TRYBE
Modulo III – Back-end

Bloco 21 – SQL - Aprofundar

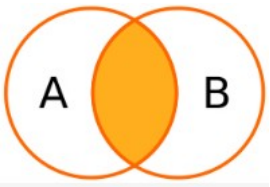
1) Funções mais usadas no SQL

Objetivo: encontrar informações estatísticas e temporais sobre seus dados.

INNER JOIN

SELECT t1.coluna, t2.coluna
FROM tabela1 AS t1
INNER JOIN tabela2 AS t2
ON t1.coluna_em_comum = t2.coluna_em_comum;

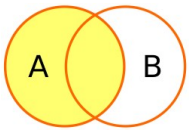
```
SELECT ci.city, ci.country_id, co.country
FROM sakila.city as ci
INNER JOIN sakila.country as co
ON ci.country_id = co.country_id;
```



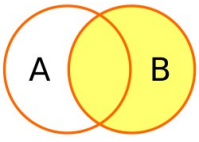
// mostrar cidade, id e pais de duas tabelas onde o ponto comum é o id (condição do ON)
INNER JOIN retorna registros de tabelas onde os dados se cruzam.

LEFT/RIGHT JOIN

A diferença é de **foco**.
No LEFT, são retornados **todos** os registros da tabela esquerda e valores correspondentes da tabela da direita, caso existam. Valores que não possuem correspondentes são exibidos como **nulos**.
O contrário com o RIGHT.



LEFT vs RIGHT



SELF

Caso em que uma tabela **faz JOIN consigo mesma**, usando os aliases:

SELECT columns
FROM sametable AS a, sametable AS b
WHERE condition

```
SELECT A.title, A.replacement_cost, B.title, B.replacement_cost
FROM sakila.film AS A, sakila.film AS B
WHERE a.length = b.length;
```

UNION / UNION ALL

Para **unir os registros de uma tabela com outra**.

SELECT columns from bank.table;
UNION
SELECT columns from otherbank.table;

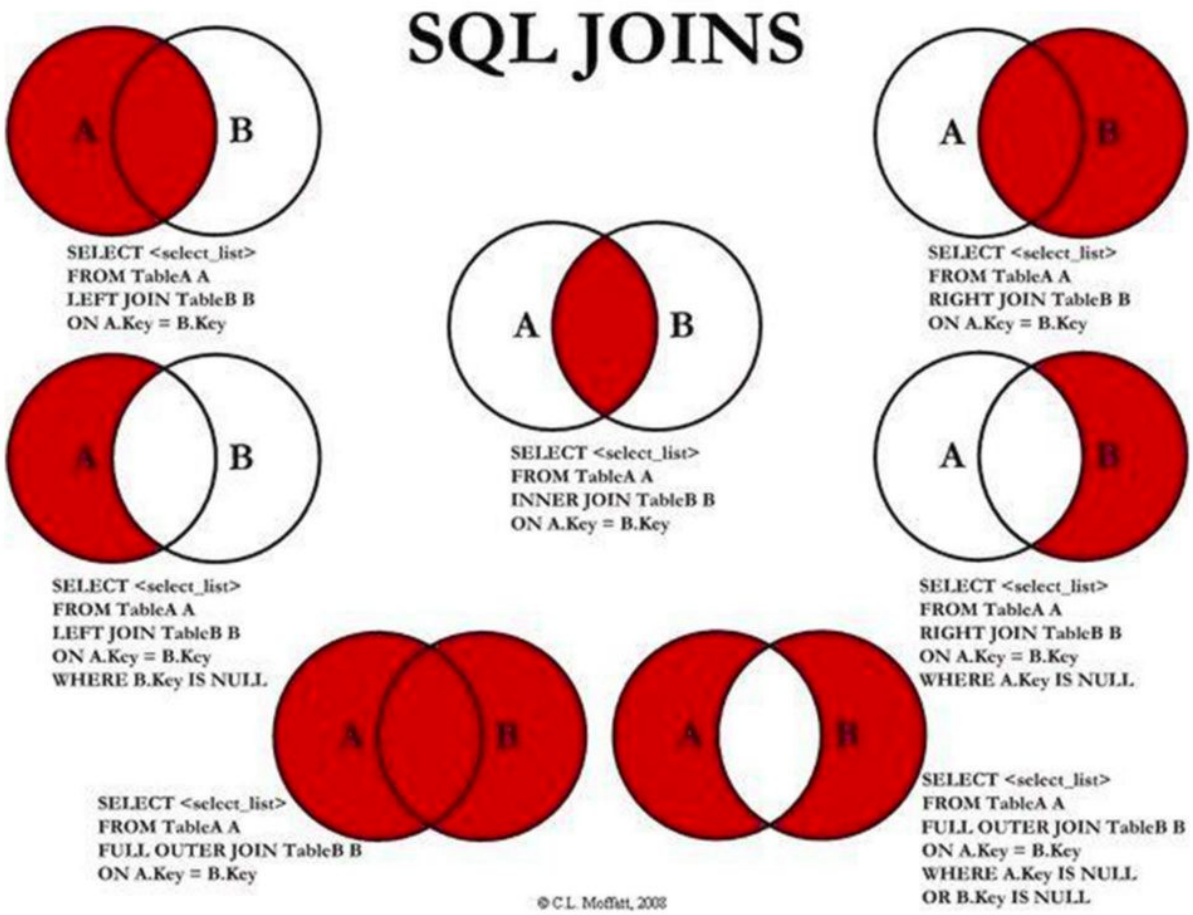
```
(SELECT first_name, last_name, 0 AS 'customer identification' FROM sakila.actor)
UNION
(SELECT first_name, last_name, customer_id FROM sakila.customer)
ORDER BY first_name;
```

// Têm que manter o mesmo número de colunas, mesmo simulando uma com valor padrão.
// Com parênteses para o Order funcionar corretamente

```
-- PÁGINAÇÃO LIMIT
( SELECT ... ORDER BY x  LIMIT 10 )
UNION
( SELECT ... ORDER BY x  LIMIT 10 )
ORDER BY x LIMIT 10
```

UNION remove os **dados duplicados**, enquanto o **UNION ALL** os **mantém.**

Resumo do dia



Select query with INNER JOIN on multiple tables

```
SELECT column, another_table_column, …
FROM mytable
INNER JOIN another_table
    ON mytable.id = another_table.id
WHERE condition(s)
ORDER BY column, … ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

Select query with LEFT/RIGHT/FULL JOINs on multiple tables

```
SELECT column, another_column, …
FROM mytable
INNER/LEFT/RIGHT/FULL JOIN another_table
    ON mytable.id = another_table.matching_id
WHERE condition(s)
ORDER BY column, … ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

//INNER vs OUTER join sintaxes

---

2) Stored Routines & Subqueries

SUBQUERY para montar uma query dentro de outra;
EXISTS para verificar se o resultado de uma query existe;
STORED PROCEDURES e STORED FUNCTIONS para criar blocos de código reutilizáveis (manifesto DRY: Don't Repeat Yourself).

SUBQUERY

Delimitada por **()** e dada um alias com **AS**. →

Usando SUBQUERY

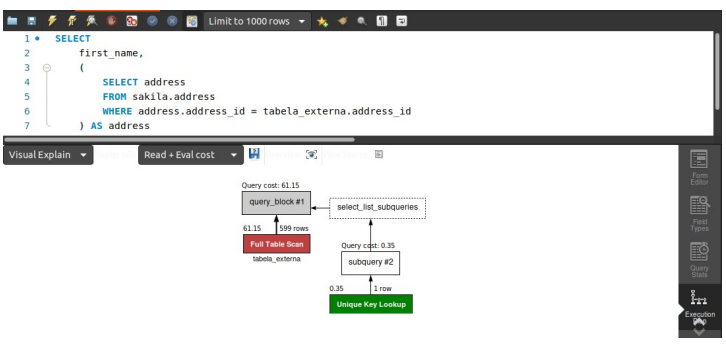
```
SELECT
    first_name,
    (
        SELECT address
        FROM sakila.address
        WHERE address.address_id = tabela_externa.address_id
    ) AS address
FROM sakila.customer AS tabela_externa;
```

Usando INNER JOIN

```
SELECT c.first_name, ad.address
FROM sakila.customer c
INNER JOIN sakila.address ad ON c.address_id = ad.address_id;
```

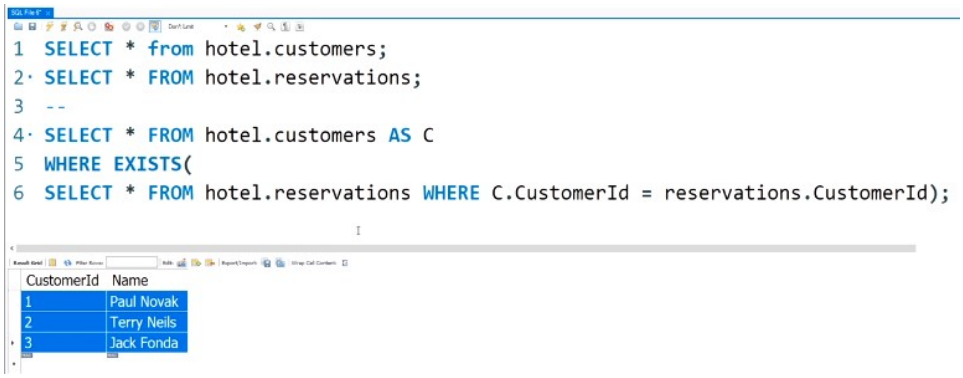
Escolher entre Subquery e Join

Comparar performances respetivas com o "query cost" ou seja Execution plan (quanto menor o valor, melhor).



EXISTS

Retorna os registros da tabela A que possuem um relacionamento com os registros da tabela B.



```
1  SELECT * from hotel.customers;
2  SELECT * FROM hotel.reservations;
3  --
4  SELECT * FROM hotel.customers AS C
5  WHERE EXISTS(
6  SELECT * FROM hotel.reservations WHERE C.CustomerId = reservations.CustomerId);
```

CustomerId	Name
1	Paul Novak
2	Terry Neils
3	Jack Fonda

//resposta: apenas 3 costumers fizeram reserva

Dica: também pode usar **NOT EXISTS**.

STORED PROCEDURES

Para armazenar pedaços de código reutilizáveis.

Variável
SET @my_school = 'BeTrybe';
SELECT @my_school;

Nomear: Padrão *VerboResultado*

Delimitar fim de declaração
DELIMITER &&

Tipos de dados (link)

MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)

Sintax

USE banco_de_dados; -- obrigatório para criar a procedure no banco correto
DELIMITER $$ -- definindo delimitador

CREATE PROCEDURE nome_da_procedure(@parametro1, @parametro2, ..., @parametroN) -- parâmetros
BEGIN -- delimitando o início do código SQL

END $$ -- delimitando o final do código SQL

DELIMITER ; -- muda o delimitador de volta para ; - o espaço entre DELIMITER e o ';' é necessário

Tipos de stored procedure (4):

Procedure *sem parâmetros*;

Procedure com parâmetros de *entrada (IN)*;
(Ao definir um parâmetro do tipo IN, podemos usá-lo e modificá-lo dentro da nossa procedure)

Procedure com parâmetros de *saída (OUT)*;
(útil quando você precisa que algo seja avaliado ou encontrado dentro de uma query e te retorne esse valor para que algo adicional possa ser feito com ele)

Procedure com parâmetros de *entrada e saída (IN-OUT)*.
(quando é necessário que um parâmetro possa ser modificado tanto antes como durante a execução de uma procedure)

Vantagem do Procedure (centralizar) vs Desvantagem (contra *separation of concerns*).

STORED FUNCTIONS

Definir o **tipo de retorno** da função:
DETERMINISTIC - Sempre retorna o mesmo valor ao receber os mesmos dados de entrada;
READS SQL DATA - Indica para o MySQL que sua função somente lerá dados.

```
-- DETERMINISTIC - Sempre retorna o mesmo valor ao receber os mesmos dados de entrada.

-- NON DETERMINISTIC - Nem sempre retorna o mesmo valor ao receber os mesmos dados de entrada.

-- READS SQL DATA - Indica para o MySQL que sua função irá somente ler dados, e não modifica tabelas.
```

Sintax

```
USE banco_de_dados; -- obrigatório para criar a função no banco correto
DELIMITER $$

CREATE FUNCTION nome_da_function(parametro1, parametro2, ..., parametroN)
RETURNS tipo_de_dado tipo_de_retorno
BEGIN
   query_sql
   RETURN resultado_a_ser_retornado;
END $$

DELIMITER ;
```

STORED PROCEDURES vs STORED FUNCTIONS

Function seria para fazer coisas pequenas porém que seriam repetitivas sem ela.
Procedure é para um trabalho mais amplo, realizando diversas operações ao mesmo tempo.

STORED FUNCTION	STORED PROCEDURE
Precisa retornar um valor	Pode ler valores e retornar um valor opcionalmente
Permite apenas parâmetros de entrada	Permite parâmetros de entrada e saida
Pode ser usada dentro de um select, having ou where	Não podem ser usadas dentro de um select, having ou where
Pode ser chamada por procedures mas não pode chamar procedures	Pode chamar outras functions

Functions VS Procedures

//chamadas através do comando SELECT (function) vs CALL (procedure)
// obrigatoriamente retorna algum valor vs retorno de valor opcional