



Unidade 2


Bancos de Dados Relacionais e Linguagem SQL

2.3 Gerenciando os dados relacionais: DDL, DML, DCL e TCL - DML.

Não se esqueça de importar os bancos!

- Vamos criar a estrutura a partir do arquivo .mwb
- <https://downloads.mysql.com/docs/sakila-db.zip>
- E importar os dados dos bancos:
- https://downloads.mysql.com/docs/world_x-db.zip
- <https://downloads.mysql.com/docs/menagerie-db.zip>

Data Manipulation Language - DML

- Ao manipularmos os dados utilizando as instruções DML iniciamos com INSERT, UPDATE, DELETE ou MERGE que são usadas para modificar os dados da tabela por meio da inclusão de novas linhas e da alteração ou remoção das linhas existentes.
-  A maior parte das operações que serão efetuadas pertencem a esta categoria.

Data Manipulation Language - DML

- 😊 SELECT é uma instrução DML porém, muito limitada. Na execução desta instrução apenas acessa-se os dados no banco de dados.
- Contudo apesar de não poder manipulá-los (alterações permanentes em registros ou campos nas tabelas), pode-se operar nos dados acessados retornando valores calculados ou organizados de forma diferente dos dados originais gravados.

Data Manipulation Language - DML

- `SELECT` é a palavras-chave mais importante da linguagem SQL pois é através dela que se recupera informações do banco de dados.
- Aprendendo a utilizar essa palavra-chave, a porta do banco de dados está aberta! Esta instrução lhe trará dados gerais e específicos.

Data Manipulation Language - DML

- A sintaxe básica da instrução é:

```
SELECT lista_dos_campos  
FROM    lista_das_tabelas  
WHERE   condição_basica_de_filtro (opcional)
```

- Na forma mais simples a cláusula SELECT, especifica as colunas a serem exibidas e a cláusula FROM, especifica a(s) tabela(s) que contêm as colunas listadas na cláusula SELECT.
- A Cláusula WHERE geralmente é utilizada para aplicação de algum critério específico de seleção.

Data Manipulation Language - DML

- Alguns conceitos para entendimento:
- Palavra-chave é um comando individual. Ex.: SELECT e FROM
- Cláusula é a parte da instrução SQL. EX.: SELECT last_name
- Instrução é uma combinação de duas ou mais cláusulas. Ex.:
SELECT last_name FROM actor

Data Manipulation Language - DML

- Dois recursos básicos são utilizados nas instruções SQL:

- **Projeção:** escolhe as colunas em uma tabela.

C1	C2	C3	C4

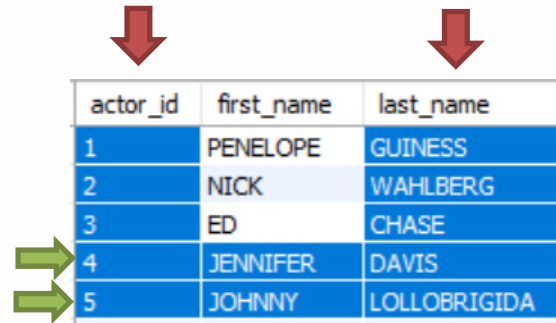
- **Seleção:** escolhe as linhas em uma tabela.

L1			
L2			
L3			
L4			

- Podemos escolher várias colunas e trabalhar com diversos critérios de seleção em uma mesma instrução

Data Manipulation Language - DML

- Na prática do comando:



actor_id	first_name	last_name
1	PENELOPE	GUINNESS
2	NICK	WAHLBERG
3	ED	CHASE
4	JENNIFER	DAVIS
5	JOHNNY	LOLLOBRIGIDA

Projeção

```
SELECT actor_id, last_name  
FROM actor  
WHERE first_name like 'J%'
```

Seleção

Data Manipulation Language - DML

- Caso queira projetar somente algumas colunas liste-as no comando separadas por vírgula.

```
SELECT * FROM country;
```

```
SELECT country_id,country,last_update FROM country;
```

```
SELECT country_id,country FROM country;
```

DML - SELECT - Operadores aritméticos

- Usando algumas regras e diretrizes simples, pode-se construir instruções SQL que tenham fácil de leitura e edição.
- A manipulação permitida através da instrução SELECT possibilita a modificação na visibilidade dos dados e, além disto, a criação de cenários hipotéticos.
- Simulações e aplicação de regras são necessárias em negócios!

DML - SELECT - Operadores aritméticos

- Esse tipo de simulação é possível com o uso de expressões aritméticas que já conhecemos da matemática como adição, subtração, multiplicação e divisão.
- Aplicando a simulação dos cálculos, os resultados são exibidos somente na saída do comando não alterando a gravação original.

DML - SELECT - Operadores aritméticos

```
SELECT payment_id, rental_id, amount  
FROM payment
```

payment_id	rental_id	amount
1	76	2.99
2	573	0.99
3	1185	5.99
4	1422	0.99
5	1476	9.99
6	1725	4.99
7	2308	4.99

```
SELECT payment_id, rental_id, amount+200  
FROM payment
```

payment_id	rental_id	amount+200
1	76	202.99
2	573	200.99
3	1185	205.99
4	1422	200.99
5	1476	209.99
6	1725	204.99
7	2308	204.99


DML - SELECT - Operadores aritméticos

- Precedência dos operadores é uma observação importantíssima a se fazer nesta abordagem pois, em uma instrução com vários operadores há que se avaliar quem tem a maior precedência.
- A ordem de precedência geralmente em bancos SQL é: Multiplicação, Divisão, Adição e Subtração (MDAS) avaliando sempre da direita para a esquerda.

DML - SELECT - Operadores aritméticos

- Precedência dos operadores é uma observação importantíssima a se fazer nesta abordagem pois, em uma instrução com vários operadores há que se avaliar quem tem a maior precedência.
- A ordem de precedência geralmente em bancos SQL é: Multiplicação, Divisão, Adição e Subtração (MDAS).
- $*$, $/$, $+$ e $-$ na ordem natural da matemática.

DML - SELECT - Operadores aritméticos

- Se esses operadores aparecerem juntos em uma expressão, a multiplicação e divisão são avaliadas primeiro.
- Se os operadores de uma expressão tiverem a mesma prioridade, a avaliação será feita da esquerda para a direita.
-  Para que você tenha uma melhor organização sempre utilize os parênteses para indicar qual operação será feita primeiro.

DML - SELECT - Operadores aritméticos

```
SELECT payment_id, payment_date,  
       amount, 30*amount+100  
FROM payment;
```

payment_id	payment_date	amount	30*amount+100
1	2005-05-25 11:30:37	2.99	189.70
2	2005-05-28 10:35:23	0.99	129.70
3	2005-06-15 00:54:12	5.99	279.70
4	2005-06-15 18:02:53	0.99	129.70
5	2005-06-15 21:08:46	9.99	399.70
6	2005-06-16 15:18:57	4.99	249.70


```
SELECT payment_id, payment_date,  
       amount, 30*(amount+100)  
FROM payment;
```

payment_id	payment_date	amount	30*(amount+100)
1	2005-05-25 11:30:37	2.99	3089.70
2	2005-05-28 10:35:23	0.99	3029.70
3	2005-06-15 00:54:12	5.99	3179.70
4	2005-06-15 18:02:53	0.99	3029.70
5	2005-06-15 21:08:46	9.99	3299.70
6	2005-06-16 15:18:57	4.99	3149.70

DML - SELECT - NULL

- Um conceito relevante e que vale a pena destacar na linguagem SQL é o NULL.
- NULL: valor indisponível, inaplicável, desconhecido ou não atribuído.
- 😊 NULL não é o mesmo que zero, espaço, traço ou qualquer outra coisa que achemos que se parece com ausência de valor.

DML - SELECT - NULL

- Caso haja a necessidade de trabalhar com uma coluna que possui valor nulo, atenção!
- Qualquer operação aritmética com o valor NULL trará resultado NULL a não na tentativa de divisão por 0 que dá erro!
-  Prestem bastante atenção nas operações que podem envolver valor o NULL pois podem invalidar o cálculo.

DML - SELECT - Aliases

- Alias é o nome que se dá ao campo renomeado na coluna de saída.
- A ideia é fazer com que a leitura fique mais amigável e entendível.
- Um alias de coluna renomeia um título de coluna, é útil para cálculos e deve vir imediatamente após o nome da coluna.

DML - SELECT - Aliases

- Para renomear a coluna a palavra chave AS é opcional, e caso haja mais de uma palavra separada por espaço no nome do *alias* este nome deverá estar entre aspas duplas.

```
SELECT payment_id, payment_date,  
       amount, 30*amount "Valor para um mês "  
FROM payment;
```

payment_id	payment_date	amount	Valor para um mês
1	2005-05-25 11:30:37	2.99	89.70
2	2005-05-28 10:35:23	0.99	29.70
3	2005-06-15 00:54:12	5.99	179.70

```
SELECT payment_id, payment_date,  
       amount, 30*(amount+100) as "Valor para um mês com multa"  
FROM payment;
```

payment_id	payment_date	amount	Valor para um mês com multa
1	2005-05-25 11:30:37	2.99	3089.70
2	2005-05-28 10:35:23	0.99	3029.70
3	2005-06-15 00:54:12	5.99	3179.70

DESCRibe - Pode salvá-lo em algum momento

- As vezes temos escassez de recursos e precisamos recuperar informações de estruturas que não conseguimos visualizar com facilidade.
- Para estes momentos utilize o comando ***describe*** que retorna o nome da tabela, os tipos de dados, as chaves primárias e estrangeiras, as colunas anuláveis e os comentários que podem conter explicações sobre as colunas.

DESCRibe - Pode salvá-lo em algum momento

```
desc payment;
```

```
describe payment;
```

Field	Type	Null	Key	Default	Extra
payment_id	smallint unsigned	NO	PRI	<code>NULL</code>	auto_increment
customer_id	smallint unsigned	NO	MUL	<code>NULL</code>	
staff_id	tinyint unsigned	NO	MUL	<code>NULL</code>	
rental_id	int	YES	MUL	<code>NULL</code>	
amount	decimal(5,2)	NO		<code>NULL</code>	
payment_date	datetime	NO		<code>NULL</code>	
last_update	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED on update CURRENT_TI...

- Essas informações são importantes para inserir novas linhas em uma tabela porque você precisa saber o tipo de dados aceitável por cada coluna e se ela pode ser deixada vazia ou não.

Função de concatenação de campos

- Em SQL, o operador de concatenação pode vincular colunas a outras colunas, expressões aritméticas ou valores de constante e criar uma expressão de caracteres.
- Este recurso deixa a saída mais amigável e economiza código.
- 😊 Além disto às vezes vocês terão que produzir alguns relatórios diretamente das queries do banco “os urgentes” e este recurso irá ajuda-los a apresentar o dado.

Função de concatenação de campos

- No mysql utilizaremos as funções [CONCAT](#) e [CONCAT_WS](#) para fazer este merge de colunas e utilize o alias para um nome amigável.

```
SELECT first_name, last_name,  
       CONCAT('Bem vindo: ',CONCAT_WS(' ', first_name, last_name)) as "Nome Completo"  
FROM actor
```

first_name	last_name	Nome Completo
PENELOPE	GUINESS	Bem vindo: PENELOPE GUINESS
NICK	WAHLBERG	Bem vindo: NICK WAHLBERG
ED	CHASE	Bem vindo: ED CHASE

Utilizando o *distinct*

- ***Distinct*** é um recurso que possuímos para eliminarmos as linhas duplicadas pois, muitas vezes, você vai querer saber quantas instâncias únicas de algo existe em uma tabela.
- Vamos por exemplo verificar em uma tabela que relaciona os filmes às categorias saber quais categorias temos filmes registrados em nossa tabela de relação.

Utilizando o distinct

- Se eu não utilizo o *distinct*

observe as linhas duplicadas

```
SELECT category_id  
FROM film_category
```

category_id
6
11
6
11
8
9
5
11
11

- Com o *distinct* eliminamos as linhas duplicadas.

```
SELECT DISTINCT (category_id)  
FROM film_category
```

category_id
1
2
3
4
5
6
7
8

Operadores de comparação

- $(=)$, $(<)$ e $(>)$ são operadores que já estamos acostumados no nosso dia a dia porém, existem outros comuns que o banco de dados nos ajuda a aplicar na solução de nossos problemas.
- Imagine uma agenda de encontro que marque compromissos entre 9:00 e 10:00 da quarta-feira, ou a compra de lugares no avião onde evita-se as fileiras de 15 a 24 por estarem em cima da asa como faríamos esta representação?

Operadores de comparação

- A linguagem SQL tem outros operadores para recuperar estes conjuntos de dados que são:
- **BETWEEN ... AND** : é usado para selecionar e exibir linhas com base em uma faixa de valores.
- **IN**: conhecida como "**condição de associação**", utiliza-se para testar se um valor está DENTRO (IN, em inglês) de um conjunto específico de valores informados.

Operadores de comparação

- **LIKE:** permite selecionar linhas correspondentes a caracteres, datas ou padrões de números. Possui dois símbolos chamados de caracteres curinga utilizados para construir a pesquisa:
 - (%): é usado para representar qualquer sequência de zeros ou mais caracteres.
 - (_): é usado para representar um caractere.

Between ... and

- Os valores especificados com a condição BETWEEN são inclusivos porém há que se observar que o valor do limite inferior deve ser listado primeiro.

```
SELECT payment_id, rental_id, amount, payment_date  
FROM payment  
WHERE amount BETWEEN 2 AND 5;
```

payment_id	rental_id	amount	payment_date
1	76	2.99	2005-05-25 11:30:37
6	1725	4.99	2005-06-16 15:18:57
7	2308	4.99	2005-06-18 08:41:48
9	3284	3.99	2005-06-21 06:24:45
12	5244	4.99	2005-07-09 13:24:07
13	5326	4.99	2005-07-09 16:38:01
15	7273	2.99	2005-07-27 11:31:22

- Poderemos incluir listas como por exemplo alguns endereços que pertencem a alguns distritos específicos.

```
SELECT *  
FROM address  
WHERE district IN ('Georgia', 'Tete', 'Gois') ;
```

address_id	address	address2	district	city_id	postal_code	phone	location	last_update
96	984 Efon-Alaiye Avenue		Gois	183	17119	132986892228	BLOB	2014-09-25 22:30:17
100	1308 Arecibo Way		Georgia	41	30695	6171054059	BLOB	2014-09-25 22:33:43
101	1599 Plock Drive		Tete	534	71986	817248913162	BLOB	2014-09-25 22:32:18
134	758 Junan Lane		Gois	190	82639	935448624185	BLOB	2014-09-25 22:30:18
187	1839 Szkesfehrvr Parkway		Gois	317	55709	947468818183	BLOB	2014-09-25 22:30:20
220	1201 Qomsheh Manor		Gois	28	21464	873492228462	BLOB	2014-09-25 22:30:15
233	356 Olomouc Manor		Gois	26	93323	22326410776	BLOB	2014-09-25 22:30:15

Poderíamos produzir este mesmo resultado utilizando apenas a cláusula Where?

LIKE

- Filtramos com o *like* partes de campos que possuem conteúdo que nos interessa.

```
SELECT *  
FROM customer  
WHERE email LIKE ('%MARY%');
```

customer_id	store_id	first_name	last_name	email	address_id	active
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	1
204	1	ROSEMARY	SCHMIDT	ROSEMARY.SCHMIDT@sakilacustomer.org	208	1

Será que as buscas textuais na Internet utilizam algo parecido com este princípio?

LIKE

- **IMPORTANTE:**
- Quando você precisar ter uma correspondência exata para uma string que tenha um caractere % ou _ , será preciso indicar que % ou _ não são curingas, mas parte do item que está sendo pesquisado.
- A opção ESCAPE pode ser usada para indicar que _ ou % faz parte do nome e não é um valor curinga.

IS NULL e IS NOT NULL

- **Falamos anteriormente do quão importante é considerar os dados nulos nas operações e consultas!**
- A condição IS NULL testa os dados indisponíveis, não atribuídos ou desconhecidos e IS NOT NULL testa os dados que estão disponíveis no banco de dados.
- Desta forma você garante que não estará trabalhando com dados nulos que podem gerar resultados errados na consulta.

INSERT

- Para aplicar o comando INSERT vamos primeiro ver a estrutura da tabela para sabermos o que estamos fazendo, **certo?!?**

```
DESC customer;
```

Field	Type	Null	Key	Default	Extra
customer_id	smallint unsigned	NO	PRI	NULL	auto_increment
store_id	tinyint unsigned	NO	MUL	NULL	
first_name	varchar(45)	NO		NULL	
last_name	varchar(45)	NO	MUL	NULL	
email	varchar(50)	YES		NULL	
address_id	smallint unsigned	NO	MUL	NULL	
active	tinyint(1)	NO		1	
create_date	datetime	NO		NULL	
last_update	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED on update CURRENT_TI...

INSERT

- E então vamos gerar o nosso comando INSERT com os parâmetros necessários:

```
INSERT INTO customer  
(store_id, first_name, last_name, email, address_id, active) VALUES  
( 2, 'AUGUSTO', 'ZADRA', 'AUGUSTO.ZADRA@sakilacustomer.com', 601, 0);
```

600	2	AUGUSTO	ZADRA	AUGUSTO.ZADRA@sakilacustomer.com	601	0	2021-10-19 02:38:05	2021-10-19 02:38:05
-----	---	---------	-------	----------------------------------	-----	---	---------------------	---------------------

- Há opção de supressão dos campos no cabeçalho mas deve ser feito com bastante atenção pois todos os parâmetros devem ser repassados de acordo com o describe, vejam:

```
INSERT INTO customer VALUES  
( NULL, 2, 'AUGUSTO', 'ZADRA', 'AUGUSTO.ZADRA@sakilacustomer.com', 601, 0, NULL, NULL);
```

INSERT

- E se eu precisar fazer uma carga de dados em uma tabela, copiando dados de uma outra, será que é possível?
- Sim, e este recurso é muito utilizado, às vezes para que possamos fazer correções, simulações e em outras situações precisamos fazer cópia de dados e inserção em outra tabela.
- Mas linha a linha demora, não é? Então como fazer?

INSERT

- E se eu precisar fazer uma carga de dados em uma tabela, copiando dados de uma outra, será que é possível?
- Sim, e este recurso é muito utilizado, às vezes para que possamos fazer correções, simulações e em outras situações precisamos fazer cópia de dados e inserção em outra tabela.
- Mas linha a linha demora, não é? Então como fazer?

INSERT

- Primeiro é criar a estrutura: customer_copy.
- Após é montar o *INSERT* extraindo os dados do *SELECT*:

```
INSERT INTO customer_copy (customer_id, store_id, first_name, last_name, email,  
                           address_id, active, create_date, last_update)  
select * from customer
```

- Perceba que é necessário repetir todos os campos conforme o *describe* para garantir que vai funcionar:

UPDATE

- A instrução UPDATE é usada para modificar as linhas existentes em uma tabela e requer quatro valores:
 - o nome da tabela
 - o nome da(s) coluna(s) cujos valores serão modificados
 - um novo valor para cada coluna que será modificada
 - **uma condição que identifique as linhas da tabela que serão modificadas**
- O novo valor de uma coluna pode ser o resultado de uma subconsulta de linha única.

UPDATE

- Nunca de esqueçam de incluir a clausula Where em uma instrução update.
- **Se a cláusula WHERE for omitida, todas as linhas da tabela serão atualizadas!**

```
update customer_copy set address_id = 602 where first_name = 'AUGUSTO'
```

*Now, we need to pratice
with our exercises!*

Referências bibliográficas

MySQL, Channel, 2021. MySQL Workbench Tutorial. Disponível em: <https://www.youtube.com/watch?v=X_umYKqKaF0>. Acesso em: 04 out. 2021.

MySQL, Manual 2021. MySQL Documentation Archive. Disponível em: <<https://dev.mysql.com/doc/index-archive.html>>. Acesso em: 04 out. 2021.