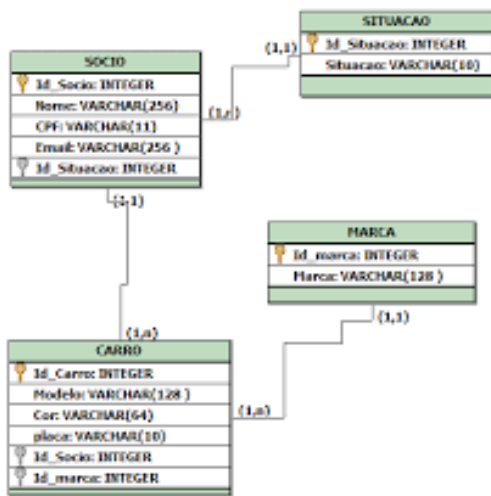


Bancos de Dados

Linguagem SQL – DQL – Data Query Language

Tabela Temporária, Tabela Derivada, Common Table Expression e Visões



Clóvis Ferraro
cferraro@unicid.edu.br

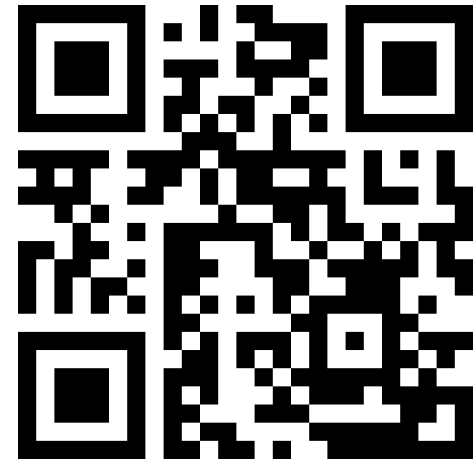
Exercício Aula Anterior

Criar Banco para funcionário

Esse banco é usado em muitos exemplos no Oracle, vamos criá-lo aqui no MySQL.

Fiz isso para facilitar para vocês quando houver exemplos na internet.

<https://codeshare.io/G6OPE>
N



Fazer o EER-Diagram do banco funcionário

Gerando EER-Diagram

- O MySQL Workbench traz uma ferramenta bem útil, que faz uma engenharia reversa no script SQL e gera a partir dele o modelo relacional (em tabelas).
- No workbench, acesse o menu **File -> New Model**,
- Depois acesse novamente o meu **File -> Import -> Reverse Engineer MySQL Create Script...**

Gerando EER-Diagram

- Após feito isso, você verá esta janela:
- Clique nos e selecione o arquivo **funcionario.sql**,
- Depois clique em **execute -> next -> finish.**

Reverse Engineer SQL Script

Input and Options

Reverse Engineer

Results

Input and Options

Select the script containing the schemas to reverse engineer

Select SQL script file:



File encoding:

☐ Place imported objects on a diagram

☐ Use ANSI quotes

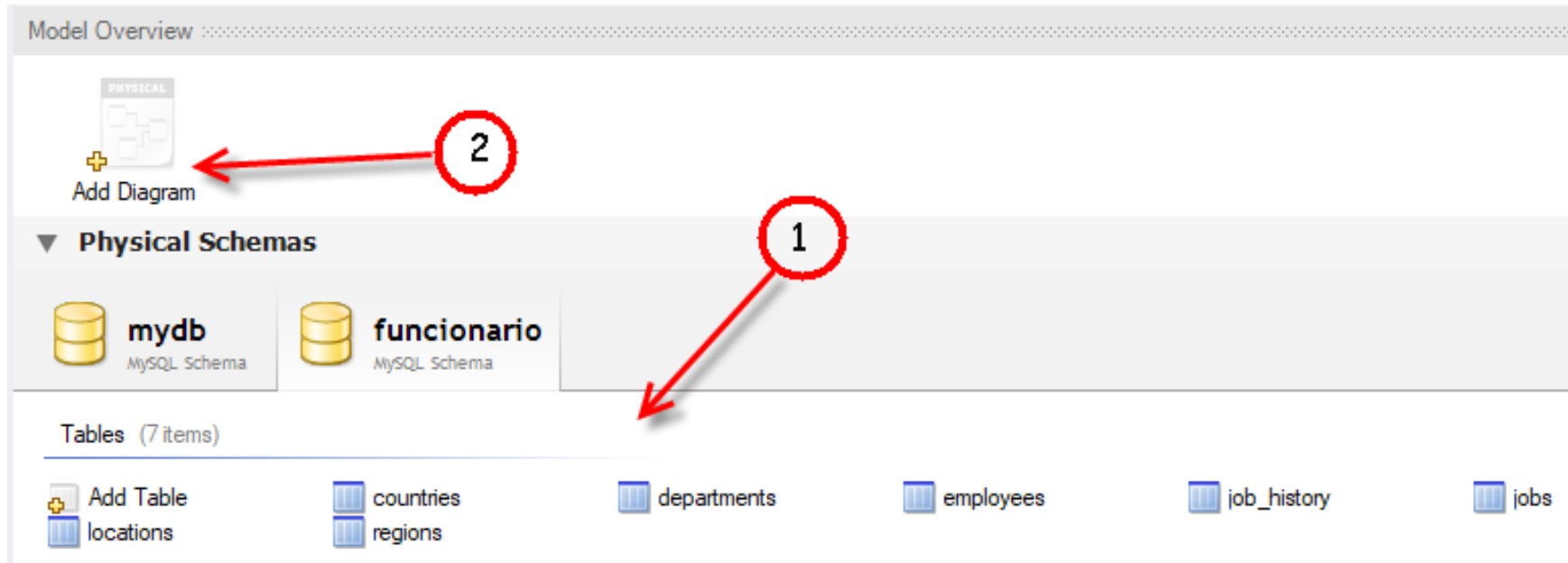
Back

Execute >

Cancel

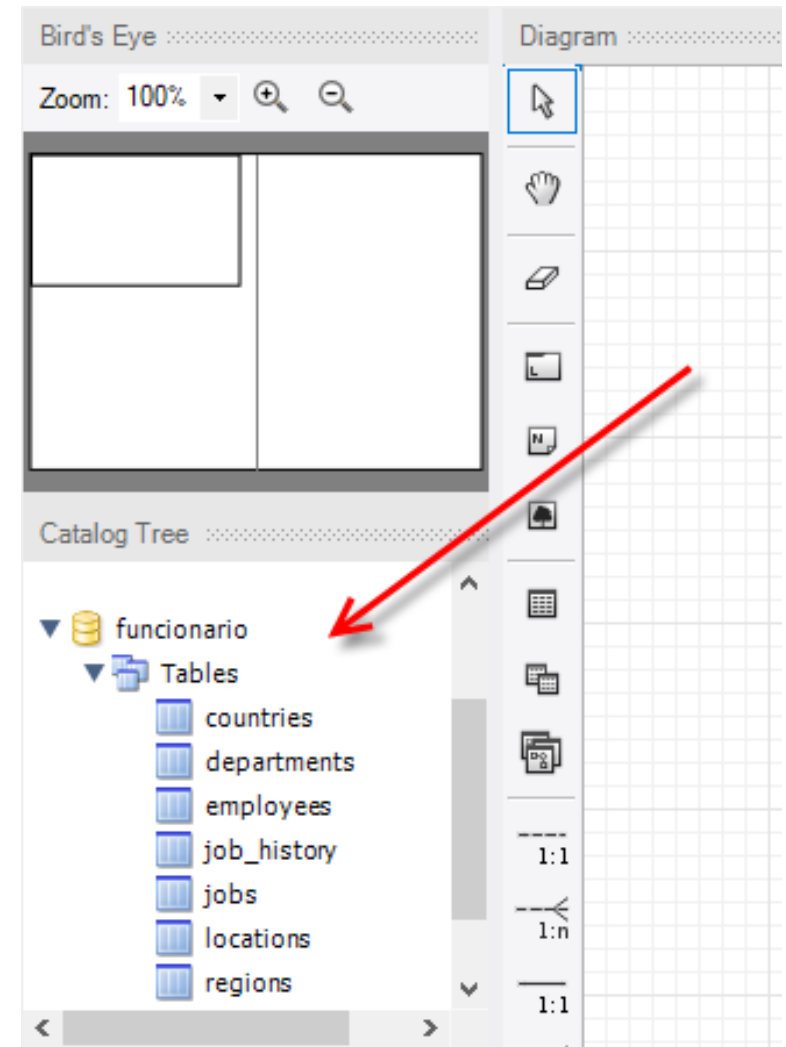
Gerando EER-Diagram

- Feito isso as tabelas já foram criadas, clique em **Add Diagram** na parte superior da tela.



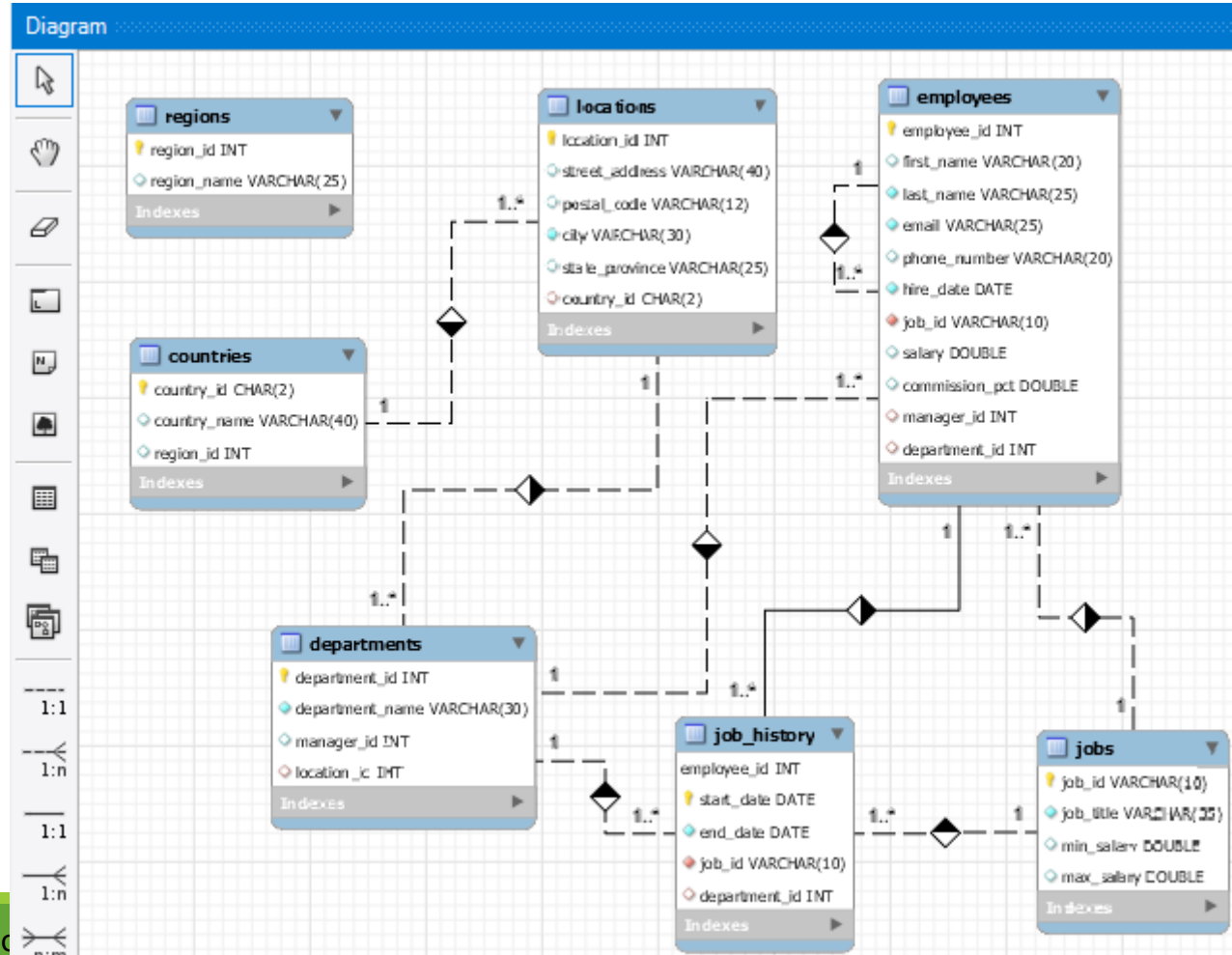
Gerando EER-Diagram

- Aberta a tela de criar um diagrama, navegue no menu à esquerda e selecione **funcionario -> tables**
- Clique e arraste as tabelas que quiser colocar no diagrama.



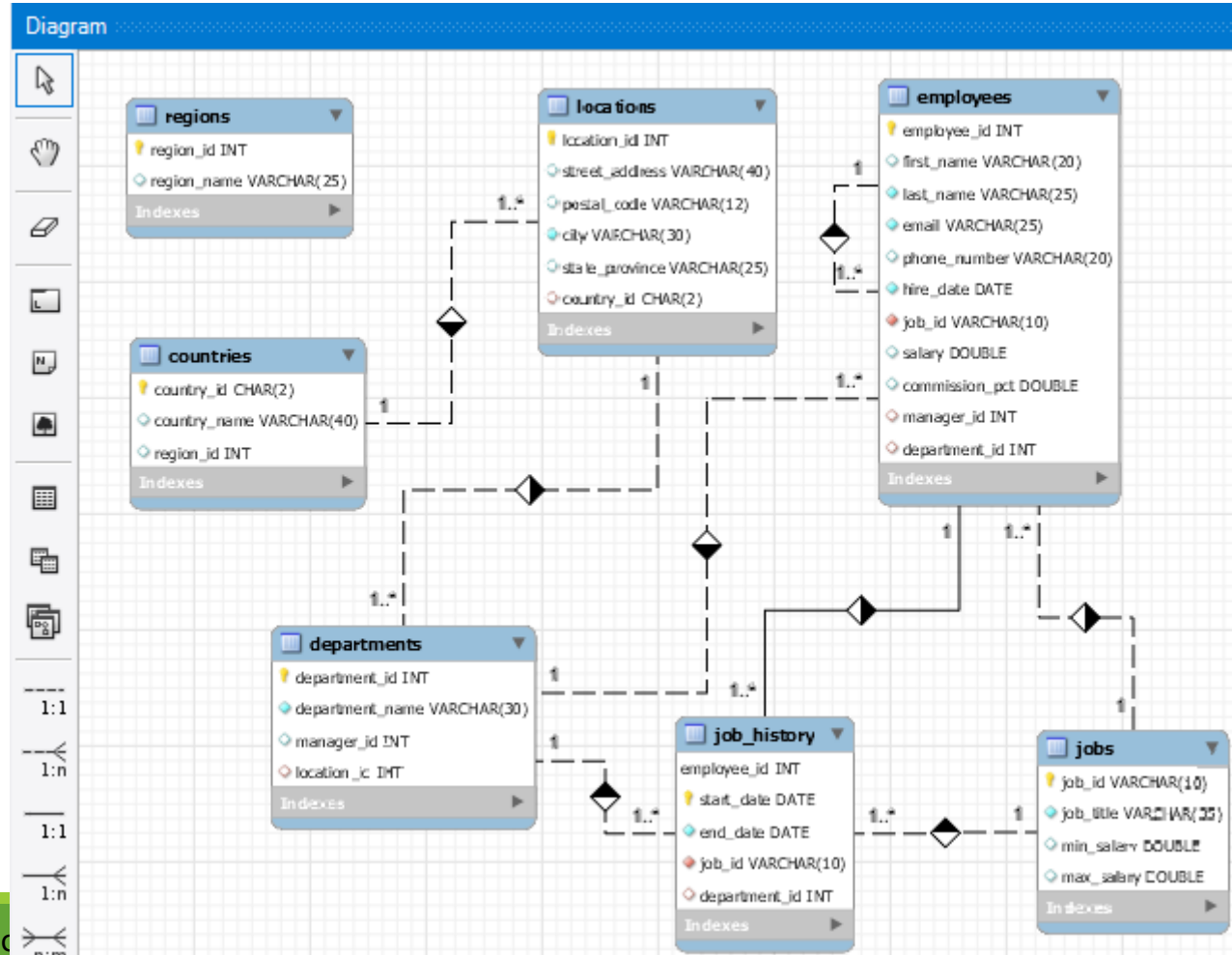
Gerando EER-Diagram

- O resultado deverá ser o diagrama ao lado, depois que você arrumá-lo 😊.



Gerando EER-Diagram

- Reparem que a tabela **regions** não tem relacionamento com nenhuma outra.
- Reparem também que na tabela **countries** existem uma chave estrangeira (region_id).
- O que faltou fazer no *script*?



Exercício para acompanhar a aula de hoje (classicmodels)

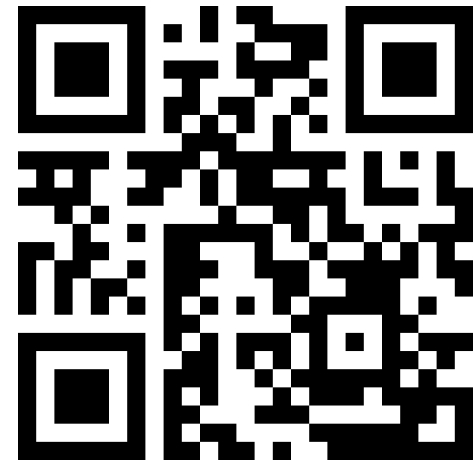
Criar Banco Exemplo -

classicmodels

<https://codeshare.io/G6ZV1z>

Esse banco também é usado em muitos exemplos no Oracle, vamos criá-lo aqui no MySQL.

Fiz isso para facilitar para vocês quando houver exemplos na internet.



Se quiserem, façam o EER-Diagram deste banco

MySQL Sample Database Diagram

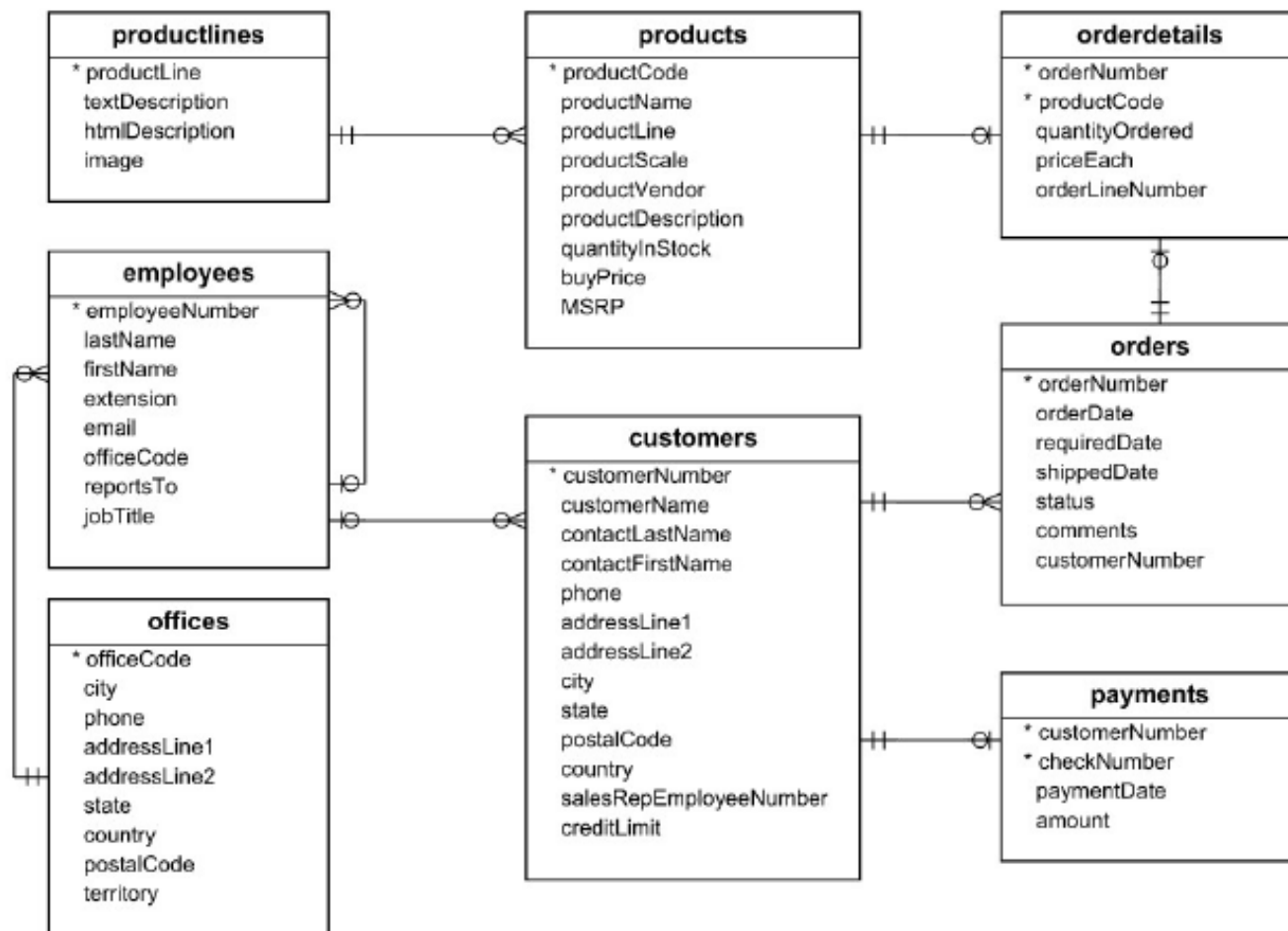


Tabela Temporária

Tabela Temporária

- Uma tabela temporária é muito útil quando é impossível ou caro consultar dados que requerem uma única instrução SELECT com as cláusulas JOIN.
- Nesse caso, você pode usar uma tabela temporária para armazenar o resultado imediato e usar outra consulta para processá-lo.
- O MySQL remove a tabela temporária automaticamente quando a sessão termina ou a conexão é encerrada. Claro, você pode usar a instrução DROP **TEMPORARY** para remover uma tabela temporária explicitamente quando não a estiver mais usando.

Tabela Temporária

- Sintaxe:

```
CREATE TEMPORARY TABLE table_name(  
    column_1_definition  
    ,column_2_definition  
    , ...  
    , table_constraints );
```

- Exemplo de criação de tabela temporária:

```
CREATE TEMPORARY TABLE credits(  
    customerNumber INT PRIMARY KEY  
    ,creditLimit DEC(10,2)  
    );
```

Tabela Temporária

- ✓ Uma tabela temporária está disponível e acessível apenas para o cliente que a cria. Usuários diferentes podem criar tabelas temporárias com o mesmo nome sem causar erros, porque apenas o usuário que cria a tabela temporária pode vê-la. No entanto, na **mesma sessão**, duas tabelas temporárias **não podem compartilhar o mesmo nome**.
- ✓ Uma tabela temporária **pode ter o mesmo nome de uma tabela normal** em um banco de dados. Por exemplo, se você criar uma tabela temporária *employees*, a tabela que já existia **ficará inacessível**. Cada consulta que você fizer na tabela **passará a se referenciar à tabela temporária**. Quando você “*dropar*” a tabela *employees* temporária, a *employees* permanente ficará disponível e acessível.

Tabela Temporária

- Mesmo que uma tabela temporária possa ter o **mesmo nome de uma tabela permanente, isso não é recomendado**. Porque isso pode causar confusão e, potencialmente, causar uma perda inesperada de dados.
- Por exemplo, no caso de a conexão com o servidor de banco de dados ser perdida e você se reconectar ao servidor automaticamente, não é possível diferenciar entre a tabela temporária e a permanente. Em seguida, você pode emitir uma declaração de **DROP TABLE para remover a tabela permanente em vez da tabela temporária**, o que não é esperado.
- Para evitar esse problema, você pode usar a declaração DROP **TEMPORARY** TABLE para eliminar uma tabela temporária.

Tabela Temporária

- Para criar uma tabela temporária cuja estrutura é **baseada** em uma **tabela existente**:

```
CREATE TEMPORARY TABLE temp_table_name  
  SELECT * FROM original_table  
  LIMIT 0;
```

Tabela Temporária

- Criando uma tabela temporária:

```
CREATE TEMPORARY TABLE CREDITS (  
  customerNumber int;  
  creditLimit decimal(10,2));
```


Tabela Temporária

- Inserindo linhas de outra tabela na tabela temporária:

```
INSERT INTO credits(customerNumber  
                    ,creditLimit)  
SELECT customerNumber  
        ,creditLimit  
FROM customers  
WHERE creditLimit > 0;
```

```
SELECT * FROM credits; -- Apenas para confirmar
```

98 row(s) returned



customerNumber	creditLimit
157	100600.00
161	84600.00
166	97900.00
167	96800.00
171	82900.00

Tabela Temporária

- Criando uma tabela temporária que armazena os 10 principais clientes por receita. A estrutura da tabela temporária é derivada de uma declaração *SELECT*.

```
CREATE TEMPORARY TABLE top_customers
SELECT p.customerNumber
      ,c.customerName
      ,ROUND(SUM(p.amount),2) sales
FROM payments p
INNER JOIN customers c
ON c.customerNumber = p.customerNumber
GROUP BY p.customerNumber
ORDER BY sales DESC
LIMIT 10;
```

Aqui apenas criamos a tabela temporária.

Tabela Temporária

- Consultando a tabela temporária (igual a uma tabela permanente)

```
SELECT
```

```
    customerNumber
```

```
    ,customerName
```

```
    ,sales
```

```
FROM top_customers
```

```
ORDER BY sales;
```

10 row(s) returned

customerNumber	customerName	sales
146	Saveley & Henriot, Co.	130305.35
321	Corporate Gift Ideas Co.	132340.78
276	Anna's Decorations, Ltd	137034.22
187	AV Stores, Co.	148410.09
323	Down Under Souvenirs, Inc	154622.08
148	Dragon Souvenirs, Ltd.	156251.03
151	Muscle Machine Inc	177913.95
114	Australian Collectors, Co.	180585.07
124	Mini Gifts Distributors Ltd.	584188.24
141	Euro+ Shopping Channel	715738.98

Tabela Temporária

- Eliminando uma tabela temporária.
- Você pode usar a instrução *DROP TABLE* para remover tabelas temporárias, mas é uma boa prática adicionar a palavra **TEMPORARY**.

```
DROP TEMPORARY TABLE top_customers;
```

- Se você tentar remover uma tabela permanente com a instrução *DROP TEMPORARY TABLE* receberá uma mensagem de erro dizendo que a tabela que você está tentando eliminar é desconhecida.

Tabela Temporária

- O MySQL não fornece uma função ou instrução para verificar diretamente se existe uma tabela temporária.
- Podemos criar um **procedimento** armazenado que verifica se uma tabela temporária existe ou não. (Veremos *PROCEDURE* mais adiante).

Tabela Derivada

Tabela Derivada

- Uma tabela derivada é uma **tabela virtual** retornada de uma instrução *SELECT*.
- É semelhante a uma tabela temporária, mas usar uma tabela derivada na instrução *SELECT* é muito mais simples do que uma tabela temporária, pois **não requer etapas de criação** da tabela temporária.
- O termo tabela derivada e subconsulta é frequentemente usado de forma intercambiável.
- Quando uma **subconsulta** independente é usada na cláusula **FROM** de uma instrução *SELECT*, ela é chamada de **tabela derivada**.

Tabela Derivada

- Sintaxe do comando de uma tabela derivada.

```
SELECT column_list
FROM (
    SELECT column_list
    FROM table_1
) derived_table_name
WHERE derived_table_name.c1 > 0;
```

Derived table

Must have an alias

Tabela Derivada

- Uma tabela derivada deve ter um alias para que você possa fazer referência a seu nome posteriormente na consulta.

```
SELECT column_list
FROM (
    SELECT column_list
    FROM table_1
) derived_table_name
WHERE derived_table_name.c1 > 0;
```

Derived table

Must have an alias

- Se uma tabela derivada não tiver um **alias**, o MySQL emitirá o seguinte erro:

Every derived table must have its own alias.

Tabela Derivada

Somente para quem ainda não criou o banco classicmodels da aula passada

<https://codeshare.io/5eBLol>

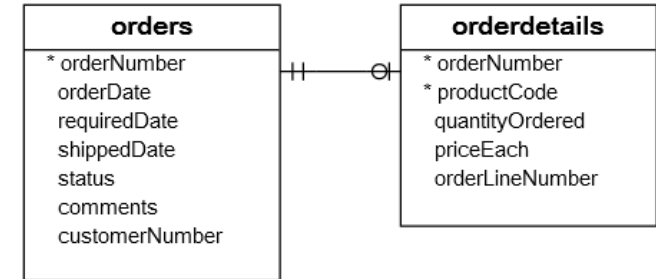


Tabela Derivada

- Exemplo: A consulta a seguir obtém os **5 produtos principais** por receita de vendas em 2003 das tabelas *orders* e *orderdetails* do banco exemplo.

```
SELECT productCode,  
       ROUND(SUM(quantityOrdered*priceEach)) sales  
FROM orderdetails  
INNER JOIN orders USING (orderNumber)  
WHERE YEAR(shippedDate) = 2003  
GROUP BY productCode  
ORDER BY sales DESC  
LIMIT 5;
```

No database selected = você não abriu o banco = USE classicmodels



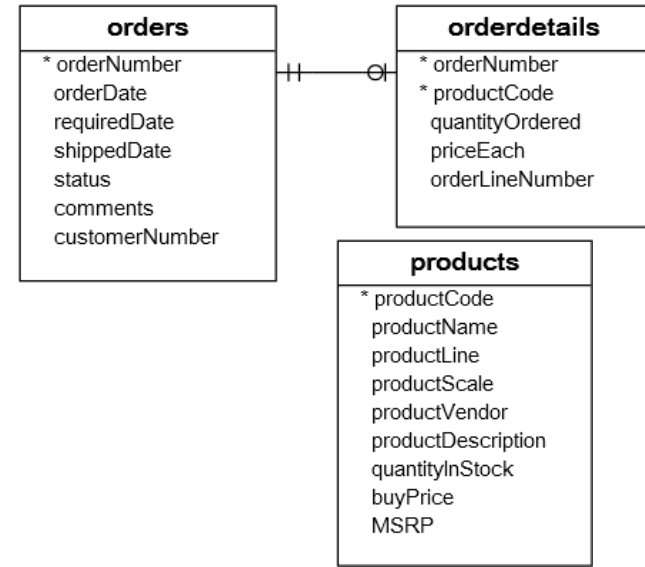
5 row(s) returned

productCode	sales
S18_3232	103480
S10_1949	67985
S12_1108	59852
S12_3891	57403
S12_1099	56462

Tabela Derivada

- Podemos usar o resultado desta consulta como uma tabela derivada e juntá-lo à tabela *products* :

```
SELECT
    productName, sales
FROM
    (SELECT
        productCode,
        ROUND(SUM(quantityOrdered * priceEach)) sales
    FROM
        orderdetails
    INNER JOIN orders USING (orderNumber)
    WHERE
        YEAR(shippedDate) = 2003
    GROUP BY productCode
    ORDER BY sales DESC
    LIMIT 5) top5products2003 --- Alias da Tabela
INNER JOIN products USING (productCode); Derivada
```

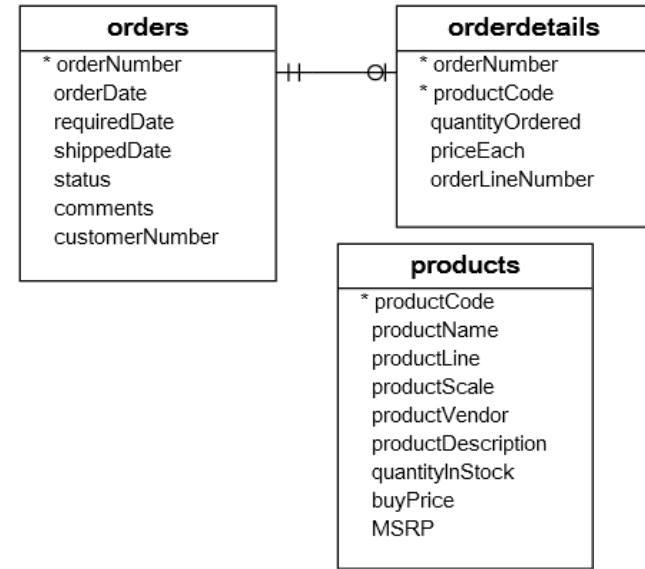


1. Primeiro, a subconsulta é executada para criar um conjunto de resultados ou tabela derivada.
2. Em seguida, é executada a consulta externa que uniu a tabela derivada *top5products2003* com a tabela *products* usando a coluna *productCode*.

Tabela Derivada

- Podemos usar o resultado desta consulta como uma tabela derivada e juntá-lo à tabela

```
SELECT
    productCode,
    productName, sales
FROM
    (SELECT
        productCode,
        ROUND(SUM(quantityOrdered * priceEach)) sales
    FROM
        orderdetails
    INNER JOIN orders USING (orderNumber)
    WHERE
        YEAR(shippedDate) = 2003
    GROUP BY productCode
    ORDER BY sales DESC
    LIMIT 5) top5products2003
INNER JOIN products USING (productCode);
```



5 row(s) returned

productName	sales
1992 Ferrari 360 Spider red	103480
1952 Alpine Renault 1300	67985
2001 Ferrari Enzo	59852
1969 Ford Falcon	57403
1968 Ford Mustang	56462

Common Table Expression

Common Table Expression

- Uma **Common Table Expression (CTE)** pode ser vista como um conjunto de resultados temporário que é definido no escopo de execução de uma única instrução *SELECT*, *INSERT*, *UPDATE* ou *DELETE*.
- Uma **CTE** é muito parecida com uma tabela derivada que não é armazenada como um objeto e que existe apenas durante a execução da consulta.
- Diferente de uma tabela derivada, uma CTE pode ser **autorreferenciada** e pode ser **referenciada várias vezes** na mesma consulta.

Common Table Expression

- Usando uma **CTE** temos as vantagens de melhorar a legibilidade e facilidade de manutenção de consultas complexas.
- A consulta pode ser dividida em blocos lógicos simples e separados.
- Estes blocos simples podem então ser utilizados para construir blocos mais complexos até que a consulta final seja montada.

Common Table Expression

- Sintaxe de um CTE :

```
WITH cte_name (column_list) AS (  
    query  
)  
SELECT * FROM cte_name;
```

- A estrutura de um CTE inclui o nome, uma lista de colunas opcional e uma consulta que define o CTE.
- Após a definição do CTE, você pode usá-lo como uma visualização em uma declaração SELECT, INSERT, UPDATE, DELETE, ou CREATE VIEW (veremos mais adiante).

Common Table Expression

```
WITH cte_name (column_list) AS (  
    query  
)  
SELECT * FROM cte_name;
```

- Observe que o número de colunas no **query** deve ser igual ao número de colunas no `column_list`.
- Se você omitir o `column_list`, o CTE usará a lista de colunas da consulta que define o CTE.

Common Table Expression

- Como usar um CTE para consultar dados da tabela *customers*

```
WITH customers_in_usa AS (  
    SELECT customerName, state  
    FROM customers  
    WHERE country = 'USA')  
  
SELECT customerName  
FROM customers_in_usa  
WHERE state = 'CA'  
ORDER BY customerName;
```

customers
* customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit

1. Neste exemplo, o nome do CTE é **customers_in_usa**, a consulta que define o CTE retorna duas colunas `customerName` e `state`. Portanto, o CTE **customers_in_usa** retorna todos os clientes localizados nos EUA.
2. Depois de definir o CTE **customers_in_usa** nós o referimos na declaração **SELECT** para selecionar apenas clientes localizados na Califórnia.

Common Table Expression

- Como usar um CTE para consultar dados da tabela *customers*

```
WITH customers_in_usa AS (  
    SELECT customerName, state  
    FROM customers  
    WHERE country = 'USA')  
SELECT customerName  
FROM customers_in_usa  
WHERE state = 'CA'  
ORDER BY customerName;
```

Esta forma é bem mais legível!

customers
* customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit

customerName
Boards & Toys Co.
Collectable Mini Designs Co.
Corporate Gift Ideas Co.
Men 'R' US Retailers, Ltd.
Mini Gifts Distributors Ltd.
Mini Wheels Co.
Signal Collectibles Ltd.
Technics Stores Inc.
The Sharp Gifts Warehouse
Toys4GrownUps.com
West Coast Collectables Co.

Visões (Views)

Views

- É uma tabela lógica (virtual) que não ocupa lugar no BD.
- Pode ser composta por colunas e agrupamentos de uma ou mais tabelas.
- Resumindo, é o resultado de uma seleção ou consulta.
- Ao se **alterar os dados das tabelas** que compõem a View, **altera-se** o resultado exibido pela **View**!

Por que usar Views?

- Porque fornecem um recurso de abreviação, pois tornam consultas complexas em simples.
 - ✓ Uma consulta com dezenas de linhas pode ser simplificada para um `SELECT * FROM VIEW`
- Porque permitem que os mesmos dados sejam vistos por usuários diferentes de modos diferentes ao mesmo tempo.
 - ✓ Os usuários podem se concentrar em apenas o que lhes interessa, ignorando o restante.
- Porque fornecem segurança automática para dados ocultos.
 - ✓ Os usuários não tem acesso aos dados que não estão disponíveis na Visão.

(DATE, 2004)

Views

- **Visões** que contenham **JOIN, GROUP BY, DISTINCT, alias** (para as colunas) **e expressões permitem apenas seleções**.
- As demais permitem inserir, excluir e atualizar.
- Sintaxe:

```
CREATE VIEW nome_da_view  
    [nome_da_coluna_1 [,nome_da_coluna2, nome_da_coluna_n]] AS  
SELECT ...
```

Views

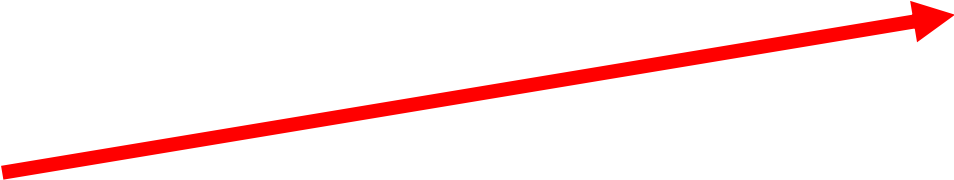
- Exemplo:

```
USE sales;
```

```
CREATE TABLE t (qtde INT, preco INT);  
INSERT INTO t VALUES (3, 50);
```

```
CREATE VIEW v AS
```

```
SELECT qtde, preco, qtde*preco AS valor  
FROM t;
```



qtde	preco	valor
3	50	150

```
SELECT * FROM v;
```

A view é dinâmica, se inserirmos outra linha e executar a view, ela vai mostrar o resultado com a nova linha.

Views

- Apagando uma View.

- Sintaxe:

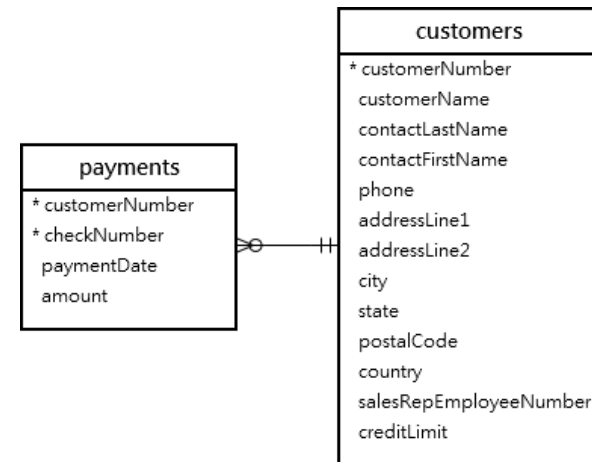
```
DROP VIEW nome_da_view;
```

```
DROP VIEW v;
```

Views

- Esta **consulta** retorna dados de ambas as tabelas *customers* e *payments* usando a junção interna.

```
USE classicmodels;  
SELECT customerName,  
       checkNumber,  
       paymentDate,  
       amount  
FROM customers  
INNER JOIN payments USING (customerNumber);
```



273 row(s) returned

customerName	checkNumber	paymentDate	amount
Atelier graphique	HQ336336	2004-10-19	6066.78
Atelier graphique	JM555205	2003-06-05	14571.44
Atelier graphique	OM314933	2004-12-18	1676.14
Signal Gift Stores	BO864823	2004-12-17	14191.12
Signal Gift Stores	HQ55022	2003-06-06	32641.98

Views

- Por definição, uma visão é uma **consulta nomeada armazenada** no catálogo do banco de dados.
- Esta declaração **cria** uma VIEW `customerPayments` baseada na consulta anterior:

```
CREATE VIEW customerPayments AS  
  SELECT customerName, checkNumber, paymentDate, amount  
  FROM customers  
  INNER JOIN payments USING (customerNumber);
```



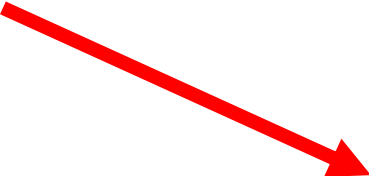
- Depois de executar a instrução CREATE VIEW, o MySQL **cria a VIEW e a armazena** no banco de dados.

Views

- Depois de criada a VIEW, você poderá consultar os dados usando uma instrução *SELECT*.

```
SELECT * FROM customerPayments;
```

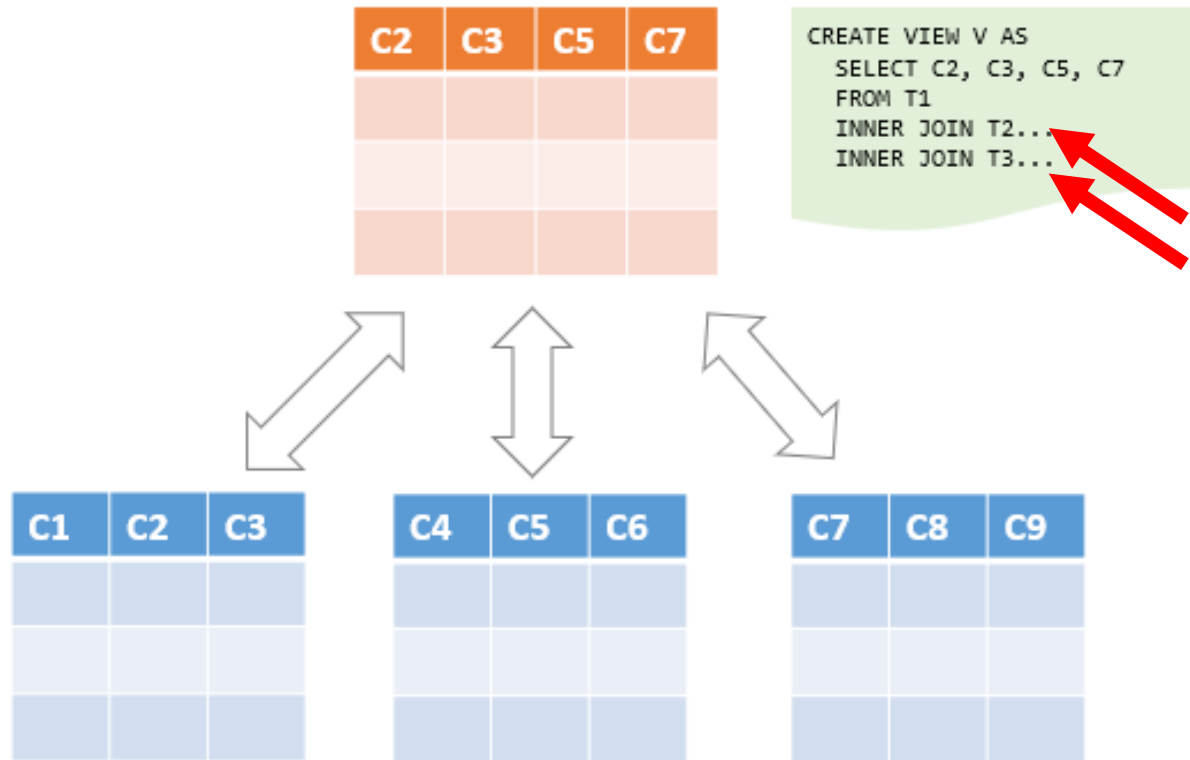
273 row(s) returned



customerName	checkNumber	paymentDate	amount
Atelier graphique	HQ336336	2004-10-19	6066.78
Atelier graphique	JM555205	2003-06-05	14571.44
Atelier graphique	OM314933	2004-12-18	1676.14
Signal Gift Stores	BO864823	2004-12-17	14191.12
Signal Gift Stores	HQ55022	2003-06-06	32641.98

Views

- Uma VIEW pode recuperar dados de mais de uma tabela.



Views

- Você pode criar uma VIEW chamada `diasdasemana` que retorna 7 dias da semana:

```
CREATE VIEW diasdasemana (dia) AS
```

```
    SELECT 'Seg' UNION
```

```
    SELECT 'Ter' UNION
```

```
    SELECT 'Qua' UNION
```

```
    SELECT 'Qui' UNION
```

```
    SELECT 'Sex' UNION
```

```
    SELECT 'Sab' UNION
```

```
    SELECT 'Dom';
```

```
SELECT * FROM diasdasemana;
```



dia
Seg
Ter
Qua
Qui
Sex
Sab
Dom

Referências

Referências

- DATE, C. J.. Introdução a Sistemas de Bancos de Dados. Rio de Janeiro: Campus, 2004. 865 p. Tradução da 8a Edição Americana.
- ELMASRI, Ramez; NAVATHE, Shamkant B.. Parte 2: Modelo de Dados Relacional e SQL: SQL Básica. In: ELMASRI, Ramez; NAVATHE, Shamkant B.. Sistema de Banco de Dados. 6. ed. São Paulo: Pearson Education, 2011. Cap. 4. p. 57-75.
- MICROSOFT. Quais são as funções do banco de dados SQL? 2017. Disponível em: <<https://docs.microsoft.com/pt-br/sql/t-sql/functions/functions?view=sql-server-2017>>. Acesso em: 27 jun. 2019.
- RAMAKRISHNAN, Raghu; GEHRKE, Johannes. SQL, Consultas, Restrições, Gatilhos. In: RAMAKRISHNAN, Raghu; GEHRKE, Johannes. Sistemas de Gerenciamento de Banco de Dados. 3. ed. São Paulo: Mc Graw Hill, 2008. Cap. 5. p. 110-147. Tradução da Terceira Edição.

Referências

- RANGEL, Alexandre Leite. DML: Data Manipulation Language. In: RANGEL, Alexandre Leite. InterBase 7: Desenvolvendo e Administrando Bancos de Dados. Rio de Janeiro: Alta Books, 2003. Cap. 2. p. 78-102.
- RANGEL, Alexandre Leite. LINGUAGEM SQL. Batatais: Claretiano, 2016. 97 p.
- RANGEL, Alexandre Leite. Manipulação de Dados em MySQL. In: RANGEL, Alexandre Leite. MySQL - Projeto, Modelagem e Desenvolvimento de Bancos de Dados. Rio de Janeiro: Alta Books, 2004. Cap. 6. p. 74-96.
- SILBERSCHATZ, Abraham; KORTH, Henry F.; SUNDARSHAN, S.. Introdução à SQL. In: SILBERSCHATZ, Abraham; KORTH, Henry F.; SUNDARSHAN, S.. Sistemas de Banco de Dados. 5. ed. Rio de Janeiro: Campus, 2006. Cap. 3. p. 37-67.

Referências

- W3SCHOOLS. MySQL String Functions. Disponível em: <https://www.w3schools.com/sql/sql_ref_mysql.asp>. Acesso em: 27 jun. 2019.
- W3SCHOOLS. SQL SELECT Statement. Disponível em: <https://www.w3schools.com/sql/sql_select.asp>. Acesso em: 27 jun. 2019.
- _____. SQL Server Functions. Disponível em: <https://www.w3schools.com/sql/sql_ref_sqlserver.asp>. Acesso em: 27 jun. 2019.

Obrigado!

Prof. Clóvis Ferraro
(Adaptado de Alexandre Rangel)