, **RIGHT JOIN** e **FULL JOIN**



JOIN (continuação)

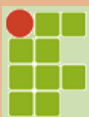
Exemplo prático:

```
SELECT * FROM AERONAVE A INNER JOIN MANUTENÇÃO M  
ON A.aeron_prefixo = M.aeron_prefixo;
```

aeron_prefixo	aeron_tipo	aeron_config	aeron_ciaaerea	aeron_prefixo	manut_dataprev	manut_categ	manut_datarealiz	aerop_codigo
PPALF	A320-200	1	AZL	PPALF	2017-04-02	MOT	2017-04-02	GRU
PPGBX	B737-300	1	GOL	PPGBX	2017-04-03	MOT	2017-04-03	GRU
PPGHU	B737-300	1	GOL	PPGHU	2017-04-09	EST	2017-04-10	GRU
PPGOK	B737-500	1	GOL	PPGOK	2017-04-16	TRE	2017-04-16	GRU
PTTLM	B777-200	1	TAM	PTTLM	2017-04-17	TRE	2017-04-17	GRU
PTTVR	B777-200	1	TAM	PTTVR	2017-05-23	MOT	NULL	GRU

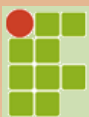
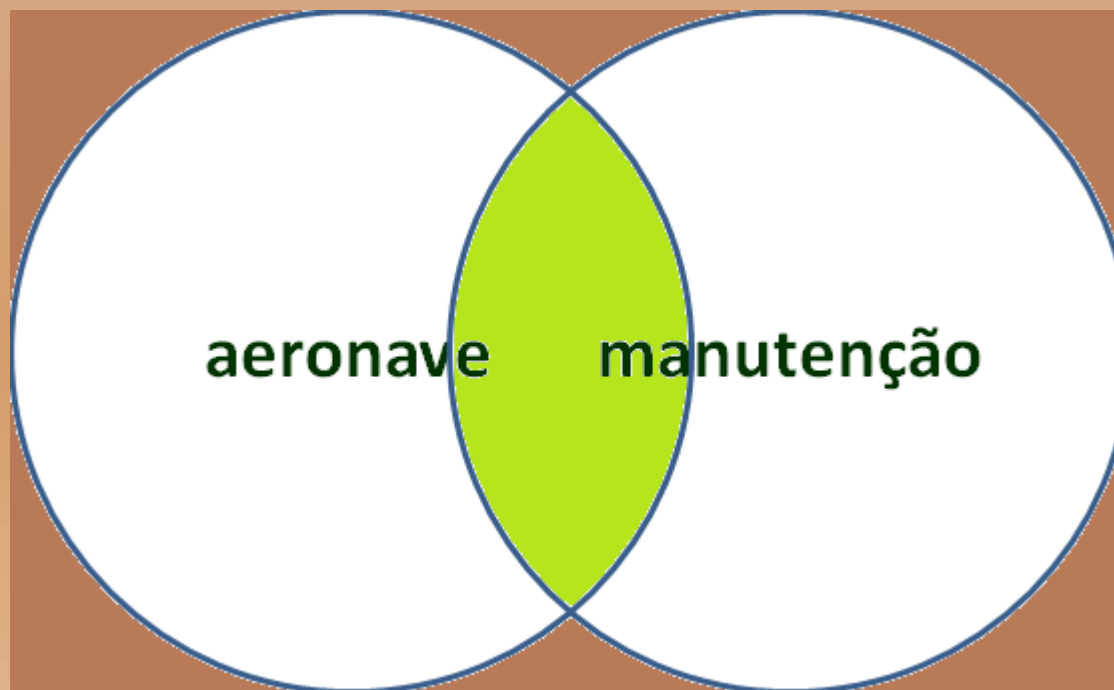
Obs: O comando acima é similar ao que se segue:

```
SELECT * FROM AERONAVE A, MANUTENÇÃO M  
WHERE A.aeron_prefixo = M.aeron_prefixo;
```



JOIN (continuação)

Representação Gráfica do INNER JOIN



OUTER JOIN

- **LEFT JOIN** – Retorna todas as linhas (registros) da tabela à esquerda, mesmo quando não houver nenhuma correspondência na tabela à direita;
- **RIGHT JOIN** – Retorna todas as linhas (registros) da tabela à direita, mesmo quando não houver nenhuma correspondência na tabela à esquerda;
- **FULL JOIN** – Retorna todas as linhas (registros) quando houver uma correspondência em qualquer uma das tabelas. É uma combinação de LEFT e RIGHT JOINS.



LEFT JOIN

```
SELECT * from aeronave A LEFT JOIN voo V
ON A.aeron_prefixo = V.voo_aeron_prefixo;
```

aeron_prefixo	aeron_tipo	aeron_config	aeron_ciaaerea	voo_numero	voo_aeron_prefixo	voo_datapart	voo_horapart	voo_datacheg	voo_horacheg
PPALF	A320-200	1	AZL	4551	PPALF	2017-03-23	07:30:00	2017-03-23	09:15:00
PPALF	A320-200	1	AZL	4551	PPALF	2017-03-23	19:30:00	2017-03-23	21:15:00
PPALF	A320-200	1	AZL	4552	PPALF	2017-03-23	12:45:00	2017-03-23	14:15:00
PPALF	A320-200	1	AZL	4552	PPALF	2017-03-23	22:45:00	2017-03-24	00:30:00
PPAWH	A321-100	1	AZL	NULL	NULL	NULL	NULL	NULL	NULL
PPAZN	A321-100	2	AZL	4404	PPAZN	2017-03-21	07:30:00	2017-03-21	10:15:00
PPAZN	A321-100	2	AZL	4405	PPAZN	2017-03-21	12:45:00	2017-03-21	15:15:00
PPAZN	A321-100	2	AZL	4996	PPAZN	2017-03-22	07:30:00	2017-03-22	10:45:00
PPAZN	A321-100	2	AZL	4997	PPAZN	2017-03-22	20:30:00	2017-03-22	23:50:00
PPFLU	A380-000	1	EMI	NULL	NULL	NULL	NULL	NULL	NULL
PPGBX	B737-300	1	GOL	2887	PPGBX	2017-03-19	07:30:00	2017-03-19	09:15:00
PPGBX	B737-300	1	GOL	2887	PPGBX	2017-03-20	07:30:00	2017-03-20	09:15:00
PPGBX	B737-300	1	GOL	2888	PPGBX	2017-03-19	12:45:00	2017-03-19	14:15:00
PPGBX	B737-300	1	GOL	2888	PPGBX	2017-03-20	12:45:00	2017-03-20	14:15:00
PPGFW	B737-400	1	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PPGHT	B737-400	1	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PPGHU	B737-300	1	GOL	2335	PPGHU	2017-03-19	20:30:00	2017-03-19	23:50:00

Todos os atributos das duas tabelas são retornadas, inclusive os registros da tabela à esquerda (aeronave) que não encontram correspondência com a tabela da direita (voo).

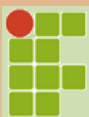
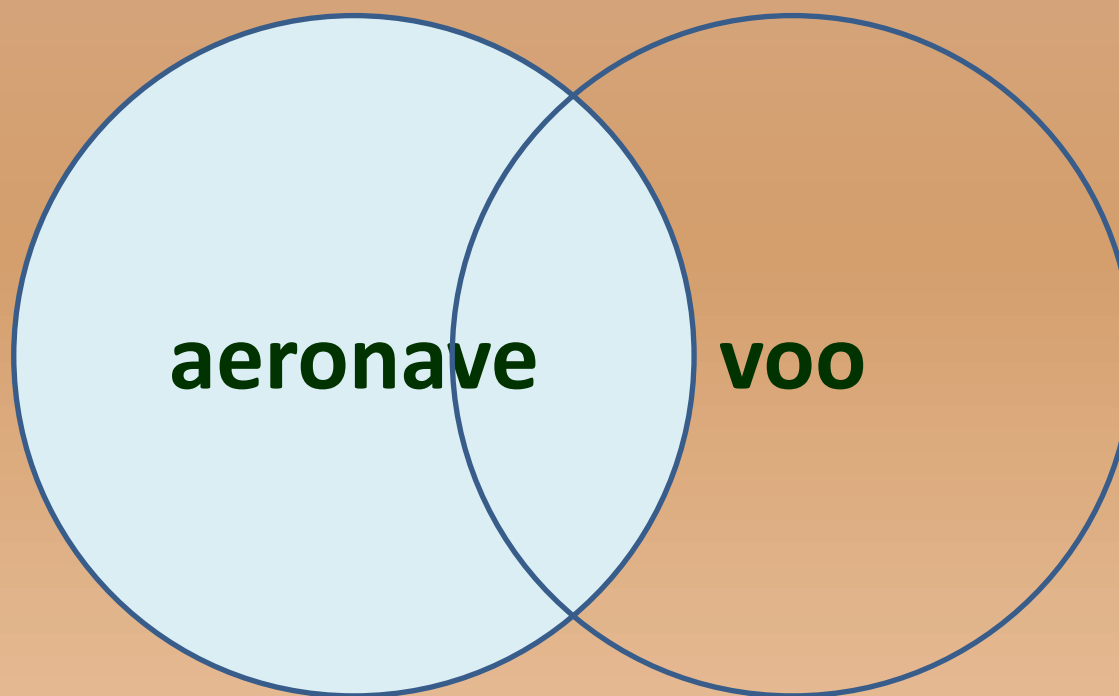


LEFT JOIN (continuação)

A consulta anterior responde à seguinte pergunta:

“Quais são as aeronaves cadastradas, inclusive aquelas que ainda não tiveram vôos registrados?”

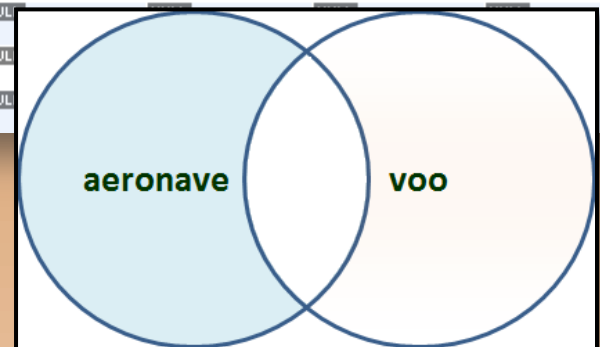
Representação Gráfica do LEFT JOIN



Outro exemplo (LEFT JOIN excluindo correspondências)

```
SELECT * from aeronave A LEFT JOIN voo V
      ON A.aeron_prefixo = V.voo_aeron_prefixo
WHERE V.voo_aeron_prefixo IS NULL;
```

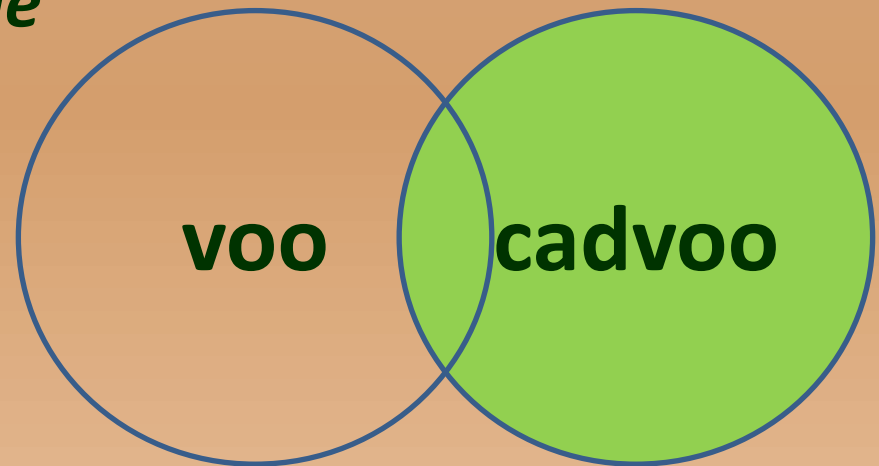
aeron_prefixo	aeron_tipo	aeron_config	aeron_ciaaerea	voo_numero	voo_aeron_prefixo	voo_datapart	voo_horapart	voo_datacheg	voo_horacheg
PPAWH	A321-100	1	AZL	NULL	NULL	NULL	NULL	NULL	NULL
PPFLU	A380-000	1	EMI	NULL	NULL	NULL	NULL	NULL	NULL
PPGFW	B737-400	1	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PPGHT	B737-400	1	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PPGJQ	B737-500	1	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PPGJV	B737-300	1	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PPGNX	B737-500	2	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PPGOK	B737-500	1	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PPGXH	B737-300	1	GOL	NULL	NULL	NULL	NULL	NULL	NULL
PRQWA	A320-200	2	AZL	NULL	NULL	NULL	NULL	NULL	NULL
PTBNT	A320-200	1	TAM	NULL	NULL	NULL	NULL	NULL	NULL
PTTEK	B777-200	1	TAM	NULL	NULL	NULL	NULL	NULL	NULL
PTTGE	A321-100	1	TAM	NULL	NULL	NULL	NULL	NULL	NULL
PTTKW	A321-100	1	TAM	NULL	NULL	NULL	NULL	NULL	NULL
PTTLM	B777-200	1	TAM	NULL	NULL	NULL	NULL	NULL	NULL
PTTVR	B777-200	1	TAM	NULL	NULL	NULL	NULL	NULL	NULL



RIGHT JOIN

Basicamente o mesmo conceito anterior, apenas invertendo a ordem das tabelas.

Todos os atributos das duas tabelas são retornadas, inclusive os registros da tabela à direita (cadvoo) que não encontram correspondência com a tabela da esquerda (voo).



RIGHT JOIN (continuação)

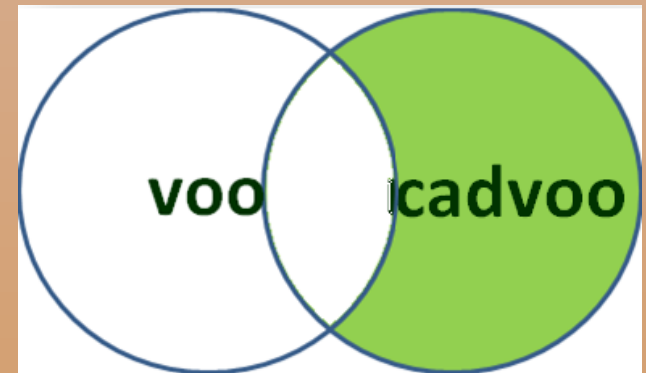
```
SELECT * FROM voo RIGHT JOIN cadvoo
ON voo_numero = cadvoo_numero
WHERE cadvoo_ciaaerea='GOL';
```

voo_numero	voo_aeron_prefixo	voo_datapart	voo_horapart	voo_datacheg	voo_horacheg	cadvoo_numero	cadvoo_ciaaerea	cadvoo_aerop_origem	cadvoo_aerop_destino
NULL	NULL	NULL	NULL	NULL	NULL	2001	GOL	REC	GIG
NULL	NULL	NULL	NULL	NULL	NULL	2002	GOL	GIG	REC
2264	PPGSD	2017-03-18	02:17:00	2017-01-01	04:55:00	2264	GOL	REC	CNF
2265	PPGSD	2017-03-18	06:00:00	2017-01-01	09:02:00	2265	GOL	CNF	REC
2335	PPGHU	2017-03-19	20:30:00	2017-03-19	23:50:00	2335	GOL	GIG	FOR
2336	PPGHU	2017-03-20	20:30:00	2017-03-20	23:50:00	2336	GOL	FOR	GIG
2711	PPGXV	2017-03-18	16:30:00	2017-03-18	19:15:00	2711	GOL	REC	BSB
2712	PPGXV	2017-03-18	22:45:00	2017-03-19	01:30:00	2712	GOL	BSB	REC
NULL	NULL	NULL	NULL	NULL	NULL	2755	GOL	GIG	BSB
NULL	NULL	NULL	NULL	NULL	NULL	2756	GOL	BSB	GIG
2887	PPGBX	2017-03-19	07:30:00	2017-03-19	09:15:00	2887	GOL	REC	FOR
2887	PPGBX	2017-03-20	07:30:00	2017-03-20	09:15:00	2887	GOL	REC	FOR
2888	PPGBX	2017-03-19	12:45:00	2017-03-19	14:15:00	2888	GOL	FOR	REC
2888	PPGBX	2017-03-20	12:45:00	2017-03-20	14:15:00	2888	GOL	FOR	REC
NULL	NULL	NULL	NULL	NULL	NULL	3131	GOL	REC	MCP
NULL	NULL	NULL	NULL	NULL	NULL	3132	GOL	MCP	REC



Outro exemplo (RIGHT JOIN excluindo correspondências)

```
SELECT * FROM voo RIGHT JOIN cadvoo
ON voo_numero = cadvoo_numero
WHERE cadvoo_ciaaerea='GOL' and
voo_numero IS NULL;
```



voo_numero	voo_aeron_prefix	voo_datapart	voo_horapart	voo_datacheg	voo_horacheg	cadvoo_numero	cadvoo_ciaaerea	cadvoo_aerop_origem	cadvoo_aerop_destino
NULL	NULL	NULL	NULL	NULL	NULL	2001	GOL	REC	GIG
NULL	NULL	NULL	NULL	NULL	NULL	2002	GOL	GIG	REC
NULL	NULL	NULL	NULL	NULL	NULL	2755	GOL	GIG	BSB
NULL	NULL	NULL	NULL	NULL	NULL	2756	GOL	BSB	GIG
NULL	NULL	NULL	NULL	NULL	NULL	3131	GOL	REC	MCP
NULL	NULL	NULL	NULL	NULL	NULL	3132	GOL	MCP	REC



REFERÊNCIAS

- Triggers: Definição, Sintaxe e correção
<https://www.youtube.com/watch?v=JOnkvqUaNOU>
- MySQL 5.7 Reference Manual
<http://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html>
- Apostila: Banco de Dados. Prof. Fernando da Fonseca de Souza e Valéria Cesário Times. Universidade Federal de Pernambuco – Centro de Informática
- Metrópole Digital (UFRN) – Sistemas de Banco de Dados – Aula 16
- INNER JOIN – Bóson Treinamentos
https://www.youtube.com/watch?v=C_OpAzDIImfl
- LEFT e RIGHT JOIN - Bóson Treinamentos
<https://www.youtube.com/watch?v=4m3HNtsFRoI>

