



Digital
College

FORMAÇÃO EM **DESENVOLVEDOR FULL STACK**

UNIDADE 3:
DESENVOLVENDO BACK-END

MÓDULO 3:
> BANCO DE DADOS >

Sumário

Introdução	05
1.1 Objetivos dos sistemas de dados	05
1.2 Abstração de dados	06
1.3 Modelos de dados	07
1.3.1. Modelos Lógicos Baseados em Objetos	08
1.3.2. Modelos Lógicos Baseados em Registro	09
1.3.3. Modelos Físicos de Dados	09
1.4. Instâncias e Esquemas	10
1.5. Independência dos Dados	11
1.6. Linguagem de Definição de Dados (DDL)	12
1.7. Linguagem de Manipulação de Dados (DML)	13
1.8. Gerenciador de Banco de Dados	14
1.9. Administrador do Banco de Dados (DBA)	16
1.10. Usuários de Banco de Dados	17
1.11. Estrutura Geral do Sistema de Gerenciamento de Banco de Dados	20
2. Linguagem SQL	21
2.1. A estrutura básica de uma expressão SQL	23
2.1.1. Cláusulas Distinct e All	26

2.2. Variáveis Tuplas (Renomeação)	28
2.3. Operações de Conjuntos	29
2.4. Exercícios	31
2.5. Ordenando a exibição de Tuplas	32
2.6. MEMBROS DE CONJUNTOS	33
2.7. FUNÇÕES AGREGADAS	36
2.8. MODIFICANDO O BANCO DE DADOS	38
2.9. Valores Nulos	41
3. DEFINIÇÃO DE DADOS	42
3.1 VISÕES (VIEWS)	44
3.2 RESTRIÇÕES DE INTEGRIDADE	46
3.2.1. Restrições de domínios	47
3.2.2. Restrições de integridade referencial	48
3.2.3. Asserções	51
3.3. EXERCÍCIOS	52
3.4. O ESQUEMA BANCO	58
3.5. O ESQUEMA EMPRESA	59
4. Instalação e Configuração MySQL Community	61
5. Introdução ao MySQL Workbench	72
5.1. Criando a Estrutura Física	75

5.2. Inserindo Dados	77
5.3. Consultando Dados	78

Um Sistema de Gerenciamento de Banco de Dados (SGBD) consiste em uma coleção de dados inter-relacionados e em um conjunto de programas para acessá-los.

Um Banco de Dados é um conjunto de dados inter-relacionados.

1. Introdução

1.1. Objetivos dos Sistemas de Banco de Dados

Um sistema que possui vários programas aplicativos acessando dados armazenados em arquivos convencionais apresenta uma série de problemas:

- **Redundância e inconsistência dos dados.** A redundância ocorre quando a mesma informação é mantida em diferentes arquivos, ou seja, quando há duplicação da informação. Isto pode gerar inconsistência dos dados, uma vez que podemos ter cópias diferentes do mesmo dado.
- **Dificuldade de acesso aos dados.** O ambiente convencional de processamento de arquivos não permite que dados necessários sejam recuperados de maneira conveniente e eficiente.
- **Isolamento dos dados.** Uma vez que os dados estão espalhados em diversos arquivos e podem ter formatos diferentes, é difícil escrever programas para recuperar os dados adequados.
- **Anomalias de acesso concorrente.** O acesso de vários usuários simultaneamente pode gerar dados inconsistentes no sistema.
- **Problemas com segurança.** Nem todo usuário deve ter acesso a todos os dados do sistema. Se programas aplicativos forem adicionados ao sistema de maneira arbitrária, é difícil assegurar restrições de segurança.

- **Problemas de integridade.** A integridade dos dados e do sistema é mantida pelos programas aplicativos, tornando-se, dessa forma, mais vulnerável a falhas. O problema torna-se mais complicado quando as restrições envolvem itens de dados e arquivos diferentes.

Esse grande número de problemas promoveu o desenvolvimento dos Sistemas Gerenciadores de Banco de Dados, que veio a eliminar todas estas complicações de forma eficiente.

1.2. Abstração dos Dados

O grande objetivo de um sistema de banco de dados é prover aos usuários uma visão abstrata dos dados, isto é, os detalhes de como os dados são armazenados e mantidos são escondidos do usuário. Não interessa ao usuário as complexas estruturas de armazenamento usadas para garantir a eficiência de acesso aos dados. Uma vez que muitos dos usuários de bancos de dados são leigos em computação, toda esta complexidade é escondida através de vários níveis de abstração que simplificam a interação do usuário com o sistema. Assim, os seguintes níveis são usados:

- **Nível físico** – o nível mais baixo de abstração descreve *como* os dados estão realmente armazenados. Num nível físico, complexas estruturas de dados de baixo nível são descritas em detalhe.
- **Nível conceitual** – este nível descreve *quais* dados estão armazenados de fato no banco de dados e as relações que existem entre eles. O nível conceitual é usado por administradores do banco de dados, que podem decidir quais informações devem ser mantidas no banco de dados.
- **Nível de visões** – a maioria dos usuários do sistema de banco de dados não está interessada em todas as informações existentes no banco de dados. Cada grupo de usuários deve enxergar somente os dados que lhe dizem respeito. Assim, cada grupo de usuários tem uma visão do banco de dados. O nível mais alto de abstração é composto de visões que cada grupo de usuários têm do banco de dados.

1.3. Modelos de Dados

Entende-se por Modelo de Dados uma coleção de ferramentas conceituais para descrição de dados, relacionamentos de dados, semântica de dados e restrições de consistência.

Há vários modelos de dados que podem ser divididos em três grupos:

- Modelos lógicos baseados em objetos
- Modelos lógicos baseados em registros
- Modelos físicos de dados

1.3.1. Modelos Lógicos Baseados em Objetos

- São usados na descrição de dados nos níveis conceitual e de visões.
- Fornecem, de modo conveniente, capacidades de estruturação flexíveis.
- Permitem especificar explicitamente restrições de dados.
- Existem vários modelos:
 - O modelo entidade-relacionamento
 - O modelo orientado a objetos.
 - O modelo binário.
 - O modelo semântico de dados
 - O modelo infológico.
 - O modelo funcional de dados.

O Modelo Entidade-Relacionamento (E-R)

O modelo entidade-relacionamento é baseado numa percepção de um mundo real que consiste em uma coleção de objetos básicos chamados *entidades*, e em *relacionamentos* entre esses objetos. Alguns conceitos desse modelo são:

- Entidades
- Relacionamentos
- Atributos
- Cardinalidade

A estrutura lógica geral de um banco de dados pode ser expressa graficamente por um *diagrama E-R*

O Modelo Orientado a Objetos

Os objetos do mundo real são agrupados em classes, de acordo com suas características comuns, e cada instância de uma classe é um objeto daquela classe.

- Objetos
- Classes
- Variáveis de instância
- Métodos
- Troca de mensagens
- Hierarquia de classes

1.3.2. Modelos Lógicos Baseados em Registro

- São também usados na descrição de dados nos níveis conceitual e de visões.
- O banco de dados é estruturado em registros de formato fixo de diversos tipos. Cada tipo de registro define um número fixo de campos, ou atributos, e cada campo é usualmente de um tamanho fixo.
- Não incluem um mecanismo para representação direta do código dentro do banco de dados.

O Modelo Relacional

- Representa dados e relacionamentos entre dados por um conjunto de tabelas, cada uma tendo um número de colunas com nomes únicos.

O Modelo de Redes

- Os dados no modelo rede são representados por coleções de registros, e os relacionamentos entre os dados são representados por ligações, que podem ser encaradas como ponteiros.
- Os registros no banco de dados são organizados como coleções de grafos arbitrários.

O Modelo Hierárquico

- É similar ao modelo de redes, pois os dados e relacionamentos são representados por registros e ligações, respectivamente. A diferença é que os registros são organizados como coleções de árvores em vez de grafos arbitrários.

1.3.3. Modelos Físicos de Dados

Os modelos físicos de dados são usados para descrever dados no nível mais baixo. Dois dos mais conhecidos são:

- Modelo unificador(Unifying Model)
- Estrutura de memória (frame memory)

1.4. Instâncias e Esquemas

Uma **instância do banco de dados** é uma coleção de informações armazenadas em um determinado momento.

O **esquema do banco de dados** é o projeto geral do banco de dados, ou seja, a definição de todas as estruturas que compõem o banco de dados.

Ex: Instâncias e esquemas em banco de dados são similares aos conceitos de variáveis e seus valores de linguagens de programação.

```
var x : integer;
```

> esquema

```
x := 10;
```

> instância (valor da variável em um dado instante)

1.5. Independência dos Dados

O termo **independência dos dados** diz respeito à habilidade de modificar a definição de um esquema em um nível sem afetar a definição do esquema num nível mais alto.

A independência dos dados pode ser de dois tipos:

- Independência física dos dados

Habilidade de modificar o esquema físico sem modificar os programas aplicativos.

- Independência lógica dos dados

Habilidade de modificar o esquema lógico sem modificar os programas aplicativos.

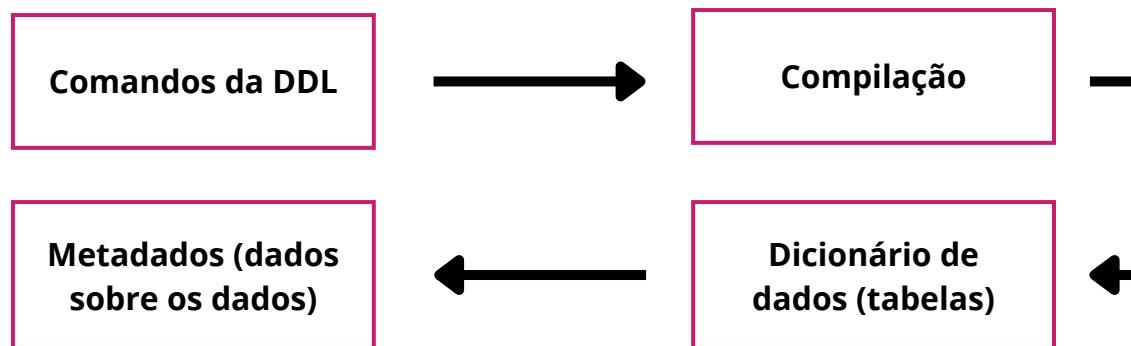
Obs.: A independência lógica é mais difícil de ser alcançada, uma vez que os programas são muito dependentes da estrutura lógica.

1.6. Linguagem de Definição de Dados (DDL)

Entende-se por Linguagem de Definição de Dados (DDL – Data Definition Language) o conjunto de comandos que permitem definir esquemas do banco de dados, ou seja, esta linguagem vai atuar na estrutura do banco de dados.

Entre as operações de definição de dados estão:

- criar tabelas, índices e visões,
- alterar a estrutura de uma tabela,
- remover tabelas, índices e visões.



Um tipo especial de DDL é a **Linguagem de Armazenagem e Definição de Dados** que permite a definição da estrutura de armazenamento e dos métodos de acesso do banco de dados.

1.7. Linguagem de Manipulação de Dados (DML)

Entende-se por **Linguagem de Manipulação de Dados (DML - Data Manipulation Language)** o conjunto de comandos que permitem acessar e manipular os dados, ou seja, esta linguagem vai atuar sobre os dados propriamente ditos armazenados no banco de dados.

Manipular significa:

- buscar informações
- inserir informações
- eliminar informações
- modificar informações

Há dois tipos de linguagens de manipulação de dados:

- DML's procedurais – o usuário especifica qual dado quer e diz como obtê-lo.
- DML's não-procedurais – o usuário especifica qual dado quer, **sem** dizer como obtê-lo.
 - Mais fáceis de aprender e usar.

Uma **consulta** é um comando requisitando a busca de uma informação.

A **linguagem de consulta** é a parte da DML que envolve a busca de informações.

1.8. Gerenciador de Banco de Dados

Os SGBD's trabalham com grandes quantidades de informação armazenada, na área dos GB (gigabyte) e TB (terabyte).

$1\text{ GB} \cong 1.000\text{ MB} \cong 1.000.000\text{ KB} \cong 1.000.000.000$ (1 bilhão de bytes)

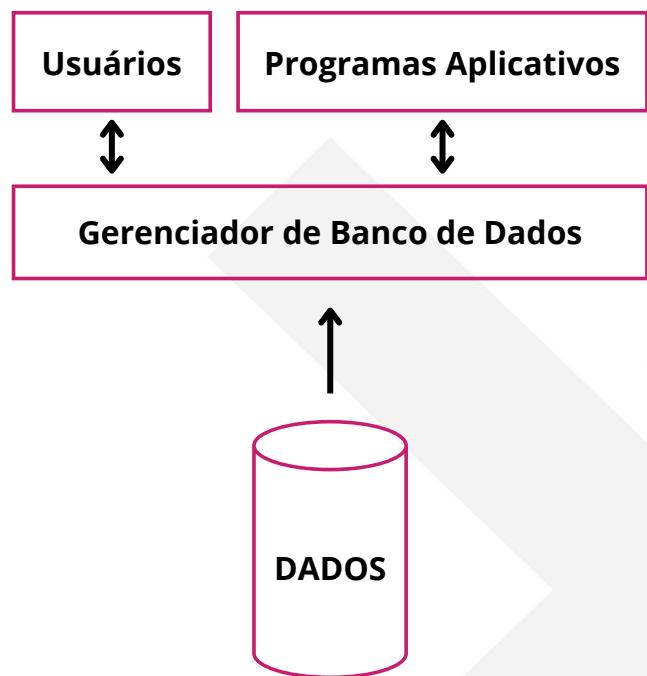
$1\text{ TB} \cong 1\text{ trilhão de bytes}$

Esta grande quantidade de informação deve ser acessada de maneira eficiente (desempenho). Para isso faz-se uso de complexas estruturas de dados. Estes dados são armazenados nos discos rígidos e, como a transferência de dados entre a memória principal e os discos é bastante lenta em comparação com a velocidade da CPU, o sistema de banco de dados deve estruturar os dados de forma a minimizar o movimento de dados entre os discos e a memória principal.

Deve-se ter em mente que o fator principal na satisfação de um usuário de banco de dados, bem como de qualquer sistema computacional, é seu **desempenho**. De nada adianta uma bela interface, se o sistema é lento. O desempenho de um sistema de banco de dados depende de dois fatores:

- da eficiência das estruturas de dados usadas para representar os dados
- de quão eficientemente o sistema é capaz de operar essas estruturas de dados.

O **Gerenciador de Banco de Dados** é um módulo de programa que faz parte do Sistema Gerenciador de Banco de Dados e provê a interface entre os dados de baixo nível armazenados num banco de dados e os programas de aplicação e as solicitações submetidas ao sistema.



O gerenciador de banco de dados é responsável pelas seguintes tarefas:

1. Interação com o gerenciador de arquivos do sistema operacional
 - o Armazenamento dos dados
 - o Busca dos dados
 - o Atualização dos dados
2. Cumprimento da integridade
3. Cumprimento da segurança (nem todo usuário precisa ver todo o banco de dados)
4. Cópias de reserva (backup) e recuperação
5. Controle de concorrência.

É importante notar que o porte do sistema de banco de dados depende do porte da empresa ou da aplicação. Sistemas de bancos de dados projetados para uso em pequenos computadores pessoais podem não ter todos os recursos descritos anteriormente.

1.9. Administrador do Banco de Dados (DBA)

Uma das principais razões para usar um banco de dados é a necessidade de ter controle central dos dados e dos programas de acesso aos dados.

O **Administrador de Banco de Dados (DBA - Database Administrator)** é a pessoa que centraliza o controle sobre os dados e programas de acesso aos dados num sistema de banco de dados.

As funções do administrador do banco de dados incluem:

- Definição de esquemas (usando DDL)
- Definição de estruturas de armazenamento e métodos de acesso
- Modificação de esquemas e da organização do armazenamento físico
- Concessão de autorização para acesso aos dados
- Especificação de restrições de integridade.

1.10. Usuários de Banco de Dados

Há, basicamente, quatro tipos de usuários de um sistema de banco de dados:

1. Programadores de aplicativos (ou de aplicação) – profissionais que escrevem programas em uma linguagem de programação qualquer, contendo, além dos comandos próprios da linguagem de programação, outros comandos de acesso ao banco de dados.

Estas linguagens de programação são chamadas de linguagens hospedeiras ou linguagens host (host languages) por hospedarem comandos de banco de dados estranhos ao seu ambiente.

Os comandos de banco de dados estranhos ao ambiente da linguagem host são denominados chamadas à DML embutidas, por estarem embutidas (inseridas) na linguagem host.

Uma chamada a um comando da DML embutida se faz usando um caractere (ou comando) especial, que varia de sistema para sistema. No Oracle, por exemplo, temos o comando EXEC SQL, que precede qualquer comando de acesso ao banco de dados. Veja o trecho de programa abaixo contendo uma chamada à DML embutida numa linguagem de programação tipo Pascal:

```
a := b + c;
```

> comando da linguagem hospedeira

```
EXEC SQL SELECT * FROM TABELA1;
```

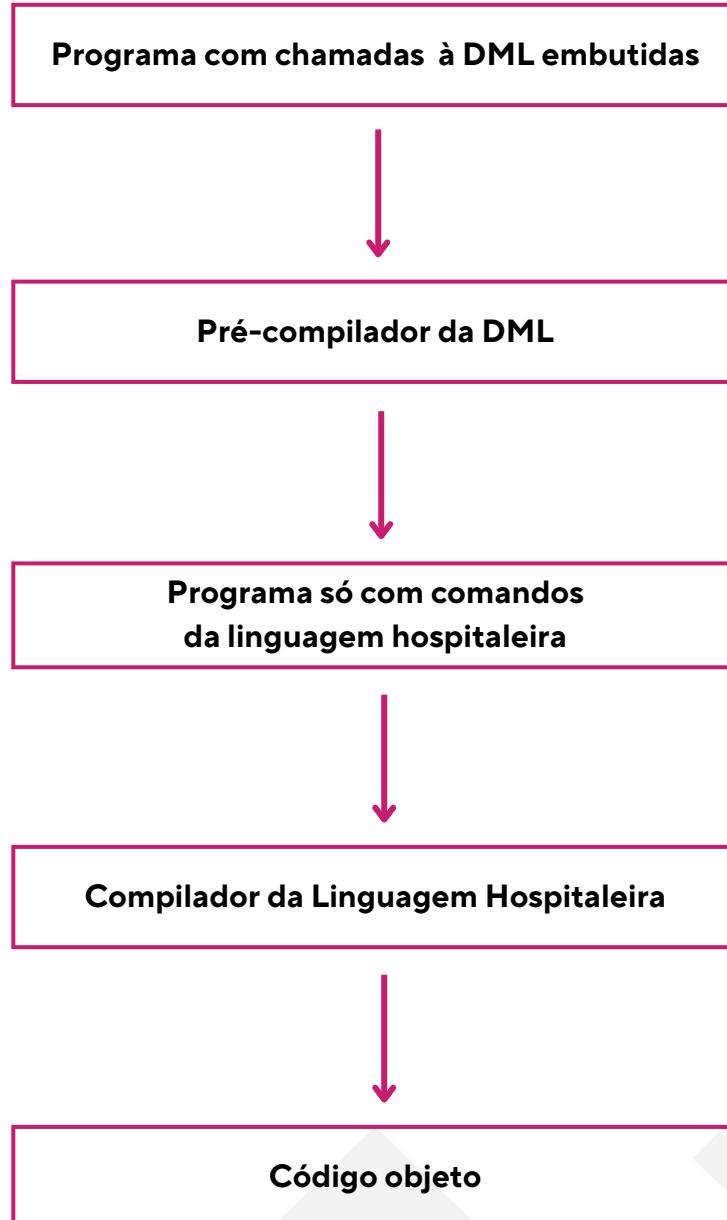
> chamada à DML embutida

```
while (a < d) do a := a + 1;
```

> outro comando da linguagem hospedeira

O processo de compilação procede-se da seguinte maneira:

- O programa com chamadas à DML embutidas passa pelo pré-compilador da DML, que transforma as chamadas à DML embutidas em chamadas de procedimentos normais na linguagem hospedeira;
- O programa pré-compilado, resultado da operação anterior, vai para o compilador da linguagem hospedeira, que efetua a compilação de forma normal, usando as bibliotecas do sistema de banco de dados, e gerando o código-objeto final.



Existem, atualmente, as chamadas linguagens de 4^a geração, que combinam as estruturas de controle das linguagens de programação com as estruturas de controle para manipulação de objetos do banco de dados. Tais linguagens são comumente usadas para criação de formulários (telas) e relatórios.

1. Usuários de alto nível – não escrevem programas, mas escrevem consultas em linguagens de consultas ao banco de dados.
2. Usuários especializados – escrevem programas especializados (CAD, CAM, sistemas especialistas, sistemas de dados complexos).
3. Usuários ingênuos ou leigos – utilizam os programas aplicativos.

1.11. Estrutura Geral do Sistema de Gerenciamento de Banco de Dados

Um sistema de gerenciamento de banco de dados é composto das seguintes partes:

- **Gerenciador de arquivos**

- Gerencia a alocação do espaço de armazenamento.
- Gerencia as estruturas de dados usadas para representar os dados armazenados no disco.

- **Gerenciador de banco de dados**

- Fornece a interface entre dados de baixo nível (em disco) e programas aplicativos e consultas de usuários.

- **Processador de consultas**

- Traduz comandos numa linguagem de consulta para instruções que o gerenciador de banco de dados entende.
- É responsável pela otimização de consultas.

- **Pré-compilador da DML**

- Traduz comandos da DML embutidos para chamadas de procedimento normais na linguagem hospedeira (linguagem "host").
- Interage com o processador de consultas.

- **Compilador da DDL**

- Converte comandos da DDL em tabelas contendo metadados.

Além desses componentes, algumas estruturas de dados são usadas para implementação do sistema físico:

- **Arquivos de dados** – armazenam o banco de dados propriamente dito.
- **Dicionário de dados** – armazenam os metadados, isto é, os dados sobre os dados.
- **Índices** – estrutura que proporciona acesso rápido aos dados.

2. Linguagem SQL

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas à sua manipulação. O Departamento de Pesquisas da IBM, desenvolveu a SQL como forma de interface para o sistema de BD relacional denominado SYSTEM R, no início dos anos 70. Em 1986 o American National Standard Institute (ANSI) publicou um padrão SQL. A SQL estabeleceu-se como linguagem padrão de Banco de Dados Relacional.

SQL apresenta uma série de comandos que permitem a definição dos dados, chamada de **DDL (Data Definition Language)**, composta entre outros pelos comandos Create, que é destinado a criação do Banco de Dados, das Tabelas que o compõe, além das relações existentes entre as tabelas. Como exemplo de comandos da classe DDL temos os comandos Create, Alter e Drop.

Os comandos da série **DML (Data Manipulation Language)**, destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML temos os comandos Select, Insert, Update e Delete. Uma subclasse de comandos DML, a **DCL (Data Control Language)**, dispõe de comandos de controle como Grant e Revoke.

A Linguagem SQL tem como grandes virtudes sua capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro (dos modelos Hierárquico e Redes). Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens independente das tabelas e organização lógica dos dados.

Outra característica interessante na linguagem SQL é a capacidade que dispomos de cancelar uma série de atualizações ou de as gravarmos, depois de iniciarmos uma sequência de atualizações. Os comandos Commit e Rollback são responsáveis por estas facilidades.

Existem inúmeras versões de SQL. A versão original foi desenvolvida no Laboratório de Pesquisa da IBM. Esta linguagem, originalmente chamada Sequel foi implementada como parte do projeto System R no início dos anos 70. A linguagem evoluiu desde então, e seu nome foi mudado para SQL (Structured Query Language).

Em 1986, o American National Standard Institute (ANSI) publicou um padrão SQL. A IBM publicou o seu próprio SQL standard, o *Systems Application Architecture Database Interface* (SAA-SQL) em 1987.

Em 1989, tanto ANSI quanto ISO publicaram padrões substitutos (ANSI X3.135-1989 e ISO/IEC 9075:1989) que aumentaram a linguagem e acrescentaram uma capacidade opcional de integridade referencial, permitindo que projetistas de bancos de dados pudessem criar relacionamentos entre dados em diferentes partes do banco de dados. A versão em uso do padrão ANSI / ISO SQL é o padrão SQL-92 (ANSI X3.135-1992) mas algumas aplicações atuais dão suporte apenas ao padrão SQL-89.

Desde 1993 há um trabalhos sendo desenvolvidos para atualizar o padrão de modo que este atenda às características das últimas versões de bancos de dados relacionais lançadas no mercado. A principal inovação da nova versão (chamada provisoriamente de SQL3) é o suporte à orientação a objetos.

Algumas das características da linguagem SQL são:

- Permite trabalhar com várias tabelas;
- Permite utilizar o resultado de uma instrução SQL em outra instrução SQL (sub-queries);
- Não necessita especificar o método de acesso ao dado;
- É uma linguagem para vários usuários como:
 - Administrador do sistema;
 - Administrador do banco de dados (DBA);
 - Programadores de aplicações;
 - Pessoal da agência e tomada de decisão;
 - É de fácil aprendizado;
- Permite a utilização dentro de uma linguagem procedural como C, COBOL, FORTRAN, Pascal e PL/I - SQL embutida.

2.1. A estrutura básica de uma expressão SQL

A estrutura básica de uma expressão SQL consiste em três cláusulas: **select**, **from** e **where**.

- A cláusula select corresponde à operação projeção da álgebra relacional. É usada para listar os atributos desejados no resultado de uma consulta.
- A cláusula from corresponde à operação produto cartesiano da álgebra relacional. Ela lista as relações a serem examinadas na avaliação da expressão.
- A cláusula where corresponde ao predicado de seleção da álgebra relacional. Consiste em um predicado envolvendo atributos de relações que aparecem na cláusula from.

Uma típica consulta SQL segue a seguinte ordem:

```
select a1, a2, ..., an    3ª
from T1, T2, ..., Tm    1ª
where P                  2ª
```

Cada a_i representa um atributo e cada T_i é uma relação e P é um predicado. Esta consulta é equivalente à expressão da álgebra relacional

```
p a1, a2, ..., an (s P (T1 x T2 x ... x Tm))
```

O SQL forma o produto cartesiano das relações chamadas na cláusula from, executa uma seleção da álgebra relacional usando o predicado da cláusula where e, então, projeta o resultado para os atributos da cláusula select. Na prática, o SQL pode converter esta expressão em uma forma equivalente que pode ser processada mais eficientemente.

Exemplo

No esquema BANCO, encontre os nomes de todos os clientes de 'Vitória'.

```
select cliente_nome
from CLIENTE
where cidade = 'Vitória'
```

Encontre os nomes de todos os clientes.

```
select cliente_nome
from CLIENTE
```

Exemplo

No esquema EMPRESA, selecionar o nome e o RG dos funcionários que trabalham no departamento número 2 na tabela EMPREGADO

```
select nome, rg
from EMPREGADOS
where depto = 2;
```

Obteremos então o seguinte resultado:

NOME	RG
FERNANDO	20202020
RICARDO	30303030
JORGE	40404040

A consulta acima é originária da seguinte função em álgebra relacional:

```
πnome, rg (σdepto = 2 (EMPREGADOS));
```

Em SQL também é permitido o uso de múltiplas condições. Veja o exemplo a seguir:

```
select nome, rg, salario
from EMPREGADOS
where depto = 2 AND salario > 2500.00;
```

Que fornece o seguinte resultado:

NOME	RG	SALÁRIO
Jorge	40404040	R\$4.200,00

e que é originária da seguinte função em álgebra relacional:

```
nome, rg, salario (σdepto = 2 .and. salario > 3500.00(EMPREGADOS));
```

O operador * dentro do especificador select seleciona todos os atributos de uma tabela, enquanto que a exclusão do especificador where faz com que todas as tuplas de uma tabela sejam selecionadas. Desta forma, a expressão:

```
select *
from empregado;
```

gera o seguinte resultado:

NOME	RG	CPF	DEPTO	RG SUPERVISOR	SALÁRIO
JOAO LUIZ	10101010	11111111	1	NULO	R\$3000,00
FERNANDO	20202020	22222222	2	10101010	R\$2500,00
RICARDO	30303030	33333333	2	10101010	R\$2300,00
JORGE	40404040	44444444	2	20202020	R\$4.200,00
RENATO	50505050	55555555	3	20202020	R\$1300,00

2.1.1. Cláusulas Distinct e All

Diferente de álgebra relacional, a operação select em SQL permite a geração de tuplas duplicadas como resultado de uma expressão. Para evitar isto, devemos utilizar a cláusula **distinct**.

Exemplo:

Sejam as seguintes consultas no esquema EMPRESA.

```
select depto
from EMPREGADO;
```

```
select distinct depto
from EMPREGADO;
```

DEPTO	DEPTO
1	1
2	2
2	3
2	
3	

que geram respectivamente os seguintes resultados:

A cláusula All é o default para o select, ou seja: Select All indica para obter todas as tuplas. Logo, esta cláusula não precisa ser colocada (a não ser, talvez por motivos de documentação).

Predicados e ligações

O SQL não tem uma representação da operação ligação natural. No entanto, uma vez que a ligação natural é definida em termos de um produto cartesiano, uma seleção e uma projeção, é relativamente simples escrever uma expressão SQL para uma ligação natural.

Exemplo

Encontre os nomes e cidades de clientes que possuam empréstimos em alguma agência.

```
Select distinct cliente_nome, cidade
From Cliente, Emprestimo
Where Cliente.cliente_cod=Emprestimo.cliente_cod
```

O SQL inclui os conectores **and**, **or** e **not**; caracteres especiais: (,), ., :, _, %<, >, <=, >=, =, <>, +, -,* e /; operador para comparação: between, como mostra o exemplo a seguir.

Exemplo

Selecionar todas as contas que possuam saldo entre 10000 e 20000.

```
Select conta_numero
From CONTA
Where saldo >= 10000 and saldo <= 20000
```

Que equivale à consulta

```
Select conta_numero
From CONTA
Where saldo between 10000 and 20000
```

A SQL inclui também um operador para comparações de cadeias de caracteres, o **like**. Ele é usado em conjunto com dois caracteres especiais:

- Por cento (%). Substitui qualquer subcadeia de caracteres.
- Sublinhado (_). Substitui qualquer caractere.

Exemplo

Encontre os nomes de todos os clientes cujas ruas incluem a subcadeia 'na'

```
Select distinct cliente_nome
From CLIENTE
Where rua like '%na%'
```

Exemplo

Encontre os nomes de todos os clientes cujas ruas finalizam com a subcadeia 'na', seguido de um caractere.

```
Select distinct cliente_nome
From CLIENTE
Where rua like '%na_'
```

Para que o padrão possa incluir os caracteres especiais (isto é, % , _ , etc...), a SQL permite a especificação de um caractere de escape. O caractere de escape é usado imediatamente antes de um caractere especial para indicar que o caractere especial deverá ser tratado como um caractere normal. Definimos o caractere de escape para uma comparação **like** usando a palavra-chave "escape". Para ilustrar, considere os padrões seguintes que utilizam uma barra invertida como caractere de escape.

- Like ' ab\%cd%' escape '\' substitui todas as cadeias começando com ' ab%cd';
- Like ' ab_cd%' escape '\' substitui todas as cadeias começando com ' ab_cd'.

OBS: A procura por não-substituições em vez de substituições dá-se através do operador **not like**.

2.2. Variáveis Tuplas (Renomeação)

Exemplo

No esquema EMPRESA, selecione o número do departamento que controla projetos localizados em Rio Claro;

```
select t1.numero_dept
from departamento_projeto as t1, projeto as t2
where t1.numero_projeto = t2.numero;
```

Na expressão SQL acima, t1 e t2 são chamados “alias” (apelidos) e representam a mesma tabela a qual estão referenciando. Um “alias” é muito importante quando há redundância nos nomes das colunas de duas ou mais tabelas que estão envolvidas em uma expressão. Ao invés de utilizar o “alias”, é possível utilizar o nome da tabela, mas isto pode ficar cansativo em consultas muito complexas além do que impossibilitaria a utilização da mesma tabela mais que uma vez em uma expressão SQL. A palavra-chave **“as”** é opcional.

Exemplo

No esquema EMPRESA, selecione o nome e o RG de todos os funcionários que são supervisores.

```
select e1.nome as "Nome do Supervisor", e1.rg as "RG do Supervisor"
from empregado e1, empregado e2
where e1.rg = e2.rg_supervisor;
```

que gera o seguinte resultado:

NOME DO SUPERVISOR	RG DO SUPERVISOR
João Luiz	10101010
Fernando	20202020

Observação: A consulta acima é decorrente da seguinte expressão em álgebra relacional:

```
\pi nome, rg (EMPREGADOS |x| tg_t1 = rg_supervisor_t2 EMPREGADOS)
```

Exemplo

Encontre o nome e a cidade de todos os clientes com uma conta em qualquer agência.

```
Select distinct C.cliente_nome, C.cidade
from CLIENTE C, CONTA S
where C.cliente_cod = S.cliente_cod
```

2.3. Operações de Conjuntos.

A SQL inclui as operações de conjunto **union**, **intersect** e **minus** que operam em relações e correspondem às operações , e – da álgebra relacional.

Uma vez que as relações são conjuntos, na união destas, as linhas duplicadas são eliminadas. Para que uma operação $R \cup S$, $R \cap S$ ou $R - S$ seja válida, necessitamos que duas condições sejam cumpridas:

- As relações R e S devem ter o mesmo número de atributos;
- Os domínios do i-ésimo atributo de R e do i-ésimo atributo de S devem ser os mesmos.

Observação: Nem todos os intérpretes SQL suportam todas as operações de conjunto. Embora a operação union seja relativamente comum, são raros os que suportam intersect ou minus.

Exemplo

Mostrar o nome dos clientes que possuem conta, empréstimo ou ambos na agência de código '051':
 Empréstimo na agência '051':

```
Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and agencia_cod = '051'
```

Conta na agência '051':

```
Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and CONTA.agencia_cod = '051'
```

Fazendo a união dos dois:

```
(Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and agencia_cod = '051' )
Union
(Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and CONTA.agencia_cod = '051')
```

Exemplo

Achar todos os clientes que possuam uma conta e um empréstimo na agência de código '051'.

```
(Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and agencia_cod = '051' )
intersect
(Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and CONTA.agencia_cod = '051')
```

Exemplo

Achar todos os clientes que possuem uma conta mas não possuem um empréstimo na agência de código '051'.

```
(Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod=Emprestimos.cliente_cod and
EMPRESTIMO.agencia_cod = '051' )
minus
(Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and CONTA.agencia_cod = '051'
)
```

2.4. Exercícios

Baseado no esquema EMPRESA, faça as seguintes consultas SQL.

1. Selecione todos os empregados com salário maior ou igual a 2000,00.
2. Selecione todos os empregados cujo nome começa com 'J'.
3. Mostre todos os empregados que têm 'Luiz' ou 'Luis' no nome.
4. Mostre todos os empregados do departamento de 'Engenharia Civil'.
5. Mostre todos os nomes dos departamentos envolvidos com o projeto 'Motor 3'.
6. Liste o nome dos empregados envolvidos com o projeto 'Financeiro 1'.
7. Mostre os funcionários cujo supervisor ganha entre 2000 e 2500.

Baseado no esquema BANCO, faça as seguintes consultas SQL.

1. Liste todas as cidades onde há agências.
2. Liste todas as agências existentes em 'Vitória'.
3. Liste todas as agências que possuem conta com saldo maior que 100.000,00.
4. Mostre os dados de todos os clientes da agência 'Centro' da cidade de 'Vitória' que têm saldo negativo.
5. Liste os dados dos clientes que possuem conta mas não possuem empréstimo na agência de código '005'.

2.5. Ordenando a exibição de tuplas (ORDER BY)

A cláusula **order by** ocasiona o aparecimento de tuplas no resultado de uma consulta em uma ordem determinada. Para listar em ordem alfabética todos os clientes do banco, fazemos:

```
Select distinct cliente_nome  
From CLIENTE  
Order by cliente_nome
```

Como padrão, SQL lista tuplas na ordem ascendente. Para especificar a ordem de classificação, podemos especificar **asc** para ordem ascendente e **desc** para descendente. Podemos ordenar uma relação por mais de um elemento. Como se segue:

```
Select *  
From EMPRESTIMO  
Order by quantia desc, agencia_cod asc
```

Para colocar as tuplas (linhas) em ordem é realizada uma operação de sort. Esta operação é relativamente custosa e portanto só deve ser usada quando realmente for necessário.

2.6. Membros de Conjuntos

O conectivo **in** testa os membros de conjunto, onde o conjunto é uma coleção de valores produzidos por uma cláusula select. Da mesma forma, pode ser usada a expressão **not in**.

Exemplo

Selecione todas as agências com código 1, 2 ou 3.

```
select *  
from agencia  
where agencia_cod in (1,2,3)
```

Exemplo

Selecione o nome de todos os funcionários que trabalham em projetos localizados em Rio Claro.

```
select e1.nome, e1.rg, e1.depto  
from empregado e1, empregado_projeto e2  
where e1.rg = e2.rg_empregado and e2.numero_projeto in (select numero  
from projeto  
where localizacao = 'Rio Claro');
```

Exemplo

Encontre todos os clientes que possuem uma conta e um empréstimo na agência 'Princesa Isabel'.

```
Select distinct cliente_nome  
From CLIENTE  
Where CLIENTE.cliente_cod in  
  
(select cliente_cod  
from CONTA, AGENCIA  
where CONTA.agencia_cod = AGENCIA.agencia_cod and agencia_nome =  
'Princesa Isabel')  
and CLIENTE.cliente_cod in  
  
(select cliente_cod  
  
from EMPRESTIMO, AGENCIA  
where EMPRESTIMO.agencia_cod= AGENCIA.agencia_cod and agencia_nome =  
'Princesa Isabel')
```

Exemplo

Encontre todas as agências que possuem ativos maiores que alguma agência de Vitória.

```
select distinct t.agencia_nome  
from AGENCIA t, AGENCIA s  
where t.ativos > s.ativos and s.cidade = 'Vitória'
```

Observação: Uma vez que isto é uma comparação "maior que", não podemos escrever a expressão usando a construção **in**.

A SQL oferece o operador **some** (equivalente ao operador **any**), usado para construir a consulta anterior. São aceitos pela linguagem: **>some, <some, >=some, <=some, =some**

```
select agencia_nome
from AGENCIA
where ativos > some
  (select ativos
   from AGENCIA
   where agencia_cidade = 'Vitória')
```

Como o operador **some**, o operador **all** pode ser usado como: **>all, <all, >=all, <=all, =all e <>all**. A construção **> all** corresponde à frase “maior que todos”.

Exemplo

Encontrar as agências que possuem ativos maiores do que todas as agências de Vitória.

```
select agencia_nome
from AGENCIA
where ativos > all
  (select ativos
   from AGENCIA
   where agencia_cidade = 'Vitória')
```

A SQL possui um recurso para testar se uma subconsulta tem alguma tupla em seus resultados. A construção **exists** retorna **true** se o argumento subconsulta está não-vazio. Podemos usar também a expressão **not exists**.

Exemplo

No esquema EMPRESA, liste o nome dos gerentes que têm ao menos um dependente.

```
Select nome
from EMPREGADO
where exists
  (select *
   from DEPENDENTES
   where DEPENDENTES.rg_responsavel = EMPREGADO.rg)
and exists
  (select *
   from DEPARTAMENTO
   where DEPARTAMENTO.rg_gerente = EMPREGADO.rg)
```

Exemplo

Usando a construção **exists**, encontre todos os clientes que possuem uma conta e um empréstimo na agência 'Princesa Isabel'.

```
Select cliente_nome
from CLIENTE
where exists
  (select *
   from CONTA, AGENCIA
   where CONTA.cliente_cod= CLIENTE.cliente_cod and AGENCIA.agencia_cod
        = CONTA.agencia_cod and agencia_nome = 'Princesa Isabel')
and exists
(select *
from EMPRESTIMO, AGENCIA
where EMPRESTIMO.cliente_cod= CLIENTE.cliente_cod and AGENCIA.agencia_cod =
EMPRESTIMO.agencia_cod and agencia_nome = 'Princesa Isabel')
```

2.7. Funções Agregadas

A SQL oferece a habilidade para computar funções em grupos de tuplas usando a cláusula **group by**. O(s) atributo(s) dados na cláusula group by são usados para formar grupos. Tuplas com o mesmo valor em todos os atributos na cláusula group by são colocados em um grupo. A SQL inclui funções para computar:

Média: **avg** Mínimo: **min** Máximo: **max** Soma: **sum** Contar: **count**

Exemplo

Encontre o saldo médio de conta em cada agência.

```
Select agencia_nome, avg(saldo)
From CONTA, AGENCIA
where CONTA.agencia_cod = AGENCIA.agencia_cod
Group by agencia_nome
```

Exemplo

Encontre o número de depositantes de cada agência.

```
Select agencia_nome, count(distinct cliente_nome)
From CONTA, AGENCIA
where CONTA.agencia_cod = AGENCIA.agencia_cod
Group by agencia_nome
```

Observação: Note que nesta última consulta é importante a existência da cláusula distinct pois um cliente pode ter mais de uma conta em uma agência, e deverá ser contado uma única vez.

Exemplo

Encontre o maior saldo de cada agência.

```
Select agencia_nome, max(saldo)
From CONTA, AGENCIA
Where CONTA.agencia_cod= AGENCIA.agencias_cod
Group by agencia_nome
```

Às vezes, é útil definir uma condição que se aplique a grupos em vez de tuplas. Por exemplo, poderíamos estar interessados apenas em agências nas quais a média dos saldos é maior que 1200. Esta condição será aplicada a cada grupo e não à tuplas simples e é definida através da cláusula **having**. Expressamos esta consulta em SQL assim:

```
Select agencia_nome, avg(saldo)
From CONTA, AGENCIA
Where CONTA.agencia_cod= AGENCIA.agencias_cod
Group by agencia_nome Having avg(saldo)>1200
```

Às vezes, desejamos tratar a relação inteira como um grupo simples. Nesses casos, não usamos a cláusula group by.

Exemplo

Encontre a média de saldos de todas as contas.

```
Select avg(saldo)  
From CONTA
```

2.8. Modificando o banco de dados.

• Remoção

Podemos remover somente tuplas inteiras, não podemos remover valores apenas em atributos particulares.

Sintaxe:

```
Delete R  
Where P
```

onde R representa uma relação e P um predicado.

Note que o comando **delete** opera em apenas uma relação. O predicado da cláusula **where** pode ser tão complexo como o predicado **where** do comando **select**.

Exemplo

Remover todas as tuplas de empréstimo.

```
delete EMPRESTIMO
```

Exemplo

Remover todos os registros da conta de 'João'.

```
delete CONTA  
where cliente_cod in  
(select cliente_cod  
from CLIENTE  
where cliente_nome = 'João')
```

Exemplo

Remover todos os empréstimos com números entre 1300 e 1500.

```
delete EMPRESTIMO  
where emprestimo_numero between 1300 and 1500
```

Exemplo

Remova todas as contas de agências localizadas em 'Vitória'.

```
Delete CONTA  
where agencia_cod in  
(select agencia_cod  
from AGENCIA  
where agencia_cidade='Vitoria')
```

- **Inserção**

Para inserir um dado em uma relação, ou especificamos uma tupla para ser inserida escrevemos uma consulta cujo resultado seja um conjunto de tuplas a serem inseridas. Os valores dos atributos para tuplas inseridas precisam necessariamente ser membros do mesmo domínio do atributo.

Exemplo

Inserir uma nova conta para João (código = 1), número 9000, na agência de código=2 cujo valor seja 1200.

```
insert into CONTA
values (2,9000,1,1200)
```

Na inserção acima é considerando a ordem na qual os atributos correspondentes estão listados no esquema de relação. Caso o usuário não se lembre da ordem destes atributos, pode fazer o mesmo comando da seguinte forma:

```
insert into CONTA (agencia_cod, conta_numero, cliente_cod, saldo)
values (2,9000,1,1200)
```

Podemos querer também inserir tuplas baseadas no resultado de uma consulta.

Exemplo

Inserir todos os clientes que possuam empréstimos na agência 'Princesa Isabel' na relação CONTA com um saldo de 200. O número da nova conta é o número do empréstimo * 10000.

```
insert into CONTA (agencia_cod, conta_numero, cliente_cod, saldo)
select AGENCIA.agencia_cod, emprestimo_numero*10000, cliente_cod, 200
from EMPRESTIMO, AGENCIA
where EMPRESTIMO.agencia_cod= AGENCIA.agencia_cod and agencia_nome =
'Princesa Isabel'
```

- **Atualizações**

Em certas situações, podemos desejar mudar um valor em uma tupla sem mudar todos os valores na tupla. Para isso, o comando **update** pode ser usado.

Suponha que esteja sendo feito o pagamento de juros, e que em todos saldos sejam acrescentados em 5%. Escrevemos

```
update CONTA
set saldo = saldo * 1,05
```

Suponha que todas as contas com saldo superiores a 10000 recebam aumento de 6% e as demais de 5%.

```
Update CONTA
set saldo = saldo * 1,06
where saldo >10000
```

```
Update CONTA
set saldo = saldo * 1,05
where saldo<=10000
```

A cláusula **where** pode conter uma série de comandos select aninhados. Considere, por exemplo, que todas as contas de pessoas que possuem empréstimos no banco terão acréscimo de 1%.

```
Update CONTA
set saldo = saldo * 1,01
where cliente_cod in
    (select cliente_cod
     from EMPRESTIMO )
```

Exemplo

No esquema EMPRESA, atualize o salário de todos os empregados que trabalham no departamento 2 para R\$3.000,00.

```
update empregado
set salario = 3000
where depto = 2;
```

Exemplo

No esquema BANCO, atualize o valor dos ativos. Os ativos são os valores dos saldos das contas da agência.

```
update agencia
set ativos =
    (select sum(saldo)
     from conta
     where conta.agencia_cod = agencia.agencia_cod)
```

2.9. Valores Nulos

É possível dar valores a apenas alguns atributos do esquema para tuplas inseridas em uma dada relação. Os atributos restantes são designados como nulos. Considere a requisição:

```
insert into CLIENTE (cliente_cod, cliente_nome, rua, cidade)
values (123, 'Andrea', null, null)
```

A palavra chave **null** pode ser usada em um predicado para testar se um valor é nulo. Assim, para achar todos os clientes que possuem valores nulos para rua, escrevemos:

```
select distinct cliente_nome
from CLIENTE
where rua is null
```

O predicado **is not null** testa a ausência de um valor nulo.

3. Definição de dados.

O conjunto de relações de um Banco de Dados precisa ser especificado ao sistema por meio de uma linguagem de definição de dados - DDL. A SQL DDL permite a especificação não apenas de um conjunto de relações, mas também de informações sobre cada relação, incluindo:

- Esquema para cada relação;
- Domínio de valores associados a cada atributo;
- Conjunto de índices a ser mantido para cada relação;
- Restrições de integridade;
- A estrutura física de armazenamento de cada relação no disco.

Uma relação SQL é definida usando o comando **create table**. A forma geral do comando **create table** então é:

```
create table <nome_tabela> (
    <nome_coluna1> <tipo_coluna1> <NOT NULL>,
    <nome_coluna2> <tipo_coluna2> <NOT NULL>,
    ...
    <nome_colunam> <tipo_colunam> <NOT NULL>);
```

A restrição **not null** indica que o atributo deve ser obrigatoriamente preenchido; se não for especificado, então o default é que o atributo possa assumir o valor nulo.

Exemplo

Criar a tabela EMPREGADO do esquema EMPRESA, teríamos o seguinte comando:

```
create table EMPREGADO
    (nome char (30) NOT NULL, rg integer NOT NULL, cic integer,
     depto integer NOT NULL, rg_supervisor integer, salario, decimal
      (7,2)NOT NULL)
```

O comando **create table** inclui opções para especificar certas restrições de integridade, conforme veremos.

A relação criada acima está inicialmente vazia. O comando insert poderá ser usado para carregar os dados para uma relação.

Para remover uma tabela de banco de dados SQL, usamos o comando **drop table**. O comando **drop table** remove todas as informações sobre a relação retirada do banco de dados. A forma geral para o comando **drop table** é:

```
drop table <nome_tabela>;
```

Exemplo

Eliminar a tabela EMPREGADO do esquema EMPRESA, teríamos o seguinte comando:

```
drop table EMPREGADOS;
```

Observe que neste caso, a chave da tabela EMPREGADOS, (rg) é utilizada como chave estrangeira ou como chave primária composta em diversos tabelas que devem ser devidamente corrigidas. Este processo não é assim tão simples pois, como vemos neste caso, a exclusão da tabela EMPREGADO implica na alteração do projeto físico de diversas tabelas. Isto acaba implicando na construção de uma nova base de dados.

O comando **alter table** é usado para alterar a estrutura de uma relação existente. Permite que o usuário faça a inclusão de novos atributos em uma tabela. A forma geral para o comando **alter table** é a seguinte:

```
alter table <tabela> <add, drop, modify> <coluna> <tipo_coluna>;
```

onde *add*, adiciona uma coluna; *drop*, remove uma coluna; e *modify*, modifica algo em uma tabela.

No caso do comando **alter table**, a restrição NOT NULL não é permitida pois assim que se insere um novo atributo na tabela, o valor para o mesmo em todas as tuplas da tabela receberão o valor NULL. Não é permitido eliminar algum atributo de uma relação já definida. Assim, caso você deseje eliminar uma chave primária devidamente referenciada em outra tabela como chave estrangeira, ao invés de obter a eliminação do campo, obterá apenas um erro.

3.1 Visões (Views)

Uma view em SQL é uma tabela que é derivada de outras tabelas ou de outras views. Uma view não necessariamente existe em forma física; é considerada uma tabela virtual (em contraste com as tabelas cujas tuplas são efetivamente armazenadas no banco de dados).

Uma view é definida usando o comando **create view**. Para definir uma visão, precisamos dar a ela um nome e definir a consulta que a processa.

A forma geral é:

```
create view <nomevisao> as <expressão de consulta>
```

onde, <expressão de consulta> é qualquer consulta SQL válida. Como exemplo, considere a visão consistindo em nomes de agências e de clientes.

```
Create view TOTOS_CLIENTES as
(select agencia_nome,cliente_nome
from CLIENTE, CONTA, AGENCIA
where CLIENTE.cliente_cod = CONTA.cliente_cod and CONTA.agencia_cod =
AGENCIA.agencia_cod)
union
(select agencia_nome,cliente_nome
from CLIENTE, EMPRESTIMO, AGENCIA
where CLIENTE.cliente_cod = EMPRESTIMO.cliente_cod and
EMPRESTIMO.agencia_cod = AGENCIA.agencia_cod)
```

Nomes de visões podem aparecer em qualquer lugar onde nome de relação (tabela) possa aparecer. Usando a visão TOTOS_CLIENTES, podemos achar todos os clientes da agência 'Princesa Isabel', escrevendo:

```
select cliente_nome
from TOTOS_CLIENTES
where agencia_nome = 'Princesa Isabel'
```

Uma modificação é permitida através de uma visão apenas se a visão em questão está definida em termos de uma relação do atual banco de dados relacional.

Exemplo

Selecionando tuplas de empréstimos.

```
Create view emprestimo_info as
(select agencia_cod, emprestimo_numero, cliente_cod
from EMPRESTIMO)
```

Uma vez que a SQL permite a um nome de visão aparecer em qualquer lugar em que um nome de relação aparece, podemos escrever:

```
insert into emprestimo_info
values (1,40,7)
```

Esta inserção é representada por uma inserção na relação EMPRÉSTIMO, uma vez que é a relação a partir do qual a visão emprestimo_info foi construída. Devemos, entretanto, ter algum valor para a quantia. Este valor é um valor nulo. Assim, o **insert** acima resulta na inserção da tupla: (1,40,7,null) na relação EMPRÉSTIMO.

Da mesma forma, poderíamos usar os comandos update, e delete.

Para apagar uma visão, usamos o comando

```
drop view <nomevisão>
```

Exemplo

Apagar a visão emprestimo_info.

```
drop view emprestimo_info
```

3.2. Restrições de integridade

As restrições de integridade fornecem meios para assegurar que mudanças feitas no banco de dados por usuários autorizados não resultem na inconsistência dos dados.

Há vários tipos de restrições de integridade que seriam cabíveis a bancos de dados. Entretanto as regras de integridade são limitadas às que podem ser verificadas com um mínimo de tempo de processamento.

3.2.1. Restrições de domínios

Um valor de domínio pode ser associado a qualquer atributo. Restrições de domínio são as mais elementares formas de restrições de integridade. Elas são facilmente verificadas pelo sistema sempre que um novo item de dados é incorporado ao banco de dados.

O princípio que está por trás do domínio de atributos é similar aos dos tipos em linguagem de programação. De fato é possível criar domínios em SQL através do comando **create domain**.

A forma geral é:

```
CREATE DOMAIN nome_do_domínio tipo_original
[ [ NOT ] NULL ]
[ DEFAULT valor_default ]
[ CHECK ( condições ) ]
```

Exemplo

Cria o tipo_codigo como um número de 3 algarismos com valores permitidos de 100 a 800.

```
create domain tipo_codigo numeric(3)
not null
check ((tipo_codigo >= 100) and (tipo_codigo <= 800))
```

A cláusula **check** da SQL-92 permite modos poderosos de restrições de domínio. Ela pode ser usada também no momento de criação da tabela.

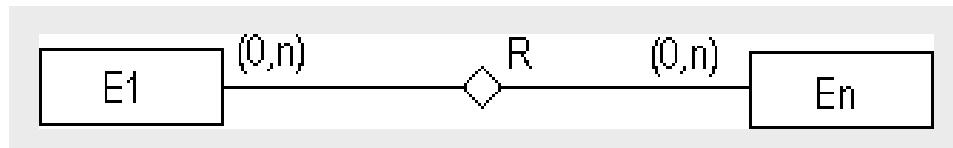
Exemplo

```
create table dept
(deptno integer(2) not null, dname char(12),
loc char(12),
check (dptono >= 100))
```

3.2.2. Restrições de integridade referencial

Muitas vezes, desejamos assegurar que um valor que aparece em uma relação para um dado conjunto de atributos, apareça também para um certo conjunto de atributos em outra relação. Isto é chamado de Integridade Referencial.

Neste relacionamento, considerando **K1** como sendo chave primária de **E1** e **Kn** como sendo chave primária de **En**, temos em **R** tuplas que serão identificadas inclusive por **K1** e **Kn** (na forma de chaves estrangeiras). Desta forma não poderá existir um elemento **K1** em **R** que não faça parte de **E1**, tão pouco um **Kn** em **R** que não faça parte de **En**.



As modificações no banco de dados podem causar violações de integridade referencial. Considerações devem ser feitas ao inserir, remover e atualizar tuplas.

Considerar: $\pi_a(R2) \subseteq \pi_k(R1)$

- **Inserir.** Se uma tupla t_2 é inserida em R_2 , o sistema precisa assegurar que existe uma tupla t_1 em R_1 tal que $t_1[k] = t_2[a]$. Isto é, $t_2[a] \in \pi_k(R1)$
- **Remover.** Se uma tupla t_2 é removida de R_2 , o sistema precisa computar o conjunto de tuplas em R_2 que referencia t_1 :

$$\sigma_a(R1) = t1[k](R2)$$

Se este conjunto não for vazio, o comando remover é rejeitado como um erro, ou as tuplas que se referem a t_1 precisam que elas mesmas sejam removidas. A última solução pode levar a uma remoção em cascata, uma vez que as tuplas precisam referir-se a tuplas que se referem a t_1 e assim por diante.

- **Atualizar.** Precisamos considerar dois casos para atualização: a atualização da relação referenciadora (filha - R_2) e atualização da relação referenciada (pai - R_1).

Se a tupla t_2 é atualizada na relação R_2 e a atualização modifica valores para a chave estrangeira a , então é feito um teste similar para o caso de inserção. Digamos que t_2' denote o novo valor da tupla t_2 . O sistema precisa assegurar que:

$$t2' [a] \in \pi_k(R1)$$

Se a tupla t_1 é atualizada em R_1 , e a atualização modifica valores para a chave primária (K), então é feito um teste similar ao caso remover. O sistema precisa computar

$$\sigma_a(R1) = t1[k](R2)$$

A SQL permite a especificação de chaves primárias, candidatas e estrangeiras como parte da instrução **create table**.

- A cláusula **primary key** da instrução create table inclui uma lista de atributos que compreende a chave primária;
- A cláusula **unique key** da instrução create table inclui uma lista de atributos que compreende a chave candidata;
- A cláusula **foreign key** da instrução create table inclui uma lista de atributos que compreende a chave estrangeira e o nome da relação referida pela chave estrangeira.

Exemplo

Criar as relações cliente, agência e conta para o esquema do banco.

```
Create table CLIENTE
(cliente_cod integer not null,
cliente_nome char(30),
rua char(30),
cidade char(30),
primary key (cliente_cod))

Create table AGENCIA
(agencia_cod integer not null,
agencia_nome char(30),
agencia_cidade char(30),
fundos decimal (7,2),
primary key (agencia_cod),
check (fundos >= 0))

Create table CONTA
(agencia_cod int,
conta_numero char(10) not null,
cliente_cod int not null, saldo
decimal (7,2),
primary key (cliente_cod,conta_numero), foreign
key (cliente_cod) references clientes, foreign
key (agencia_cod) references agencias, check
(fundos >= 0))
```

Notem que os atributos chaves precisam ter a especificação de **not null**.

Quando uma regra de integridade referencial é violada, o procedimento normal é rejeitar a ação que ocasionou essa violação. Entretanto é possível criar ações para modificação das tabelas associadas onde houve (ou haveria) a quebra de integridade referencial.

Isto é feito através das cláusulas **on delete cascade** e **on update cascade** associadas à cláusula **foreign key**.

Exemplo

```
Create table CONTA
(...

foreign key (agencia_cod) references agencia)
on delete cascade on update cascade,
...)
```

Neste exemplo, se a remoção de uma tupla da tabela agência resultar na violação da regra de integridade, o problema é resolvido removendo as tuplas associadas com esta agência da tabela conta. De forma semelhante, se o código de uma agência for alterado, as contas desta agência serão também alteradas.

3.2.3. Asserções

Uma asserção (afirmação) é um predicado expressando uma condição que desejamos que o banco de dados sempre satisfaça. Quando uma assertiva é criada, o sistema testa sua validade. Se a afirmação é válida, então qualquer modificação posterior no banco de dados será permitida apenas quando a asserção não for violada.

Restrições de domínio e regras de integridade referencial são formas especiais de asserções.

O alto custo de testar e manter afirmações têm levado a maioria de desenvolvedores de sistemas a omitir o suporte para afirmações gerais. A proposta geral para a linguagem SQL inclui uma construção de proposta geral chamada instrução **create assertion** para a expressão de restrição de integridade. Uma afirmação pertencente a uma única relação toma a forma:

```
create assertion <nome da asserção> check <predicado>
```

Exemplo

Definir uma restrição de integridade que não permita saldos negativos.

```
create assert saldo_restricao
check (not exists (select * from CONTA where saldo < 0))
```

Exemplo

Só permitir a inserção de saldos maiores que quantias emprestadas para aquele cliente.

```
create assert saldo_restricao2
check (not exists
(select * from
conta where
saldo <
(select max(quantia)
from EMPRESTIMO
where EMPRESTIMO.cliente_cod = CONTA.cliente_cod)))
```

3.3. Exercícios.

Considerando o esquema:

```
Pessoas = (CodPessoa, NomePessoa, Cidade, Chefe)
Empresas = (CodEmpresa, NomeEmpresa, Cidade)
Trabalha = (CodPessoa, CodEmpresa, Salario)
```

1. Consulte todas as pessoas que trabalham em Vitória.

```
Select Pessoas.NomePessoa
from Pessoas
where Cidade = 'Vitória'
```

2. Consulte todas as pessoas que trabalham na mesma cidade onde moram.

```
Select Pessoas.NomePessoa
from Pessoas, Trabalha, Empresas
where Pessoas.CodPessoa = Trabalha.CodPessoa and
      Trabalha.CodEmpresa = Empresas.CodEmpresa and
      Pessoa.Cidade = Empresas.Cidade
```

3. Consulte todas as pessoas que moram na mesma cidade do chefe.

```
Select Pessoas.NomePessoa
from Pessoas, Pessoas_Chefs
where Pessoas.Chefe = Chefs.CodPessoa and
      Pessoa.Cidade = Chefs.Cidade
```

4. Consulte todas as empresas que funcionam em cidades que não moram pessoas cujo primeiro nome seja Maria (usar operações de conjunto).

```
Select Empresas.NomeEmpresa
from Empresas
Where Empresas.Cidade not in
      (select Cidade
       from Pessoas
       Where Pessoas.NomePessoa like 'Maria%')
```

5. Consulte todas as pessoas que não trabalham em Vitória e que ganham acima de 2000 em ordem decrescente da cidade onde moram.

```
Select Pessoas.NomePessoa
from Pessoas, Trabalha, Empresas
where Pessoas.CodPessoa = Trabalha.CodPessoa and
      Trabalha.CodEmpresa = Empresas.CodEmpresa and
      Empresas.Cidade < > 'Vitória' and Salario > 2000
order by pessoa.cidade desc
```

6. Consulte todas as pessoas que não trabalham na empresa que comece com 'inf_' em ordem alfabética.

```
Select Pessoas.NomePessoa
from Pessoas, Trabalha, Empresas
where Pessoas.CodPessoa = Trabalha.CodPessoa and
      Trabalha.CodEmpresa = Empresas.CodEmpresa and
      Empresas.nomeEmpresa not like 'inf%_%' escape '&'
order by nomeEmpresa
```

Considere o esquema seguinte para as questões que se seguem.

```
Fabricante =(codf, nomef)
Automovel =(coda, nomea, preço, codf)
Pessoa =(codp, nomep)
Venda =(codp, coda, valor, cor, data)
```

7. Quem comprou 'Ford'? (fazer duas consultas).

```
select Pessoa.NomeP
from Pessoa, venda
where Pessoa.codp = Venda.codp and
      venda.coda in
          (select coda
             from Automovel, Fabricante
            where Automovel.codf = Fabricante.codf
              and Fabricante.Nomef = 'Ford')
```

Outra resposta:

```
select Pessoa.NomeP
from Pessoa, venda, Automovel, Fabricante
where Pessoa.codp = Venda.codp and
      venda.coda = Automovel.coda and
      Automovel.codf = Fabricante.codf and
      Fabricante.Nomef = 'Ford'
```

8. Quem não comprou 'Ford'?

```
select Pessoa.NomeP
from Pessoa, venda
where Pessoa.codp = Venda.codp and
      venda.coda not in
          (select coda
             from Automovel, Fabricante
            where Automovel.codf = Fabricante.codf and
              Fabricante.Nomef = 'Ford')
```

9. Quem comprou carro com ágio?

```
select Pessoa.NomeP
from Pessoa, venda, Automovel
where Pessoa.codp = Venda.codp and
      venda.coda = Automovel.coda and
      venda.valor > Automovel.preço
```

10. Quem comprou 'Ford' e não comprou 'Volks'?

```

select Pessoa.NomeP
from Pessoa, venda
where Pessoa.codp = Venda.codp

        and venda.coda in

        (select coda
         from Automovel, Fabricante
         where Automovel.codf = Fabricante.codf
         and Fabricante.Nomef = 'Ford')
and venda.coda not in
        (select coda
         from Automovel, Fabricante
         where Automovel.codf = Fabricante.codf
         and Fabricante.Nomef = 'Volks')
    
```

11. Quem comprou carro entre 01/01/97 e 01/01/98?

```

select Pessoa.NomeP
from Pessoa, venda
where Pessoa.codp = Venda.codp and
venda.data between '01/01/97' and '01/01/98'
    
```

Considere o esquema BANCO, reproduzido abaixo.

```

Agencias = (agencia_cod, agencia_nome, agencia_cidade, ativos)
Clientes = (cliente_cod, cliente_nome, rua, cidade)
Depositos = (agencia_cod, conta_numero, cliente_cod, saldo)
Emprestimos = (agencia_cod, cliente_cod, emprestimo_numero, quantia)
    
```

12. Selecione todos os clientes que possuem contas em agência(s) que possui(em) o maior ativo.

```

Select cliente_nome
from clientes, depositos
where clientes.cliente_cod = depositos.cliente_cod and
depositos.agencia_cod in
(select agencia_cod
from agencias
where ativos >= all
(select ativos
from agencias))
    
```

13. Selecione o total de agências por cidade, classificado por cidade.

```

Select cidade, count(agencia_cod) as total_agencias
from agencias
group by cidade
order by cidade
    
```

14. Selecione, por agências, o(s) cliente(s) com o maior saldo.

```
Select agencia_nome, cliente_nome, saldo
from clientes, depositos, agencias
where clientes.cliente_cod = depositos.cliente_cod and
      depositos.agencia_cod = agencia.agencia_cod and
      <agencia_cod, saldo> in
          (Select agencia_cod, max(saldo) as maior_saldo
           from depositos
           group by agencia_cod)
```

15. Selecione o valor médio de empréstimos efetuados por cada agência em ordem crescente das cidades onde estas agências se situam.

```
Select agencia_nome, avg(quantia) as valor_medio
from emprestimos, agencias
where emprestimos.agencia_cod = agencias.agencia_cod
group by agencia_nome
order by cidade
```

16. Selecione a(s) agência(s) que possui(m) a maior média de quantia emprestada.

```
Select agencia_nome
from agencias, emprestimos
where emprestimos.agencia_cod = agencias.agencia_cod
group by agencia_nome
having avg(quantia) >= all
       (select avg(quantia)
        from emprestimos
        group by agencia_cod)
```

17. Selecione todas as agências situadas fora de Vitória que possuem a média de depósitos maior do que alguma agência localizada em Vitória.

```
Select agencia_nome
from agencias, depositos
where depositos.agencia_cod = agencias.agencia_cod and
      agencia.cidade <> 'Vitória'
group by agencia_nome
having avg(saldo) > some
       (select avg(saldo)
        from depositos, empresas
        where depositos.codEmpresa = empresas.codEmpresa and
              empresas.cidade='Vitória'
        group by agencia_cod)
```

18. Selecione o menor saldo de clientes, por agências.

```
Select agencia_nome, min(saldo) as menor_saldo
from depositos, agencias
where depositos.agencia_cod = agencias.agencia_cod
group by agencia_cod
```

19. Selecione o saldo de cada cliente, caso ele possua mais de uma conta no banco.

```
Select cliente_nome, sum(saldo)
from clientes, depositos
where clientes.cliente_cod = depositos.cliente_cod
group by cliente_nome
having count(agencia_cod) >1
```

Considere o esquema abaixo para as questões que se seguem

```
Pessoas = (CodPessoa, NomePessoa, Cidade, Chefe)
Empresas = (codEmpresa, NomeEmpresa, Cidade)
Trabalha = (CodPessoa, CodEmpresa, Salario)
```

20. Excluir todas as pessoas que possuem salário = 1000.

```
Delete pessoas where CodPessoa in
(select cod_pessoa from trabalha where salario = 1000)
```

21. Excluir todas as pessoas que trabalham em empresas situadas em Vitória.

```
Delete pessoas where CodPessoa in
(select cod_pessoa from trabalha, Empresas
where trabalha.CodEmpresa=Empresas.CodEmpresa and cidade='Vitoria')
```

22. Incluir na empresa de código '01' todos os moradores de Vitória com um salário=100.

```
Insert into trabalha (codPessoa, codEmpresa, Salario)
Select codpessoa, '01', 100 from pessoas where cidade = 'Vitória'
```

23. Uma determinada empresa de código 'x' vai contratar todos os funcionários da empresa de código 'y' que ganham acima de 1000, dando um aumento de salário de 10%. Faça comando(s) SQL para que tal transação seja efetuada.

```
Insert into trabalha (codPessoa, codEmpresa, Salario)
Select codpessoa, 'x', salario*1.1
from trabalha
where codempresa='y' and salario >1000
```

```
Delete trabalha where codempresa = 'y' and salario >1000
```

24. Uma determinada empresa de código 'xyz' quer contratar todos que moram em Vitória e estão desempregados. Serão contratados com salário = 200. Faça comando(s) SQL para fazer tal transação.

```
Insert into trabalha (codPessoa, codEmpresa, Salario)
Select codp, 'xyx', 200
from pessoas
where cidade = 'Vitória' and codpessoa
not in (select codpessoa from trabalha)
```

25. Fazer um comando SQL para ajustar o salário de todos os funcionários da empresa 'Campana' em 5%.

```
Update trabalha
Set salario = salario * 1.05
Where codEmpresa in
  (select codEmpresa
   from empresas
   where nomeEmpresa = 'Campana')
```

26. Todas as pessoas que moram em Colatina e trabalham na empresa 'Campana' deverão mudar para Vitória, devido aos requisitos do diretor da empresa. Faça o comando(s) SQL para fazer esta atualização da cidade.

```
Update Pessoas
Set cidade = 'Vitória'
Where codPessoa in
  (select codPessoa
   from Pessoas, Trabalha, empresas
   where Pessoas.codPessoa = Trabalha.CodPessoa and
         Pessoas.cidade = 'Colatina' and
         Trabalha.CodEmpresa = Empresas.codEmpresa and
         nomeEmpresa = 'Campana')
```

3.4. O Esquema Banco

AGÊNCIA

```
= (agencia_cod, agencia_nome, agencia_cidade, ativos)
```

CLIENTE

```
= (cliente_cod, cliente_nome, rua, cidade)
```

CONTA

```
= (agencia_cod, conta_numero, cliente_cod, saldo)
```

EMPRÉSTIMO

```
= (agencia_cod, cliente_cod, emprestimo_numero, quantia)
```

De uma forma visual:

AGÊNCIA	CONTA	EMPRÉSTIMO	CLIENTE
agencia_cidade	cliente_cod	cliente_cod	cliente_cod
agencia_nome	agencia_cod	emprestimo_numero	cliente_nome
agencia_cod	conta_numero	quantia	rua
ativos	saldo	agencia_cod	cidade

3.5. O Esquema Empresa

Tabela EMPREGADO

Nome	RG	CPF	Deptº	RG Supervisor	Salário
João Luiz	10101010	11111111	1	NULO	R\$3000,00
Fernando	20202020	22222222	2	10101010	R\$2500,00
Ricardo	30303030	33333333	2	10101010	R\$2300,00
Jorge	40404040	44444444	2	20202020	R\$4200,00
Renato	50505050	55555555	3	20202020	R\$1300,00

Tabela DEPARTAMENTO

Nome	Número	RG Gerente
Contabilidade	1	10101010
Engenharia Civil	2	30303030
Engenharia Mecânica	3	20202020

Tabela PROJETO

Nome	Número	Localização
Financeiro 1	5	São Paulo
Motor 3	10	Rio Claro
Prédio Central	20	Campinas

Tabela DEPENDENTES

RG Responsável	Nome Dependente	Nascimento	Relação	Sexo
10101010	Jorge	27/12/86	Filho	Masculino
10101010	Luiz	18/11/79	Filho	Masculino
20202020	Fernanda	14/02/69	Cônjuge	Feminino
20202020	Angelô	10/02/95	Filho	Masculino
30303030	Andreia	01/05/90	Filho	Feminino

Tabela DEPARTAMENTO PROJETO

Número Depto	Número Projeto
2	5
3	10
2	20

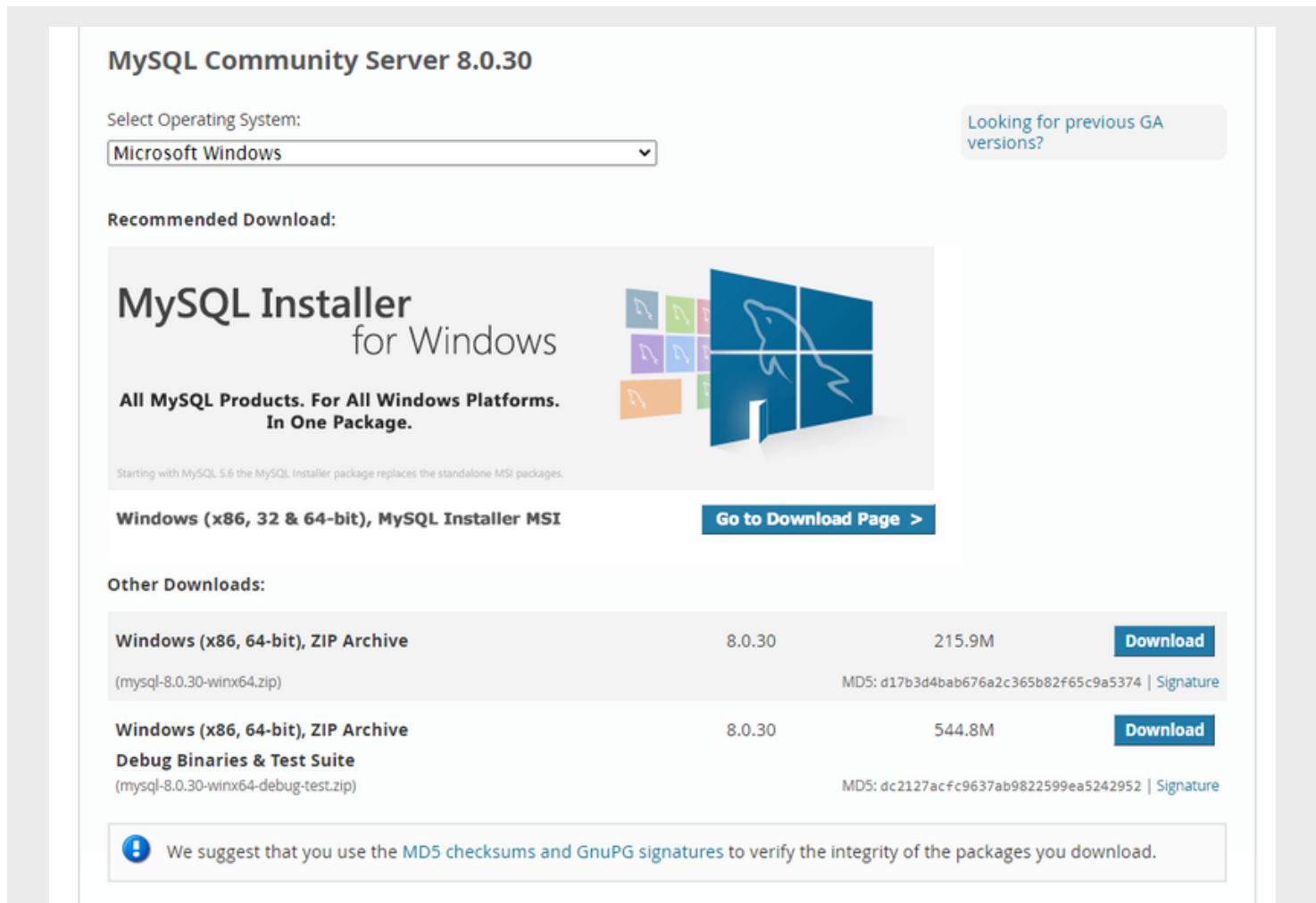
Tabela EMPREGADO PROJETO

RG Empregado	Número Projeto	Horas
20202020	5	10
20202020	10	25
30303030	5	35
40404040	20	50
50505050	20	35

4. Instalação e Configuração MySQL Community

MySQL Community Server (ou Community Edition) é o que muitos têm em mente quando pensam no MySQL. Ele oferece os recursos básicos que você espera de um sistema de gerenciamento de banco de dados, incluindo a capacidade de criar tabelas, visualizações, gatilhos e procedimentos armazenados.

Aqui temos o link para download: <https://dev.mysql.com/downloads/mysql/>



MySQL Community Server 8.0.30

Select Operating System:

Microsoft Windows

Looking for previous GA versions?

Recommended Download:

MySQL Installer
for Windows

All MySQL Products. For All Windows Platforms.
In One Package.

Starting with MySQL 5.6 the MySQL Installer package replaces the standalone MSI packages.

Windows (x86, 32 & 64-bit), MySQL Installer MSI

Go to Download Page >

Other Downloads:

Download Type	Version	File Size	Action
Windows (x86, 64-bit), ZIP Archive (mysql-8.0.30-winx64.zip)	8.0.30	215.9M	Download
Windows (x86, 64-bit), ZIP Archive Debug Binaries & Test Suite (mysql-8.0.30-winx64-debug-test.zip)	8.0.30	544.8M	Download

! We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

Figura 01: Tela inicial – Clicar em “Go to download Page”

④ MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

[Login »](#)

using my Oracle Web account

[Sign Up »](#)

for an Oracle Web account

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

[No thanks, just start my download.](#)

Figura 02: Caso não queira realizar o login clique em “no thanks, just start my download”

④ MySQL Community Downloads

To begin your download, please click the Download Now button below.

[Download Now »](#)

mysql-8.0.30-winx64.zip

MD5: d17b3d4bab676a2c365b82f65c9a5374

Size: 215.9M

[Signature](#)

ORACLE © 2022 Oracle

[Privacy / Do Not Sell My Info](#) | [Terms of Use](#) | [Trademark Policy](#) | [Preferências de Cookies](#)

Figura 03: Download Now

MySQL Community Downloads

MySQL Installer

General Availability (GA) Releases Archives

MySQL Installer 8.0.30

Select Operating System: Microsoft Windows

Looking for previous GA versions?

File Type	Version	File Size	Action
Windows (x86, 32-bit), MSI Installer	8.0.30	5.5M	Download
(mysql-installer-web-community-8.0.30.0.msi)			MD5: c095cf221e8023fd8391f81eadce65fb Signature
Windows (x86, 32-bit), MSI Installer	8.0.30	448.3M	Download
(mysql-installer-community-8.0.30.0.msi)			MD5: c9cbd5d788f45605dae914392a1dFeea Signature

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

Figura 04: Escolha a primeira opção

MySQL Installer

Choosing a Setup Type

Please select the Setup Type that suits your use case.

Developer Default
 Installs all products needed for MySQL development purposes.

Server only
 Installs only the MySQL Server product.

Client only
 Installs only the MySQL Client products, without a server.

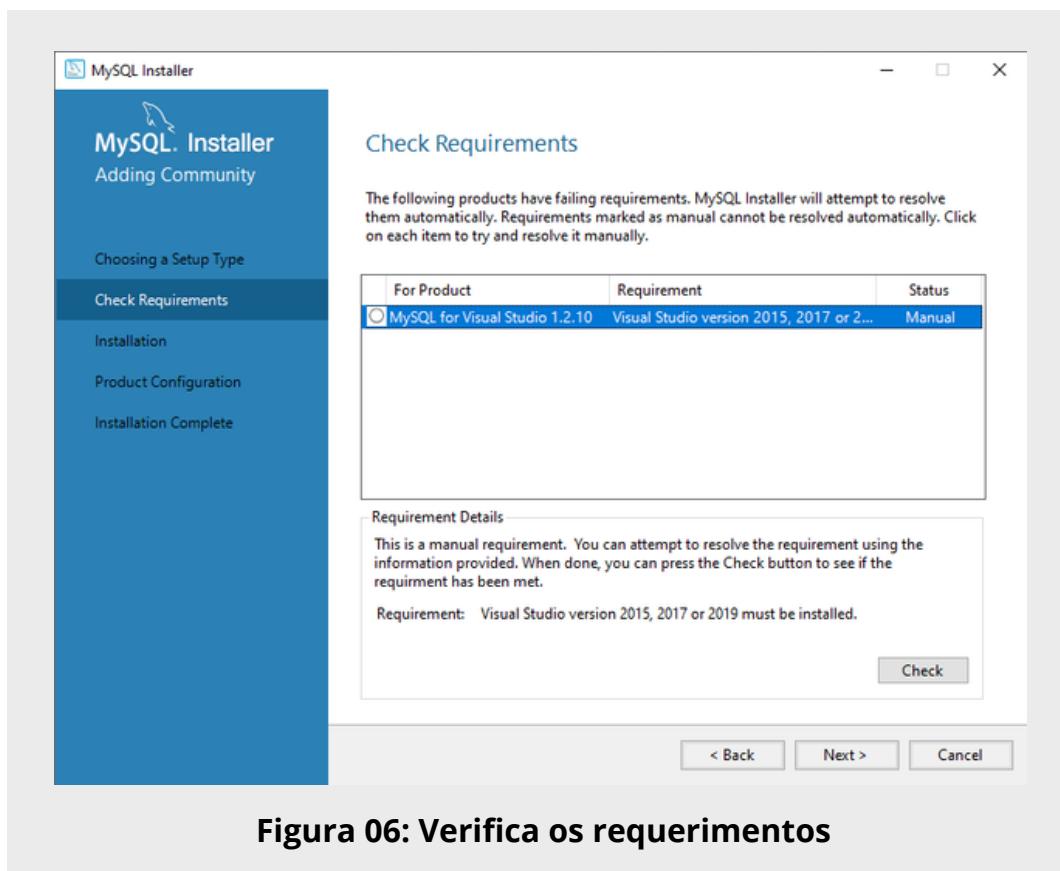
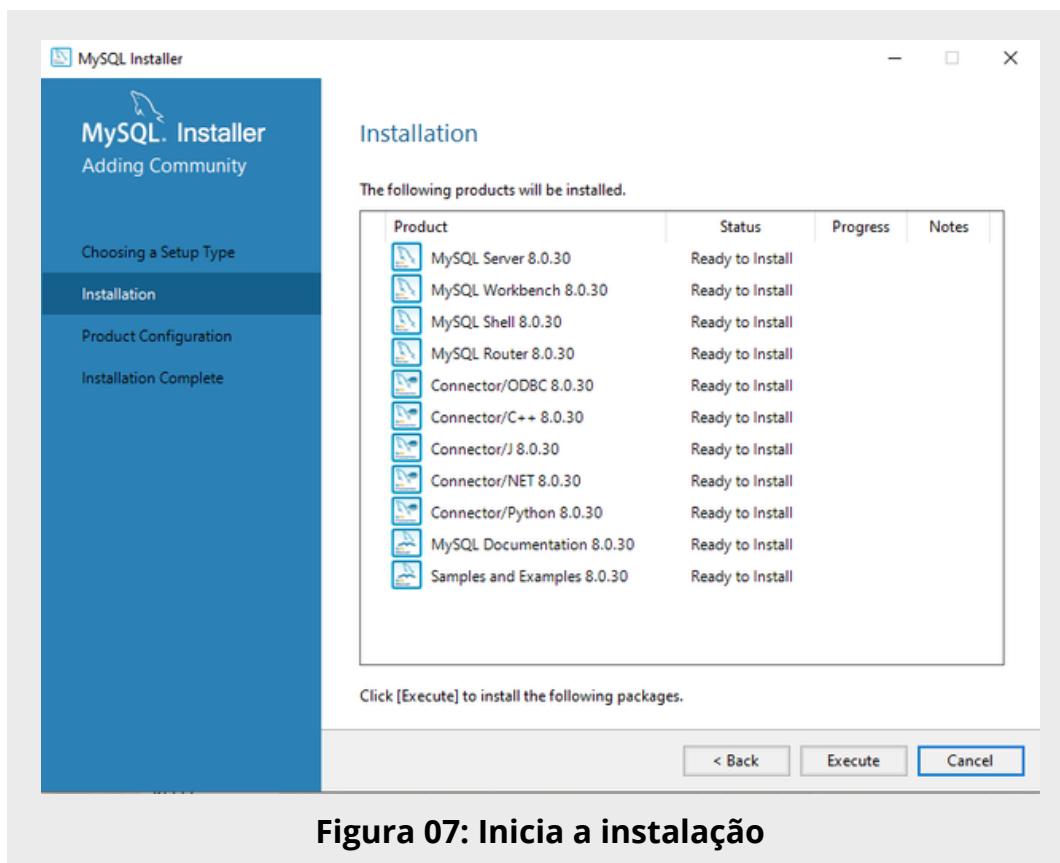
Full
 Installs all included MySQL products and features.

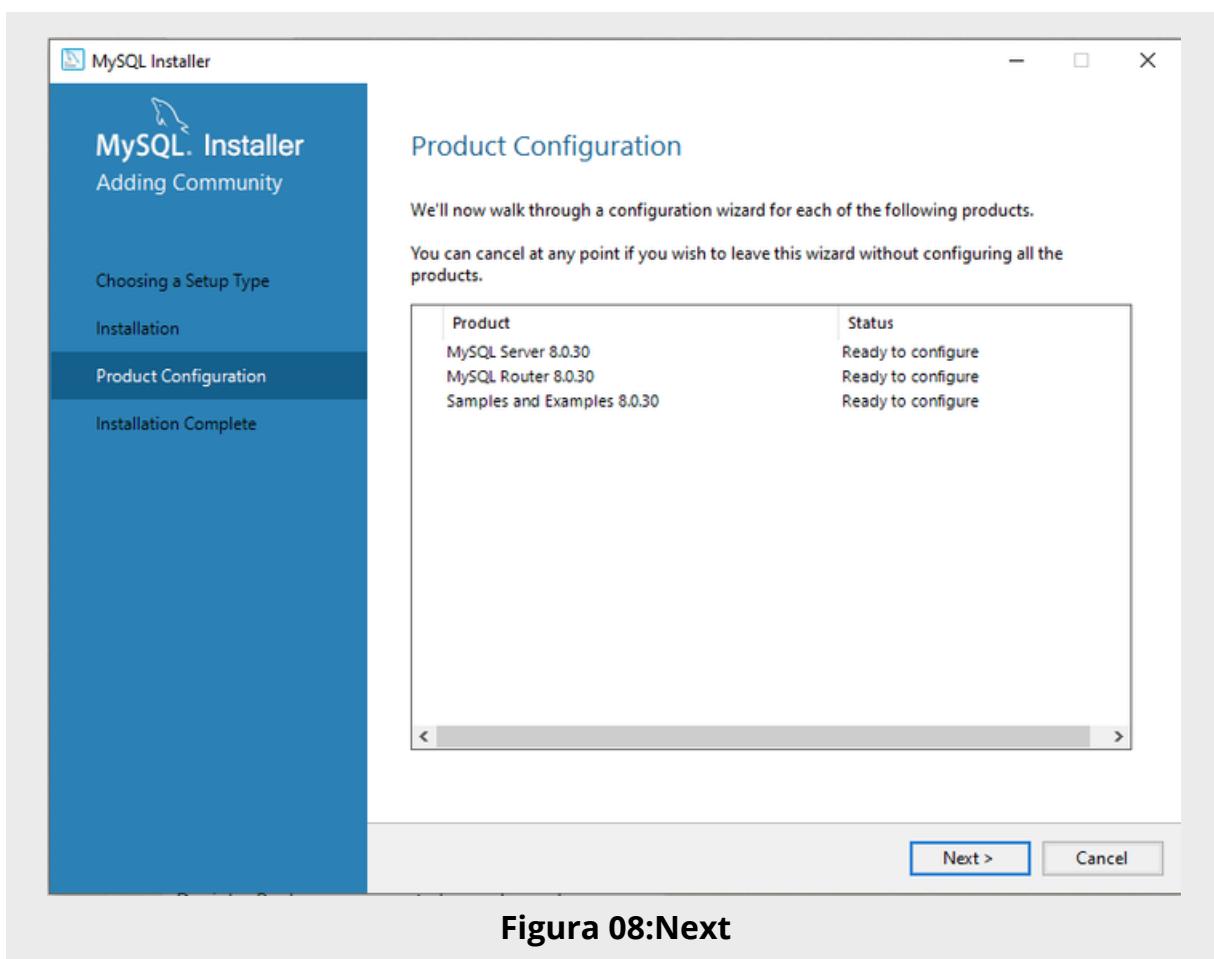
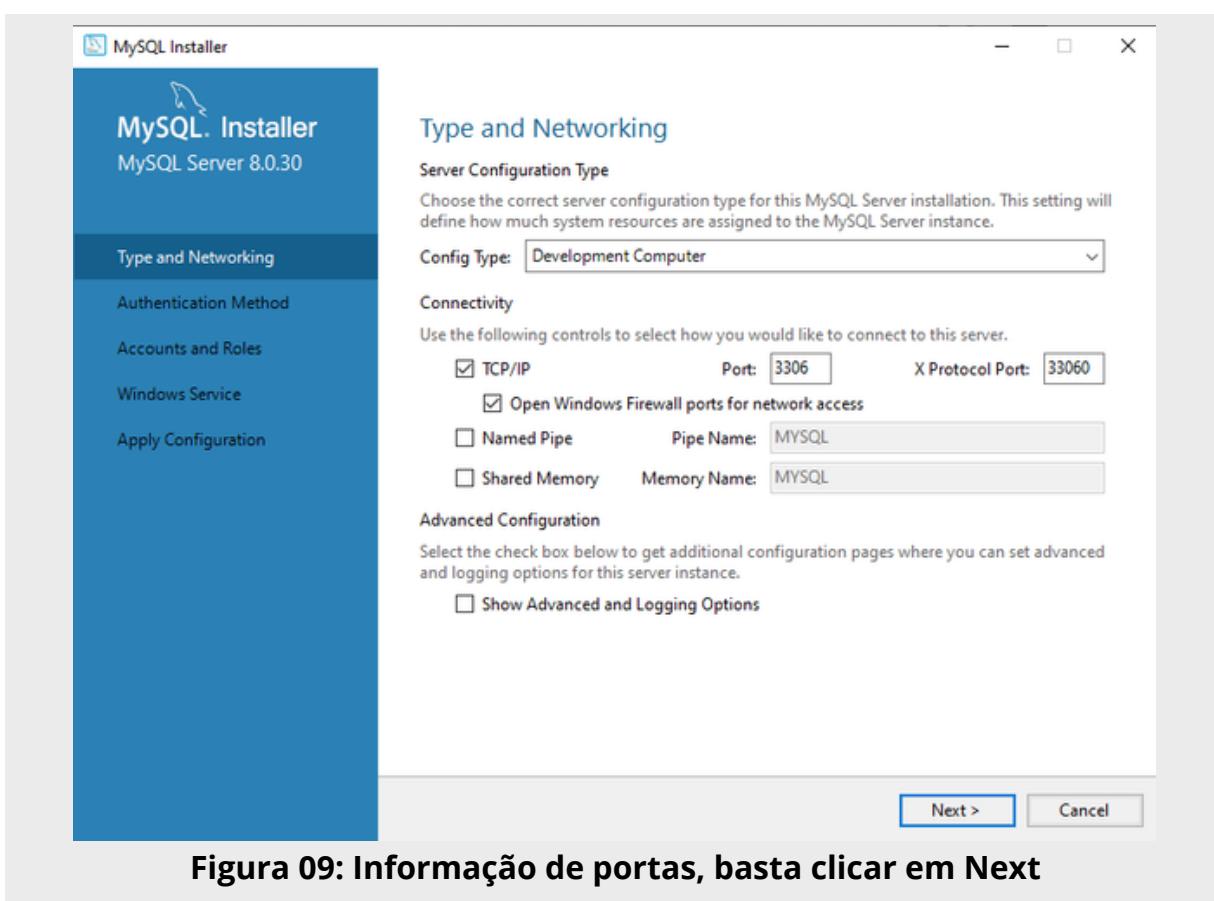
Custom
 Manually select the products that should be installed on the system.

Setup Type Description
 Installs the MySQL Server and the tools required for MySQL application development. This is useful if you intend to develop applications for an existing server.
 This Setup Type includes:
 * MySQL Server
 The most popular Open Source SQL database management system.
 * MySQL Shell
 The new MySQL client application to manage MySQL Servers and InnoDB cluster instances.
 * MySQL Router

Next > Cancel

Figura 05: Após descompactar execute o arquivo, escolha a opção acima.


Figura 06: Verifica os requerimentos

Figura 07: Inicia a instalação


Figura 08:Next

Figura 09: Informação de portas, basta clicar em Next

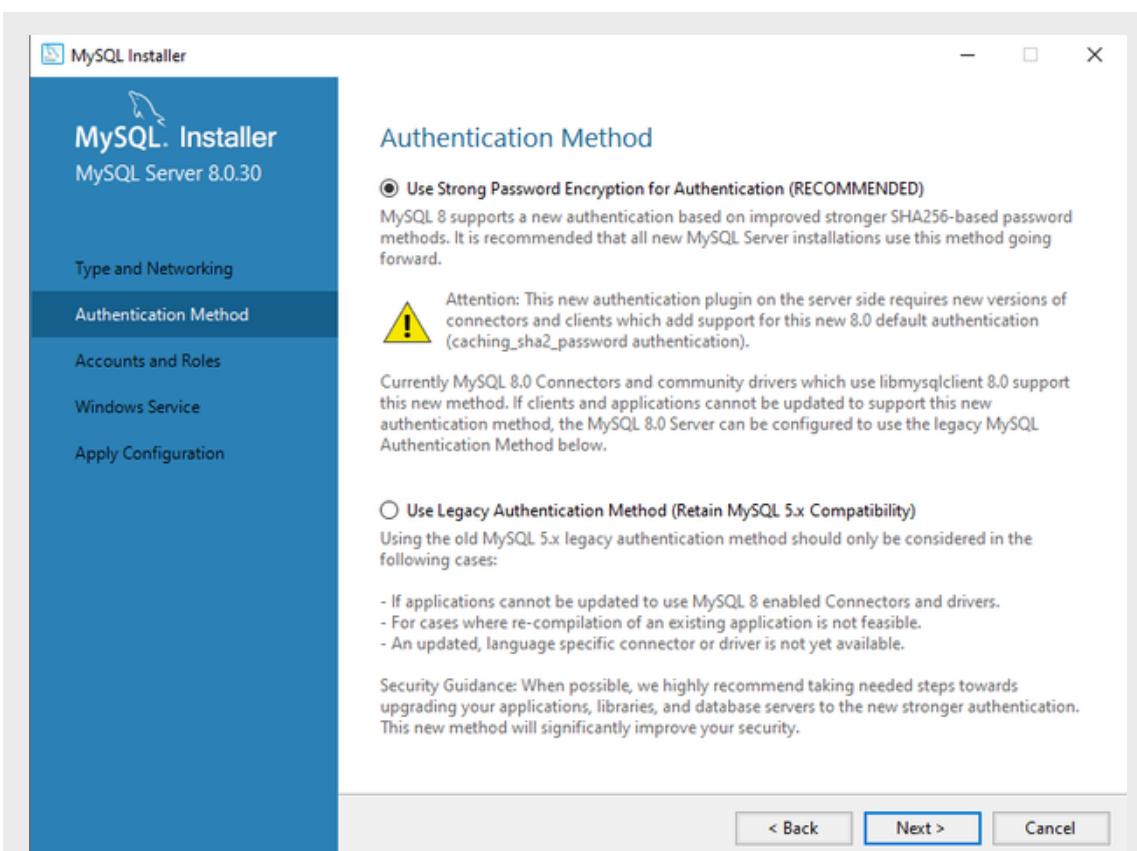


Figura 10: Next

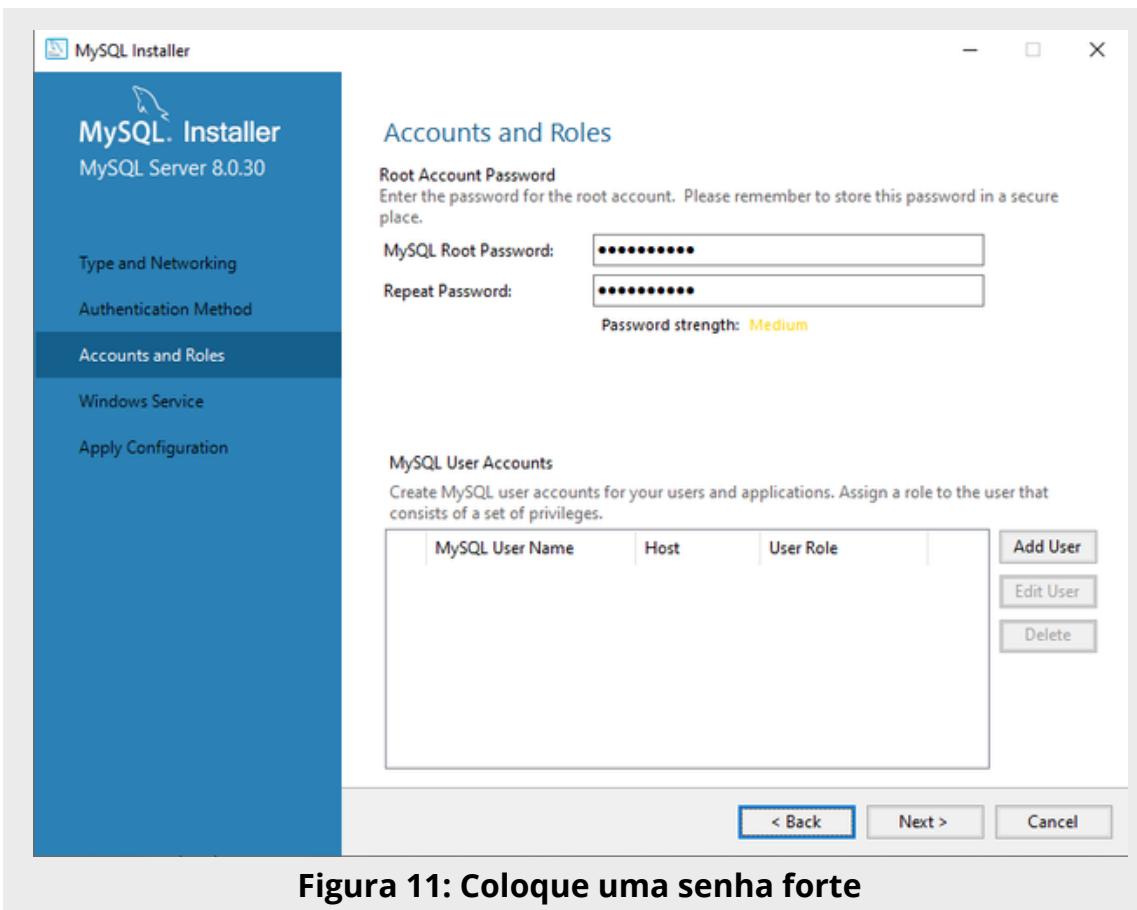


Figura 11: Coloque uma senha forte

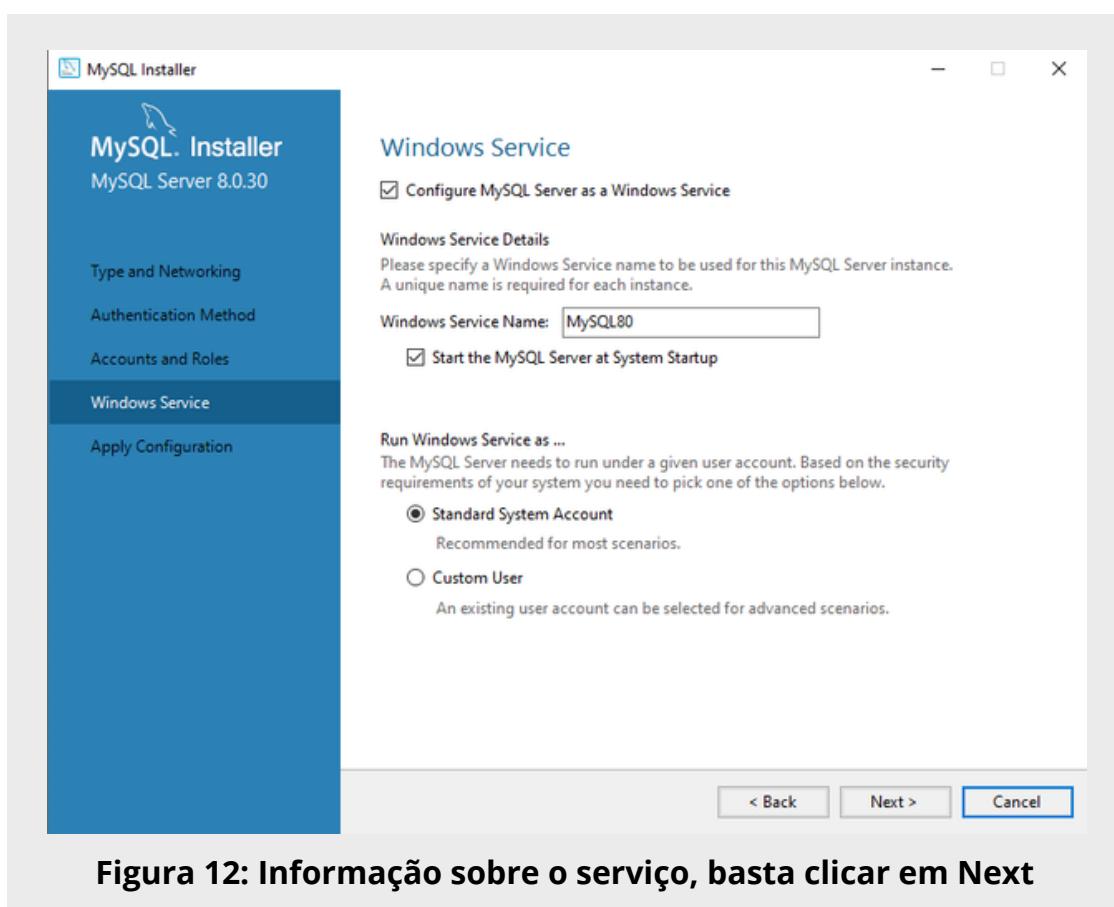


Figura 12: Informação sobre o serviço, basta clicar em Next

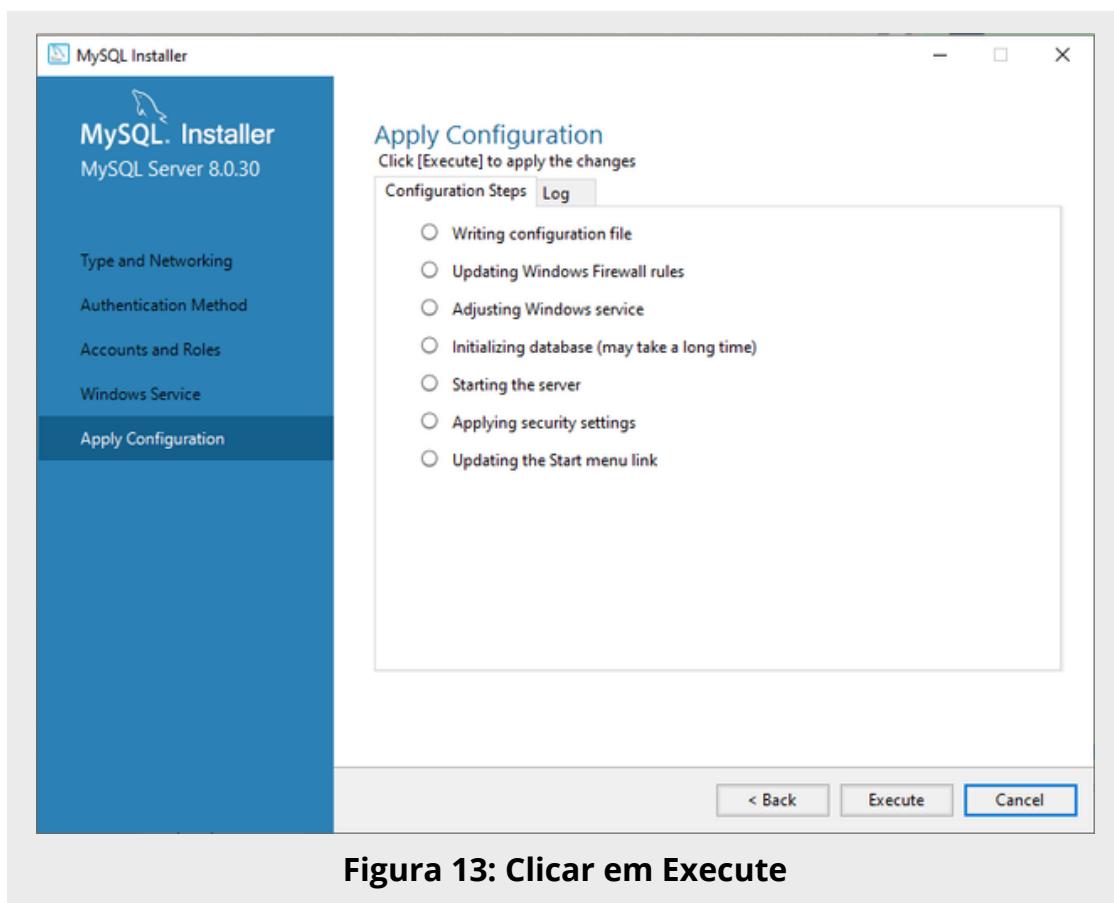


Figura 13: Clicar em Execute

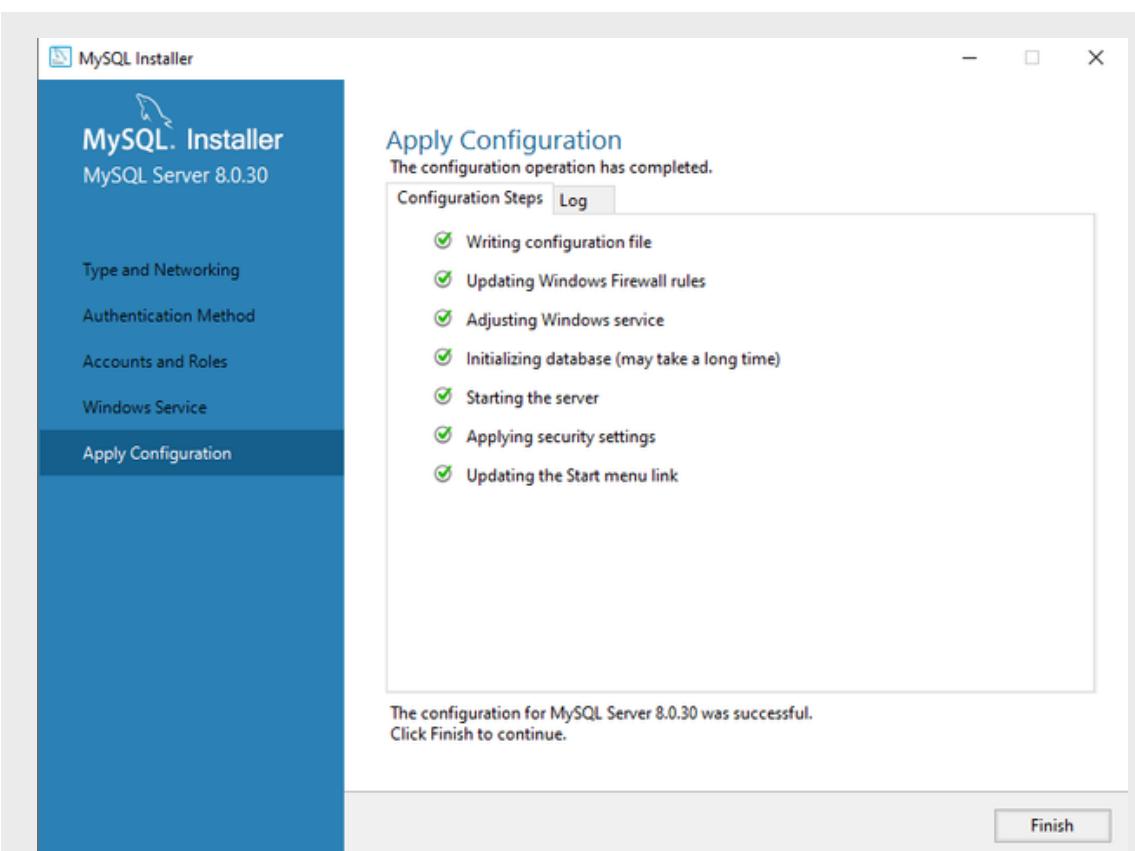


Figura 14: Finish

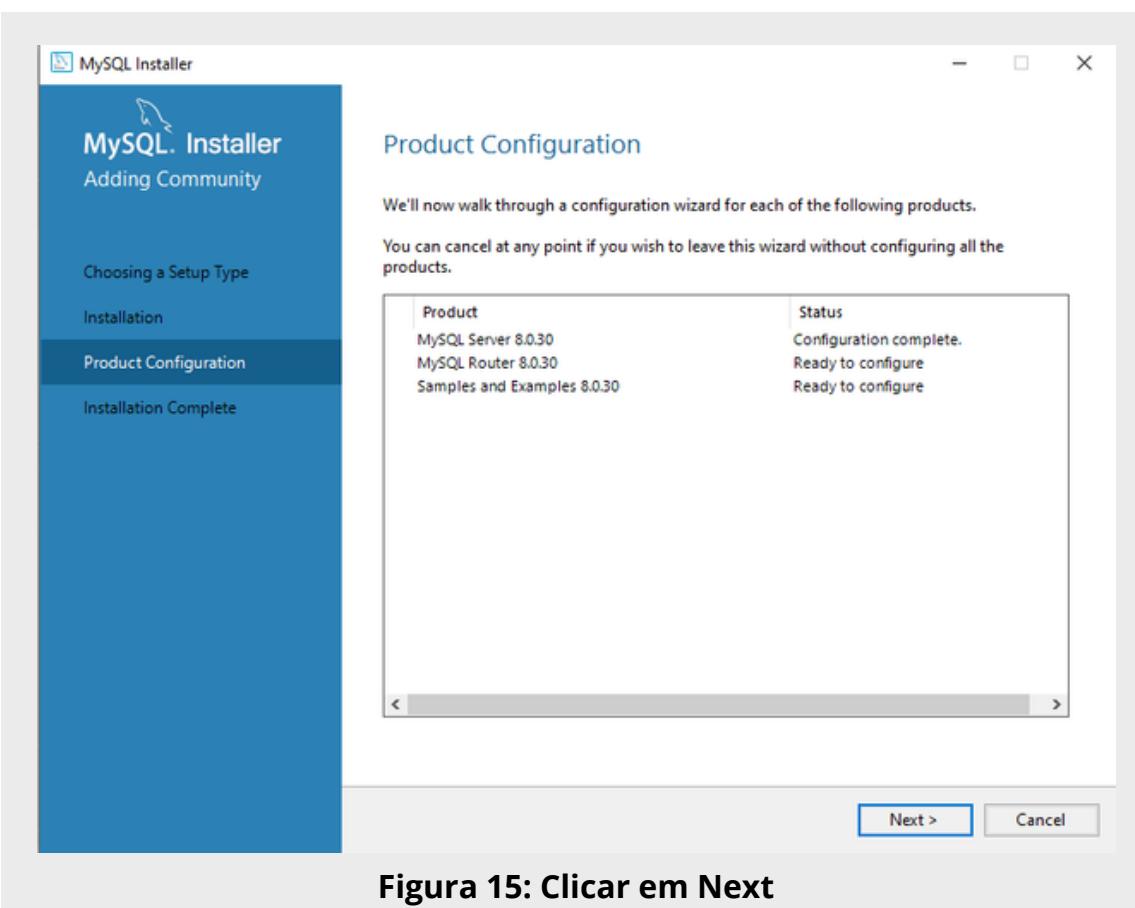
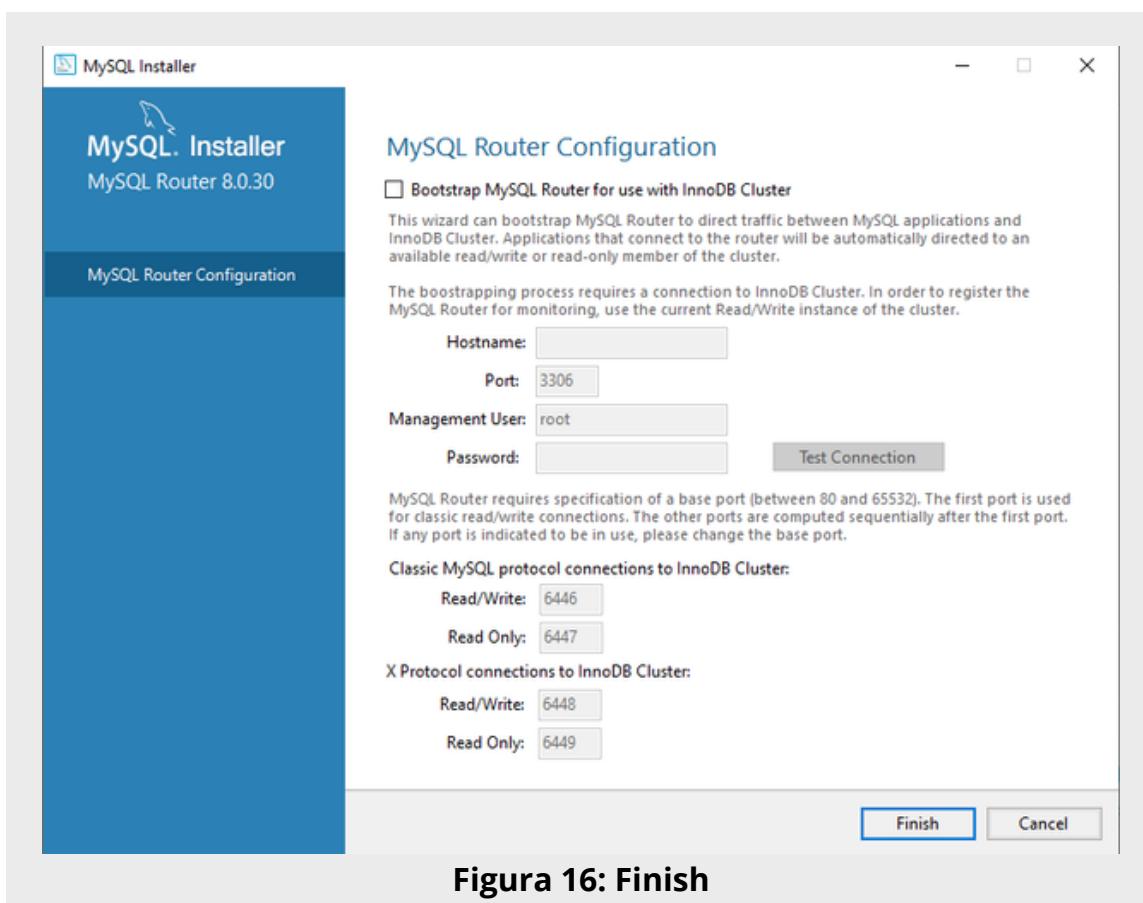
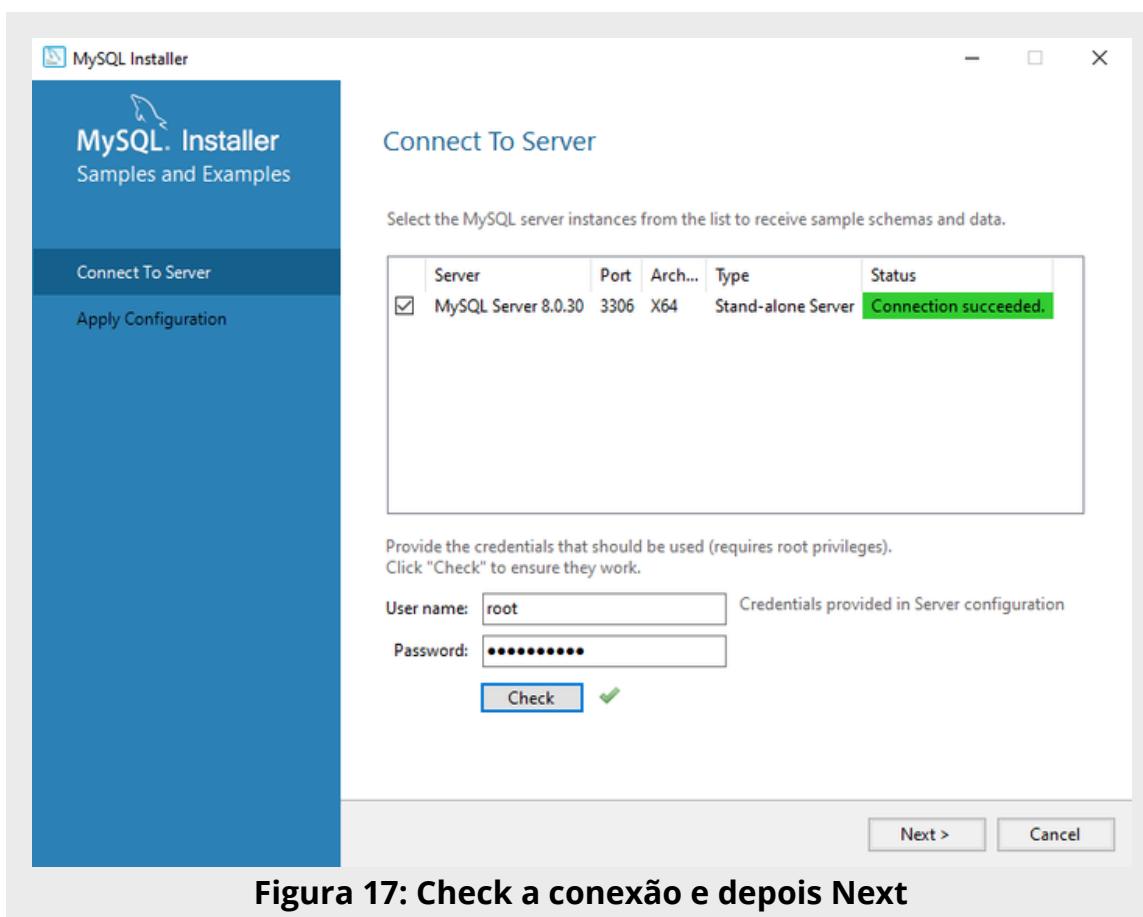


Figura 15: Clicar em Next


Figura 16: Finish

Figura 17: Check a conexão e depois Next

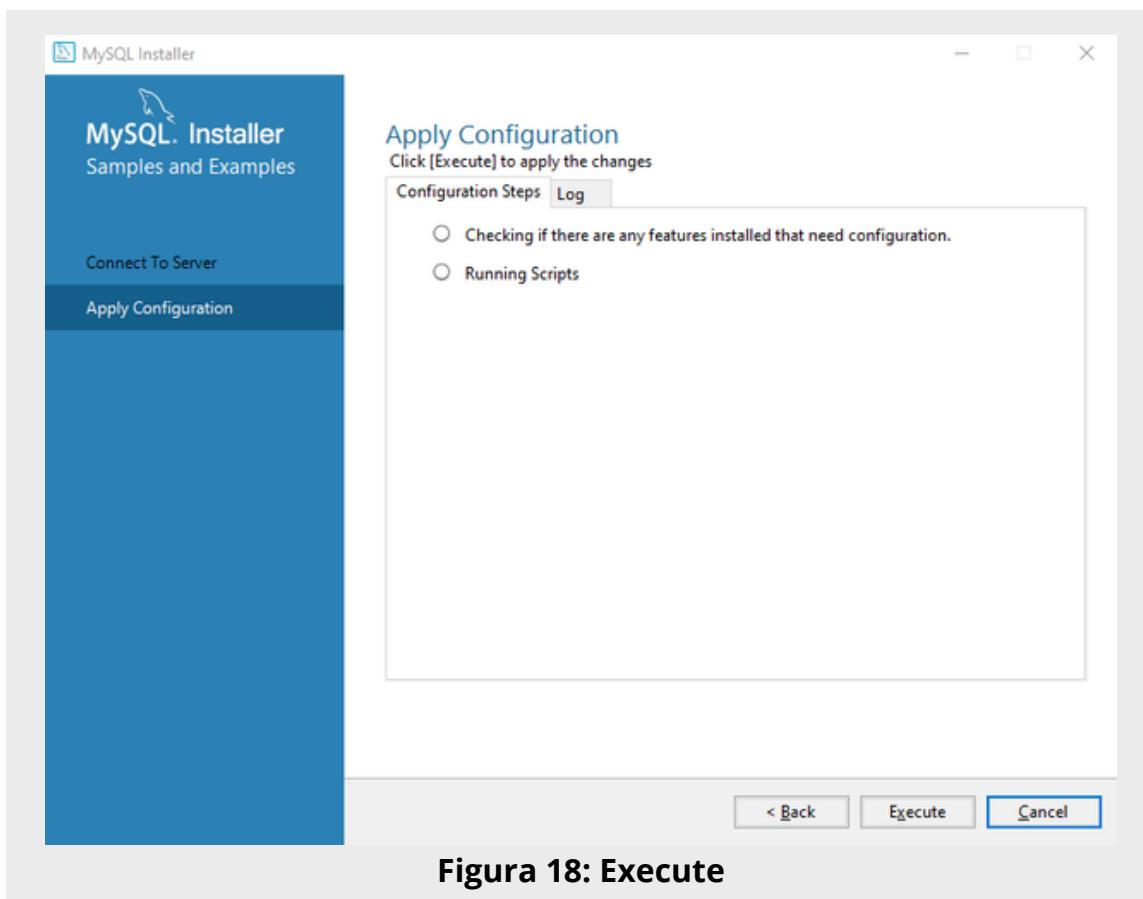


Figura 18: Execute

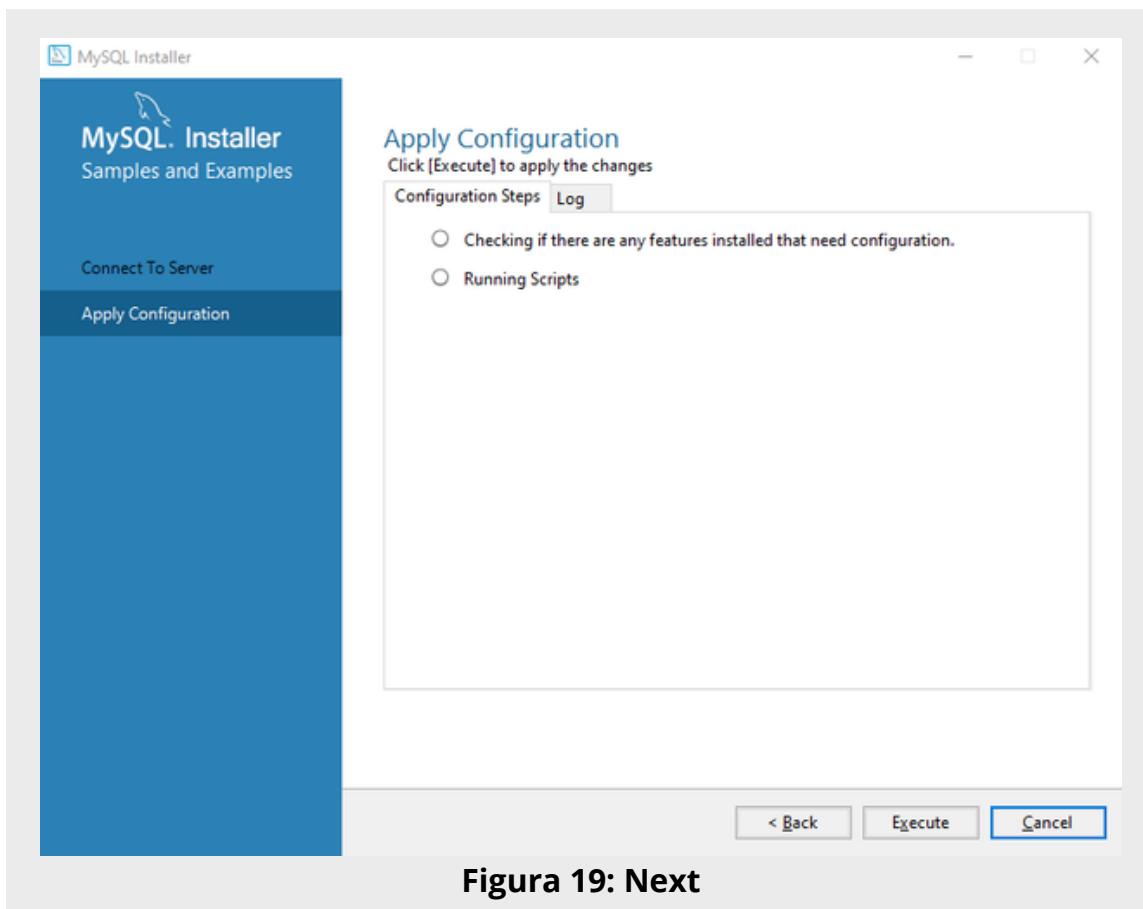


Figura 19: Next

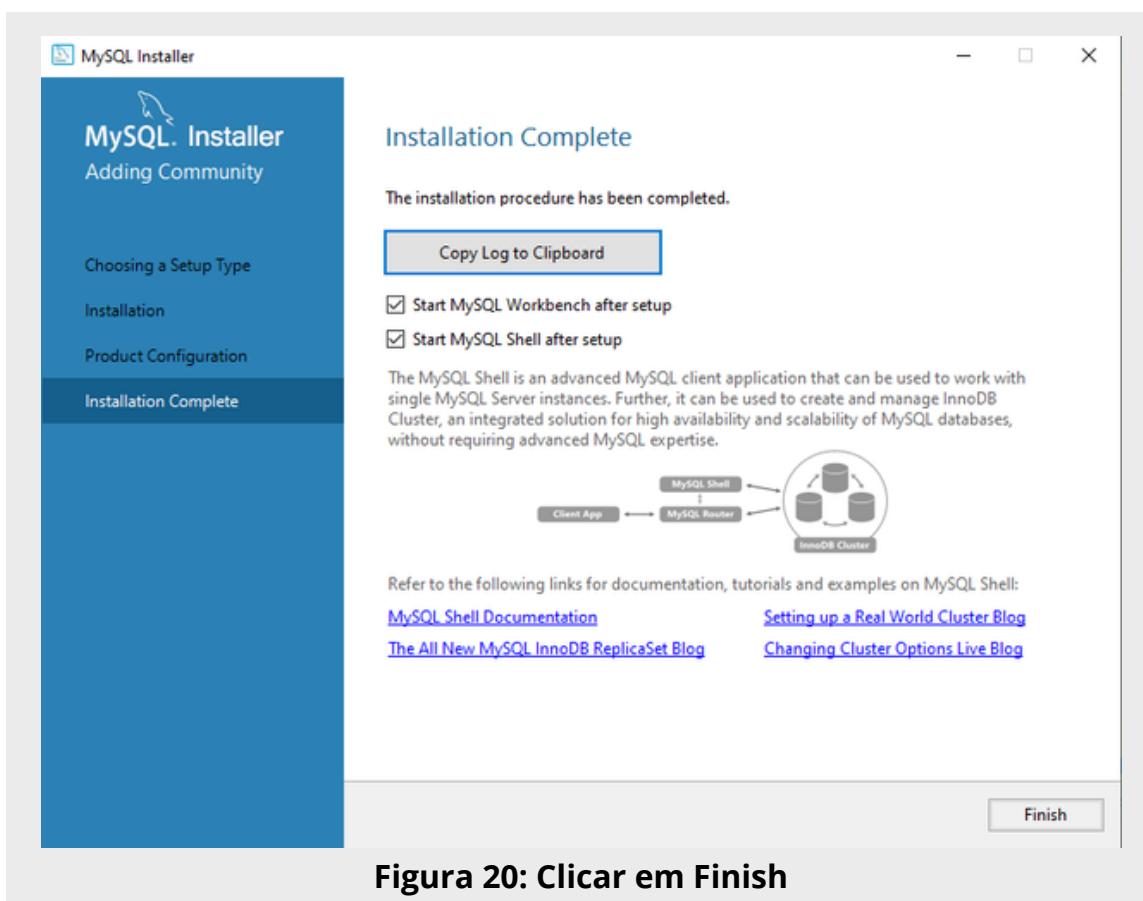


Figura 20: Clicar em Finish

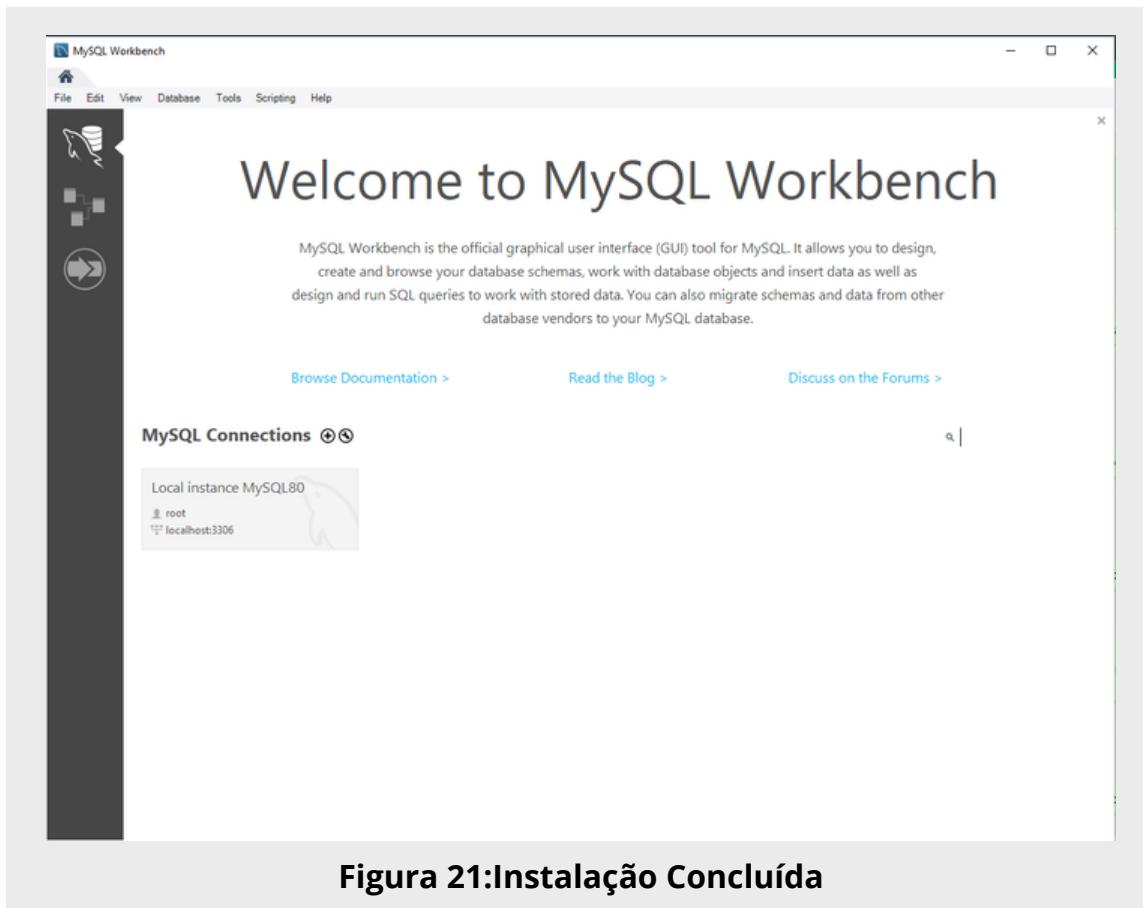


Figura 21:Instalação Concluída

5. Introdução ao MySQL Workbench

Como demonstrado na figura 23, temos a tela inicial do MySQL Workbench, ao clicar em Local Instance será solicitado a senha que foi utilizada na instalação do MySQL. A figura 24 exibe a tela inicial do Workbench.

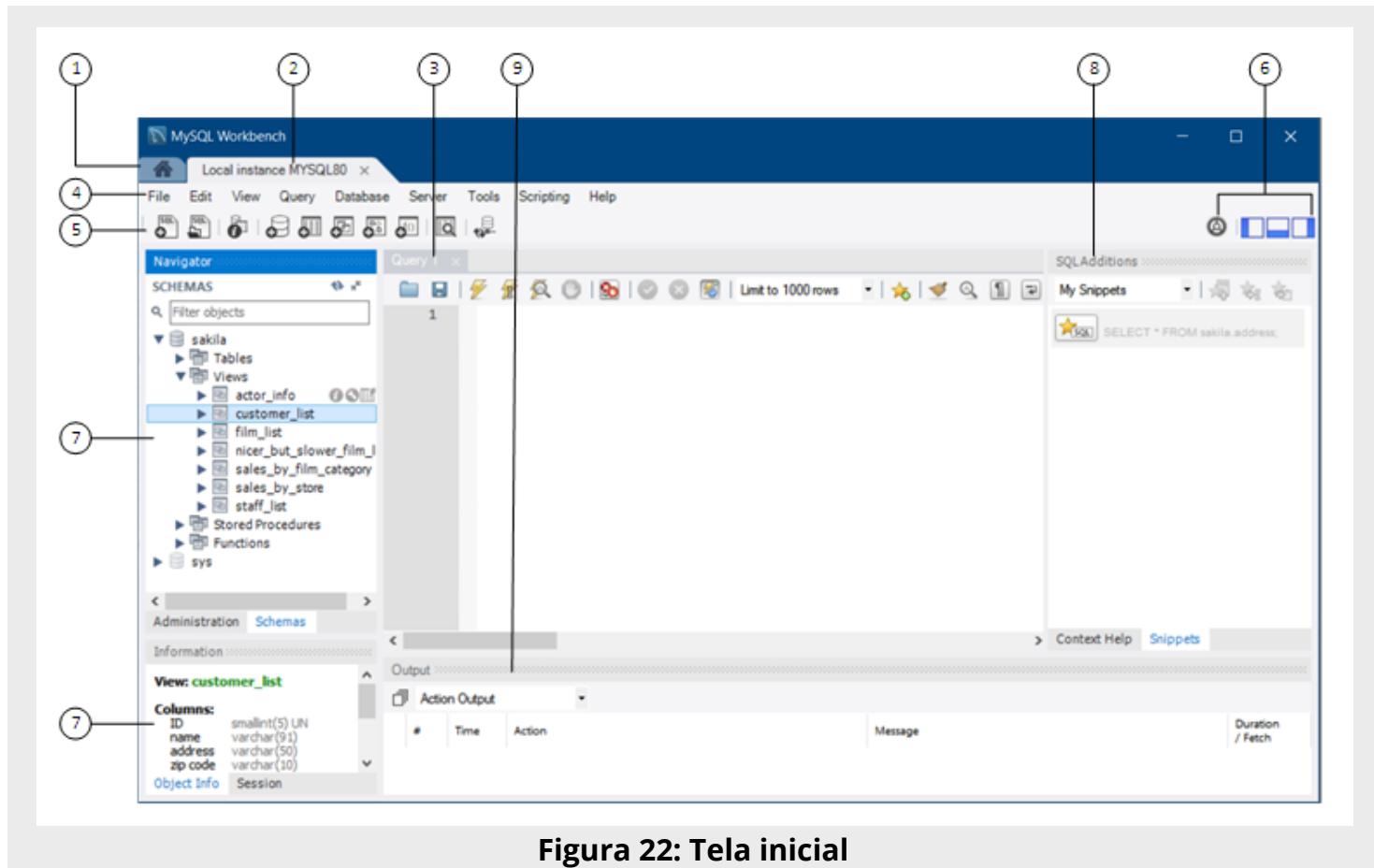


Figura 22: Tela inicial

Descrição dos elementos do Visual SQL Editor

01 - Guia da tela inicial. A guia da tela inicial fornece acesso rápido a conexões, modelos e o assistente de migração do MySQL. Ao contrário das outras guias principais, a guia da tela inicial não fecha.

02 - Aba de conexão. Cada conexão feita ao servidor MySQL é representada por uma guia de conexão separada. Um servidor pode estar ativo ou inativo quando a guia de conexão para ele é aberta.

03 - Guia de consulta SQL. A guia de consulta SQL é uma guia secundária que é aberta por padrão quando você faz uma conexão com um servidor MySQL. Cada guia de consulta é identificado exclusivamente por um número crescente: consulta 1, consulta 2 e assim por diante. Para fechar uma guia aberta, clique no x na guia. Todas as guias de consulta SQL fornecem uma área para editar consultas. Você pode abrir outros editores especializados em abas nesta mesma área central. Por exemplo, você pode editar esquemas, tabelas, colunas e assim por diante. As guias de administração também são abertas nesta área.

04 - Barra de menu principal. A barra de menus possui os seguintes menus: Arquivo, Editar, Exibir, Consulta, Banco de dados, Servidor, Ferramentas, Script e Ajuda. As ações disponíveis para você dependem de qual guia é selecionado quando você clica em um menu.

05 - Barra de ferramentas principal. As ações rápidas nesta barra de ferramentas são (ordenadas da esquerda para a direita):

- Crie uma nova guia SQL para executar consultas
- Abra um arquivo de script SQL em uma nova guia de consulta
- Abra o Inspetor para o objeto selecionado
- Crie um novo esquema no servidor conectado
- Crie uma nova tabela no esquema ativo no servidor conectado
- Crie uma nova visualização no esquema ativo no servidor conectado
- Crie um novo procedimento armazenado no esquema ativo no servidor conectado
- Crie uma nova função no esquema ativo no servidor conectado
- Pesquisar dados de tabela para texto em objetos selecionados na árvore de esquema da barra lateral
- Reconectar ao DBMS

06 - Ações de atalho. Fornece os seguintes atalhos (ordenados da esquerda para a direita):

- Mostrar caixa de diálogo de preferências
- Ocultar ou mostrar o painel da barra lateral
- Ocultar ou mostrar o painel da área de saída
- Ocultar ou mostrar o painel secundário da barra lateral

07 - Painel da barra lateral. A barra lateral possui dois rótulos principais: Navegador e Informações. Os rótulos são omitidos em alguns hosts. O Navegador possui duas sub-guias: Administração (anteriormente denominada Gerenciamento) e Esquemas. Você pode mesclar (ou separar) o conteúdo das duas guias em uma única lista clicando em mesclar (). A área Informações fornece as subguias Informações do Objeto e Sessão, que incluem informações somente leitura sobre um objeto selecionado e sobre a conexão ativa.

08 - Painel da barra lateral secundária (Adições SQL). A área SQL Additions fornece as seguintes sub-guias:

- Ajuda de Contexto
- Trechos

09 - Painel da área de saída. O painel de saída pode exibir um resumo das consultas executadas nos seguintes formatos: Saída de ação, Saída de texto ou Saída de histórico.

5.1. Criando a Estrutura Física

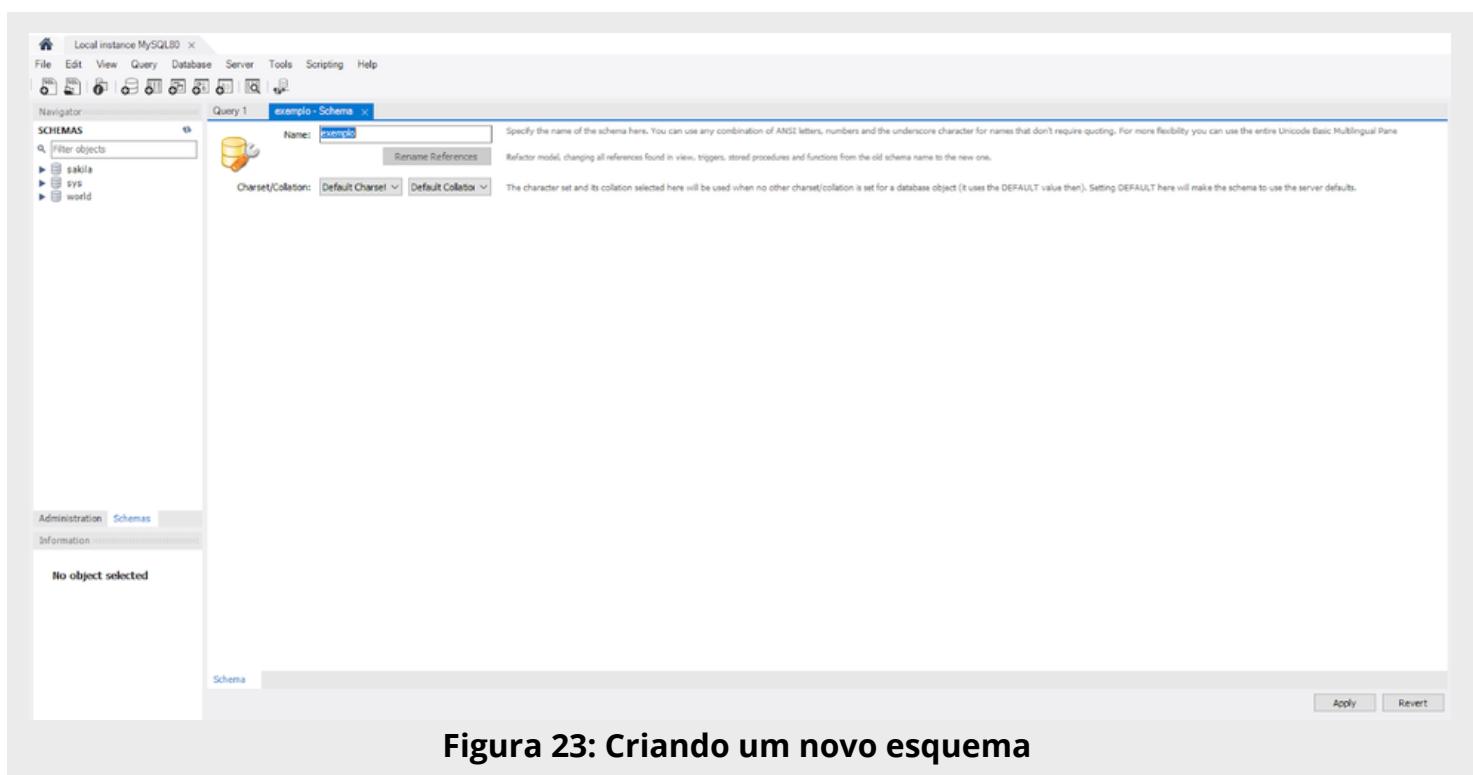


Figura 23: Criando um novo esquema

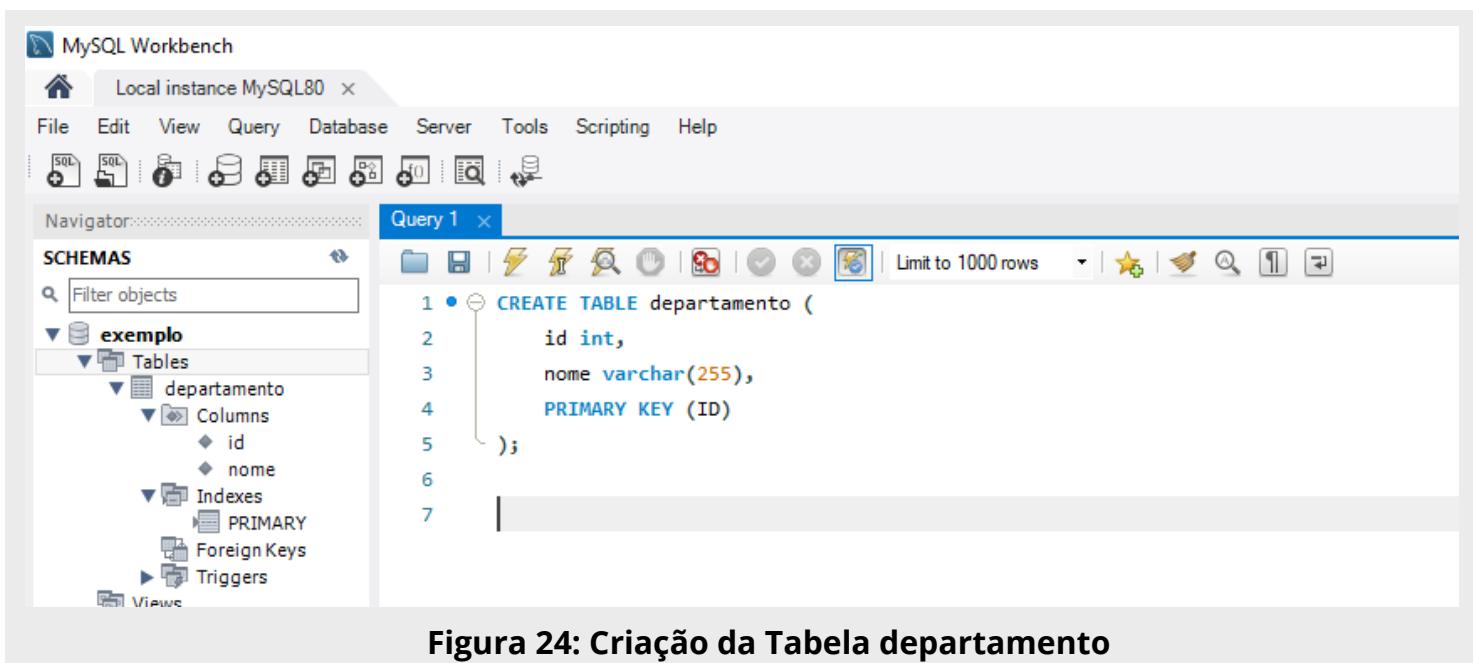


Figura 24: Criação da Tabela departamento

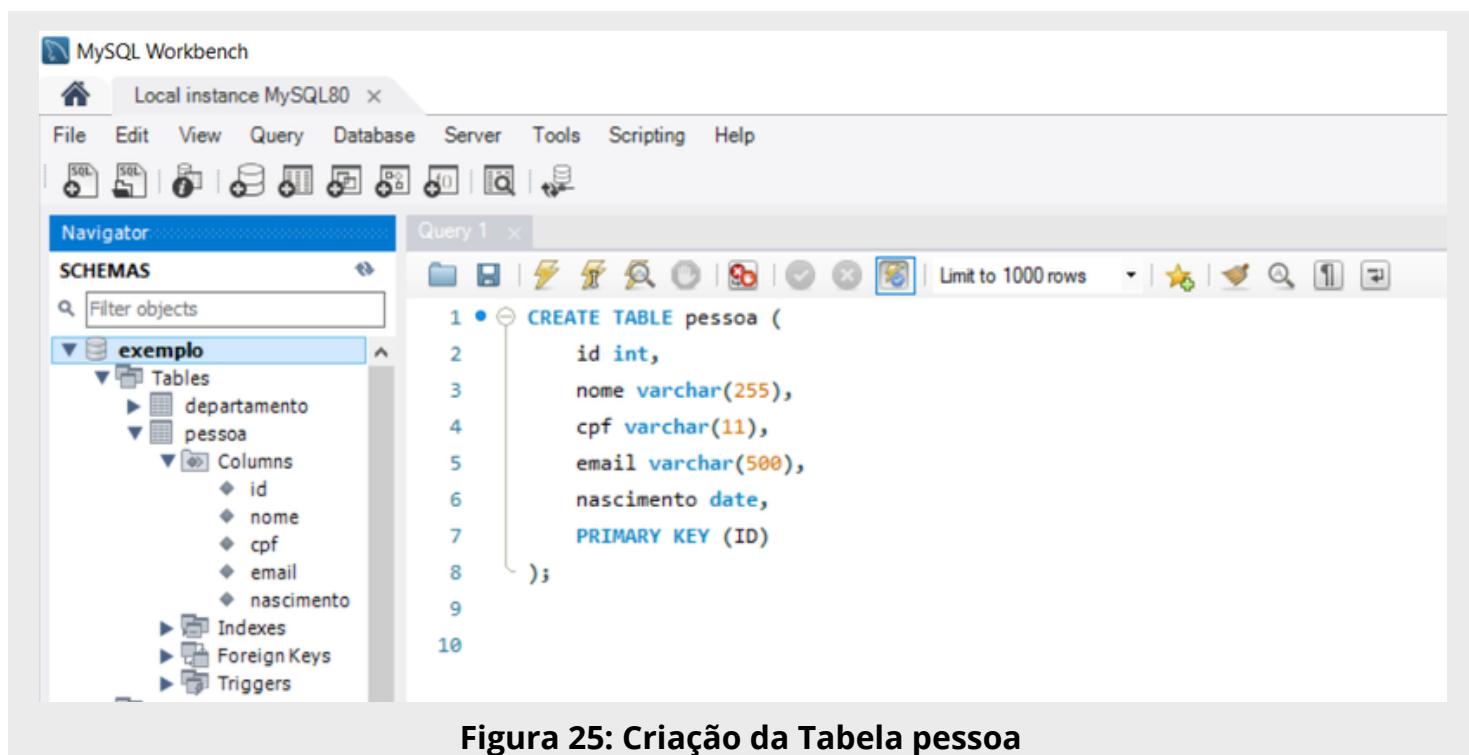


Figura 25: Criação da Tabela pessoa

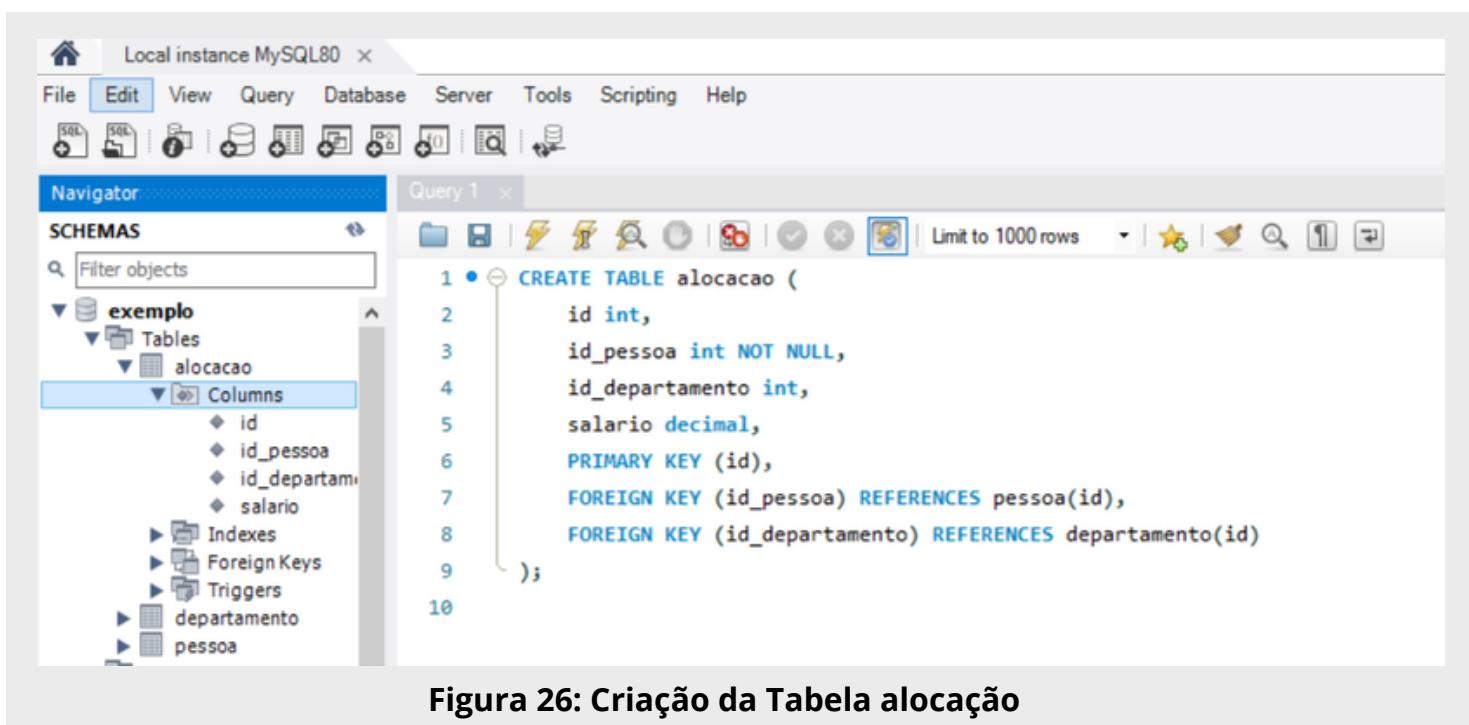
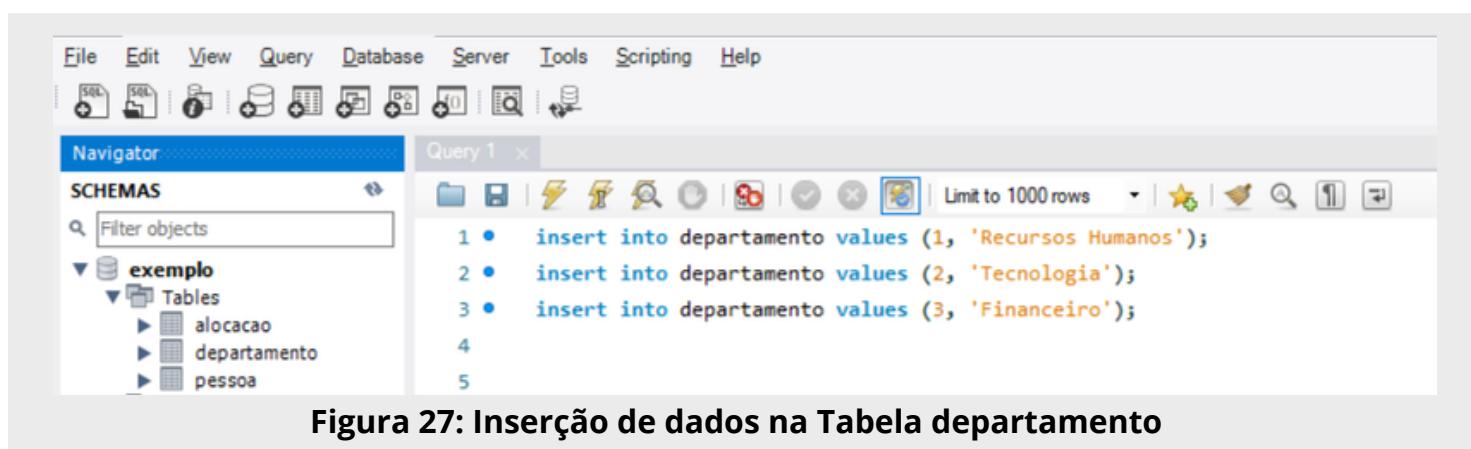


Figura 26: Criação da Tabela alocação

5.2. Inserindo Dados

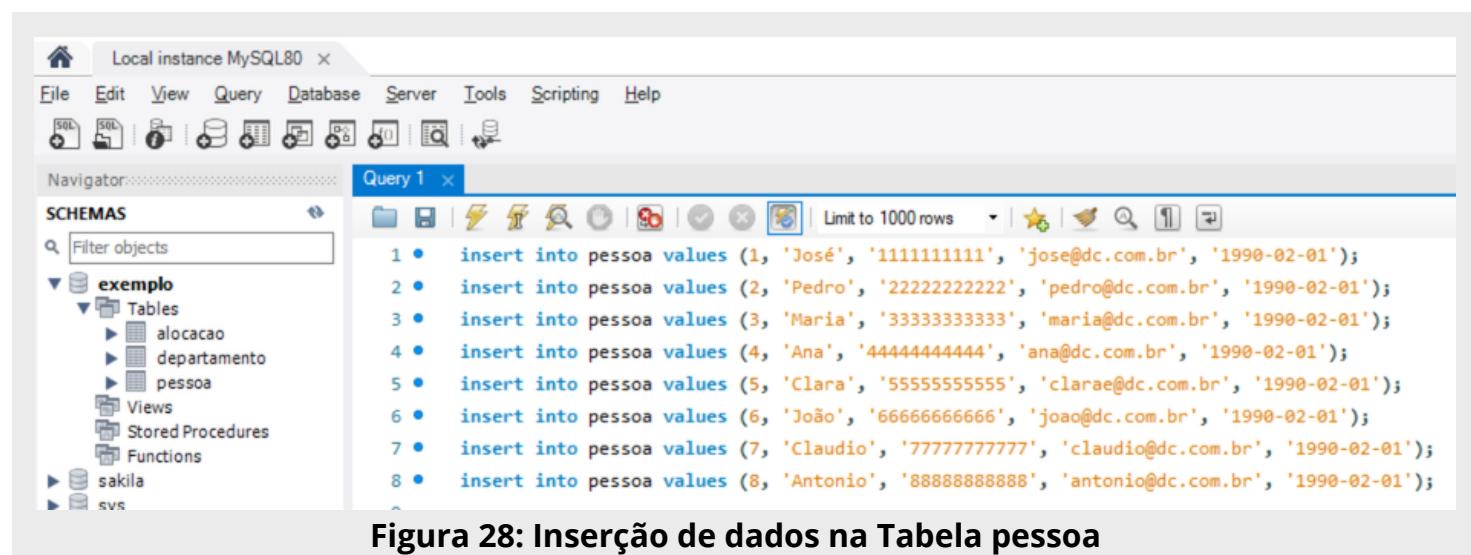


The screenshot shows the SQL Server Management Studio interface. The left pane displays the Navigator with the SCHEMAS section expanded, showing the 'exemplo' schema which contains three tables: 'alocacao', 'departamento', and 'pessoa'. The right pane shows a query window titled 'Query 1' containing the following SQL code:

```

1 • insert into departamento values (1, 'Recursos Humanos');
2 • insert into departamento values (2, 'Tecnologia');
3 • insert into departamento values (3, 'Financeiro');
4
5
    
```

Figura 27: Inserção de dados na Tabela departamento

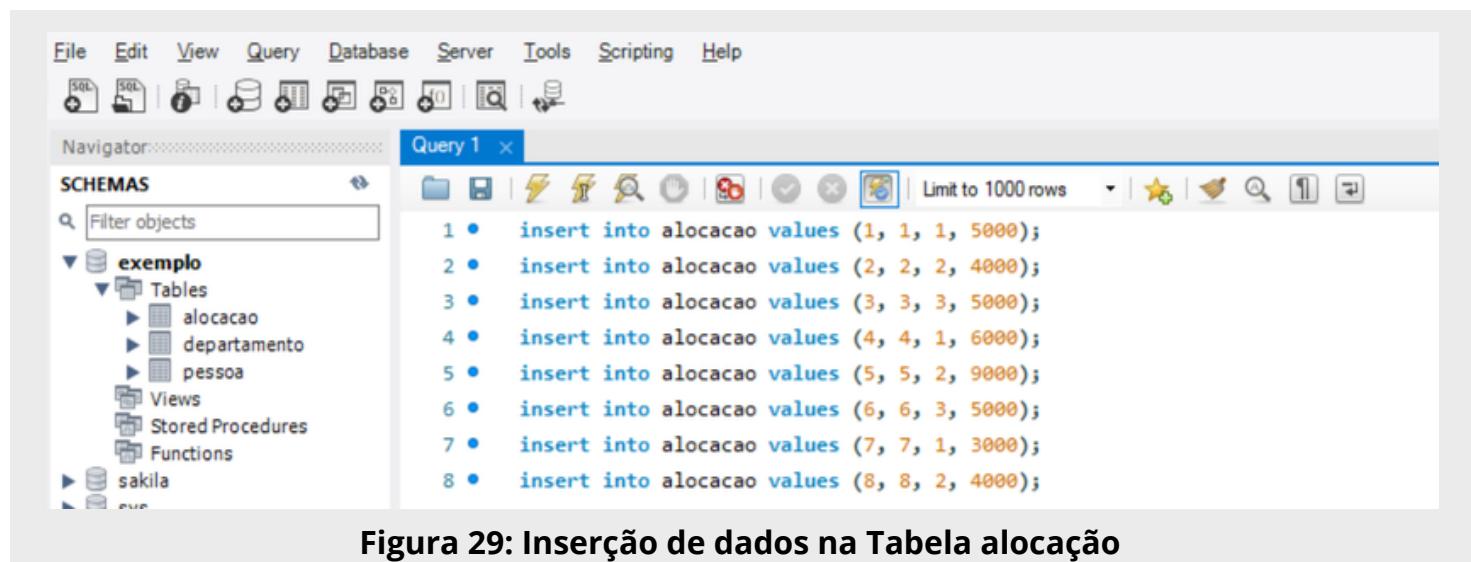


The screenshot shows the MySQL Workbench interface. The left pane displays the Navigator with the SCHEMAS section expanded, showing the 'exemplo' schema which contains four tables: 'alocacao', 'departamento', 'pessoa', and 'sakila'. It also shows 'Views', 'Stored Procedures', and 'Functions'. The right pane shows a query window titled 'Query 1' containing the following SQL code:

```

1 • insert into pessoa values (1, 'José', '1111111111', 'jose@dc.com.br', '1990-02-01');
2 • insert into pessoa values (2, 'Pedro', '2222222222', 'pedro@dc.com.br', '1990-02-01');
3 • insert into pessoa values (3, 'Maria', '3333333333', 'maria@dc.com.br', '1990-02-01');
4 • insert into pessoa values (4, 'Ana', '4444444444', 'ana@dc.com.br', '1990-02-01');
5 • insert into pessoa values (5, 'Clara', '5555555555', 'clarae@dc.com.br', '1990-02-01');
6 • insert into pessoa values (6, 'João', '6666666666', 'joao@dc.com.br', '1990-02-01');
7 • insert into pessoa values (7, 'Claudio', '7777777777', 'claudio@dc.com.br', '1990-02-01');
8 • insert into pessoa values (8, 'Antonio', '8888888888', 'antonio@dc.com.br', '1990-02-01');
    
```

Figura 28: Inserção de dados na Tabela pessoa



The screenshot shows the SQL Server Management Studio interface. The left pane displays the Navigator with the SCHEMAS section expanded, showing the 'exemplo' schema which contains three tables: 'alocacao', 'departamento', and 'pessoa'. It also shows 'Views', 'Stored Procedures', and 'Functions'. The right pane shows a query window titled 'Query 1' containing the following SQL code:

```

1 • insert into alocacao values (1, 1, 1, 5000);
2 • insert into alocacao values (2, 2, 2, 4000);
3 • insert into alocacao values (3, 3, 3, 5000);
4 • insert into alocacao values (4, 4, 1, 6000);
5 • insert into alocacao values (5, 5, 2, 9000);
6 • insert into alocacao values (6, 6, 3, 5000);
7 • insert into alocacao values (7, 7, 1, 3000);
8 • insert into alocacao values (8, 8, 2, 4000);
    
```

Figura 29: Inserção de dados na Tabela alocação

5.3. Consultando Dados

- **Cláusulas SQL**

FROM: Especifica a tabela de seleção de registros.

WHERE: Especifica as condições de seleção de registros.

GROUP BY: Especifica grupos específicos de registros.

HAVING: Especifica uma condição específica para o retorno de registros.

ORDER BY: Ordena as colunas de determinada tabela.

DISTINCT: A seleção de dados sem repetição.

- **Operadores Lógicos**

AND: "e" lógico

OR: "ou" lógico

NOT: negação

- **Operadores Relacionais**

<: menor

>: maior

<=: menor ou igual

>=: maior ou igual

=: igual

<>: diferente

BETWEEN: intervalo de valores

LIKE: semelhante a “=”. Porém pode ser utilizado com a extensão “%”

IN: verifica um valor dentro de uma lista

- **Funções de Agregação**

AVG: Calcula a média de valores de um determinado atributo / campo

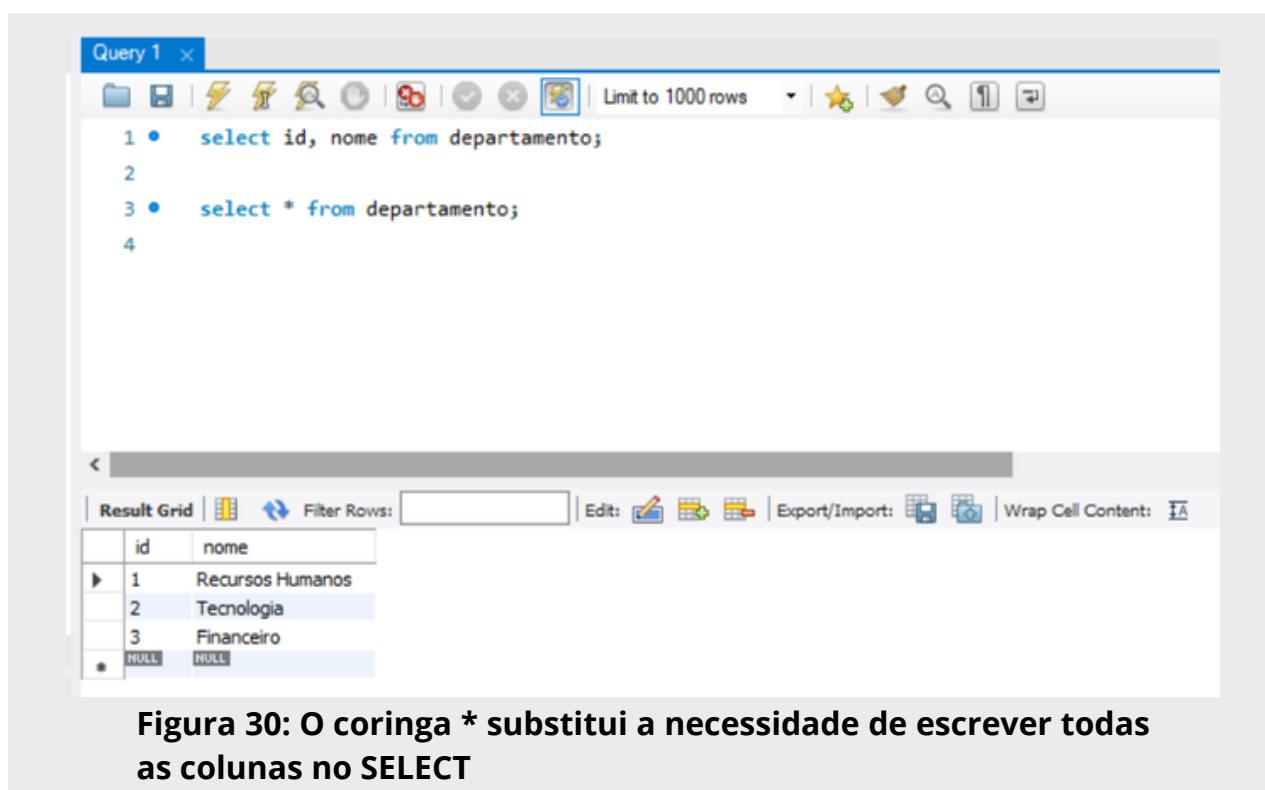
COUNT: Conta a quantidade de registros de uma seleção

SUM: Soma a quantidade de registros de uma seleção

MAX: Devolve o maior valor de um atributo / campo especificado

MIN: Devolve o menor valor de um atributo / campo especificado

SELECT é uma das instruções da linguagem SQL mais utilizadas, principalmente para a projeção de relatórios e o retorno de informações.



The screenshot shows a MySQL Workbench interface. In the top-left corner, there's a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the following SQL code:

```

1 •  select id, nome from departamento;
2
3 •  select * from departamento;
4
    
```

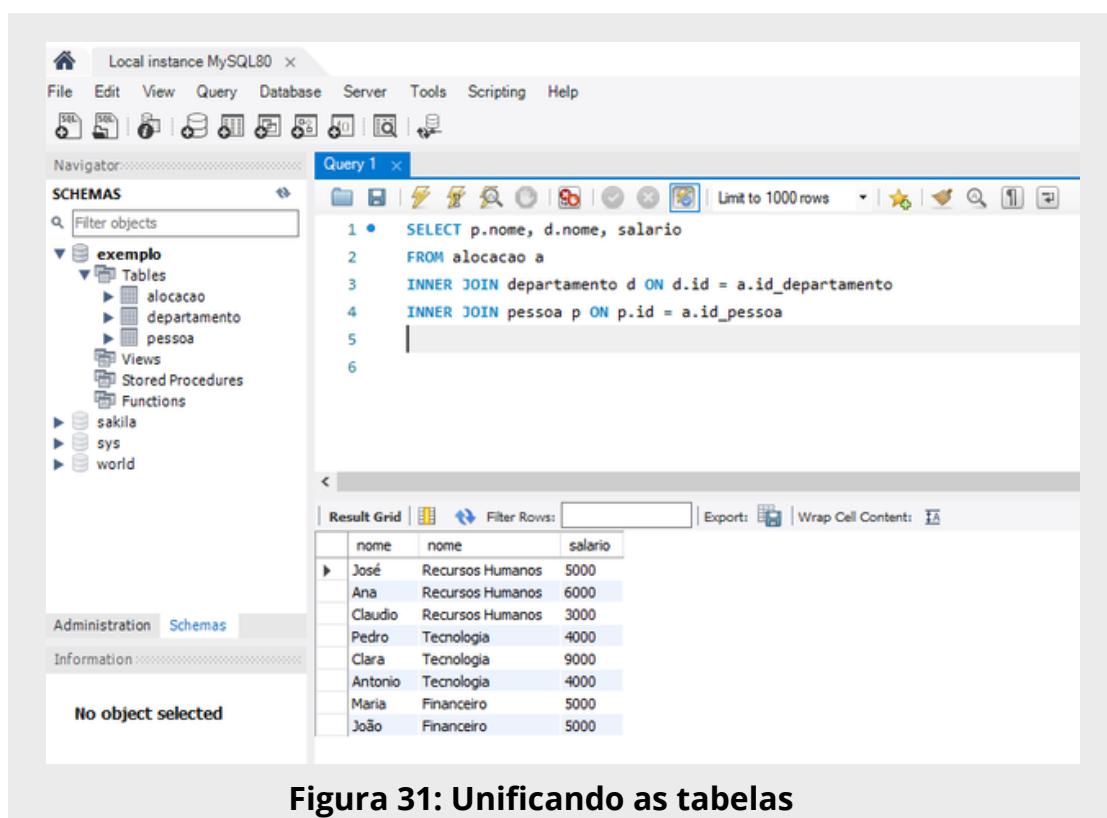
Below the query editor is a "Result Grid" window displaying the results of the second query. The grid has two columns: "id" and "nome". The data is as follows:

	id	nome
▶	1	Recursos Humanos
▶	2	Tecnologia
▶	3	Financeiro
*	NULL	NULL

Figura 30: O coringa * substitui a necessidade de escrever todas as colunas no SELECT

- **Trabalhando com JOINS**

Essa instrução é muito importante devido à sua funcionalidade. Ela faz a ligação entre as diferentes entidades pelos respectivos atributos desejados. Geralmente interliga as chaves primária e estrangeira.



The screenshot shows a MySQL Workbench interface with a more complex query. The "Navigator" pane on the left shows the "SCHEMAS" section with "exemplo" selected, which contains "Tables" like "alocacao", "departamento", and "pessoa". The "Query 1" editor contains the following SQL code:

```

1 •  SELECT p.nome, d.nome, salario
2   FROM alocacao a
3   INNER JOIN departamento d ON d.id = a.id_departamento
4   INNER JOIN pessoa p ON p.id = a.id_pessoa
5
6
    
```

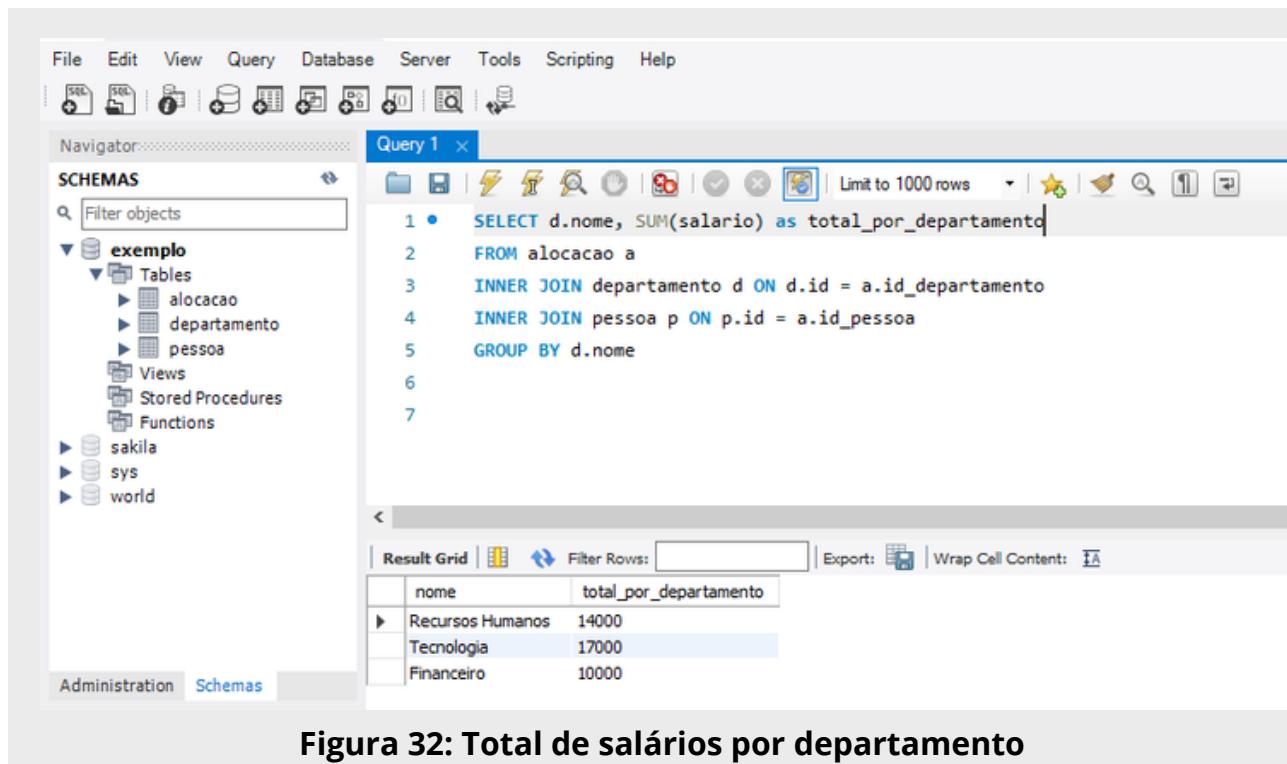
Below the query editor is a "Result Grid" window displaying the results of the query. The grid has three columns: "nome", "nome", and "salario". The data is as follows:

	nome	nome	salario
▶	José	Recursos Humanos	5000
▶	Ana	Recursos Humanos	6000
▶	Claudio	Recursos Humanos	3000
▶	Pedro	Tecnologia	4000
▶	Clara	Tecnologia	9000
▶	Antonio	Tecnologia	4000
▶	Maria	Financeiro	5000
▶	João	Financeiro	5000

Figura 31: Unificando as tabelas

- **Funções de Agregação**

SUM: Muito utilizado para a soma de valores, independentemente da forma como eles são abstraídos do banco de dados. É importante saber que se algum campo for adicionado à cláusula SELECT, obrigatoriamente deverá ser utilizada a função de agregação GROUP BY.

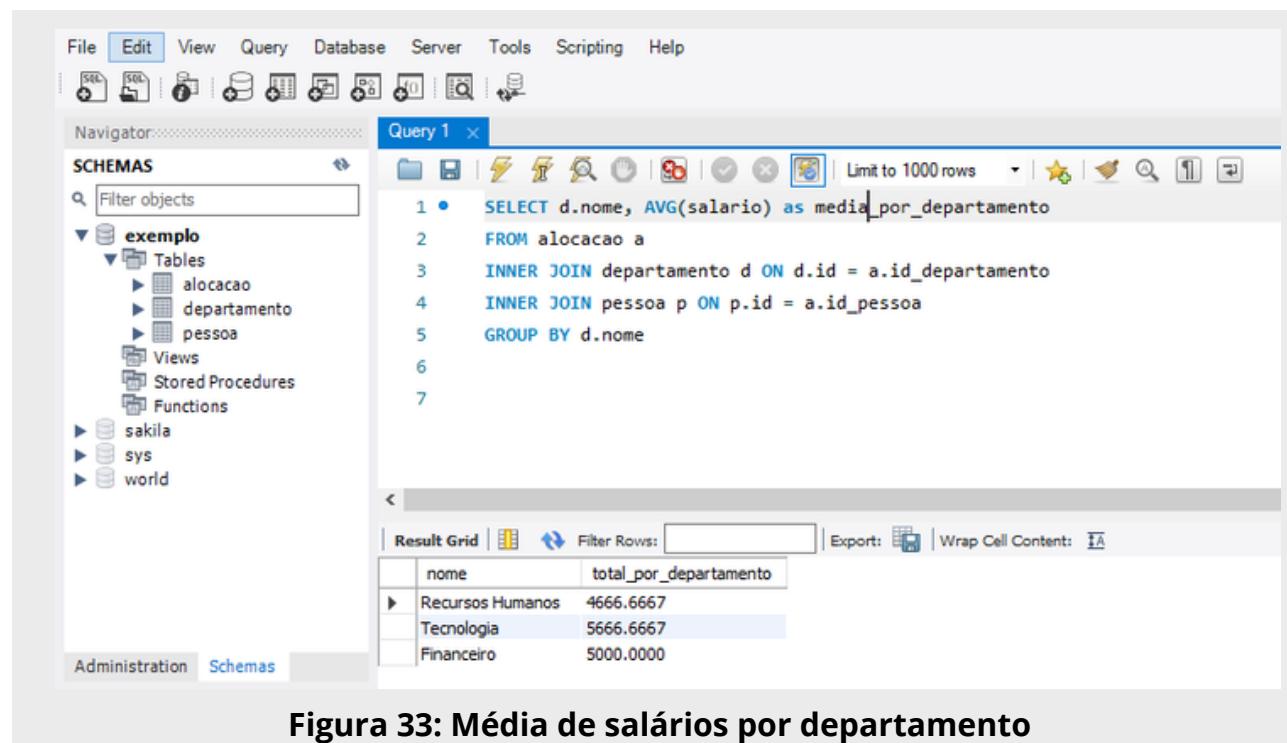


```

File Edit View Query Database Server Tools Scripting Help
Navigator: Query 1
SCHEMAS
Filter objects
exemplo
Tables
    alocacao
    departamento
    pessoa
Views
Stored Procedures
Functions
sakila
sys
world
Administration Schemas
1 • SELECT d.nome, SUM(salario) as total_por_departamento
2 FROM alocacao a
3 INNER JOIN departamento d ON d.id = a.id_departamento
4 INNER JOIN pessoa p ON p.id = a.id_pessoa
5 GROUP BY d.nome
6
7
Result Grid | Filter Rows: Export: Wrap Cell Content:
nome      total_por_departamento
Recursos Humanos 14000
Tecnologia 17000
Financeiro 10000
    
```

Figura 32: Total de salários por departamento

AVG: Contabiliza a média aritmética entre os valores de uma determinada coluna.

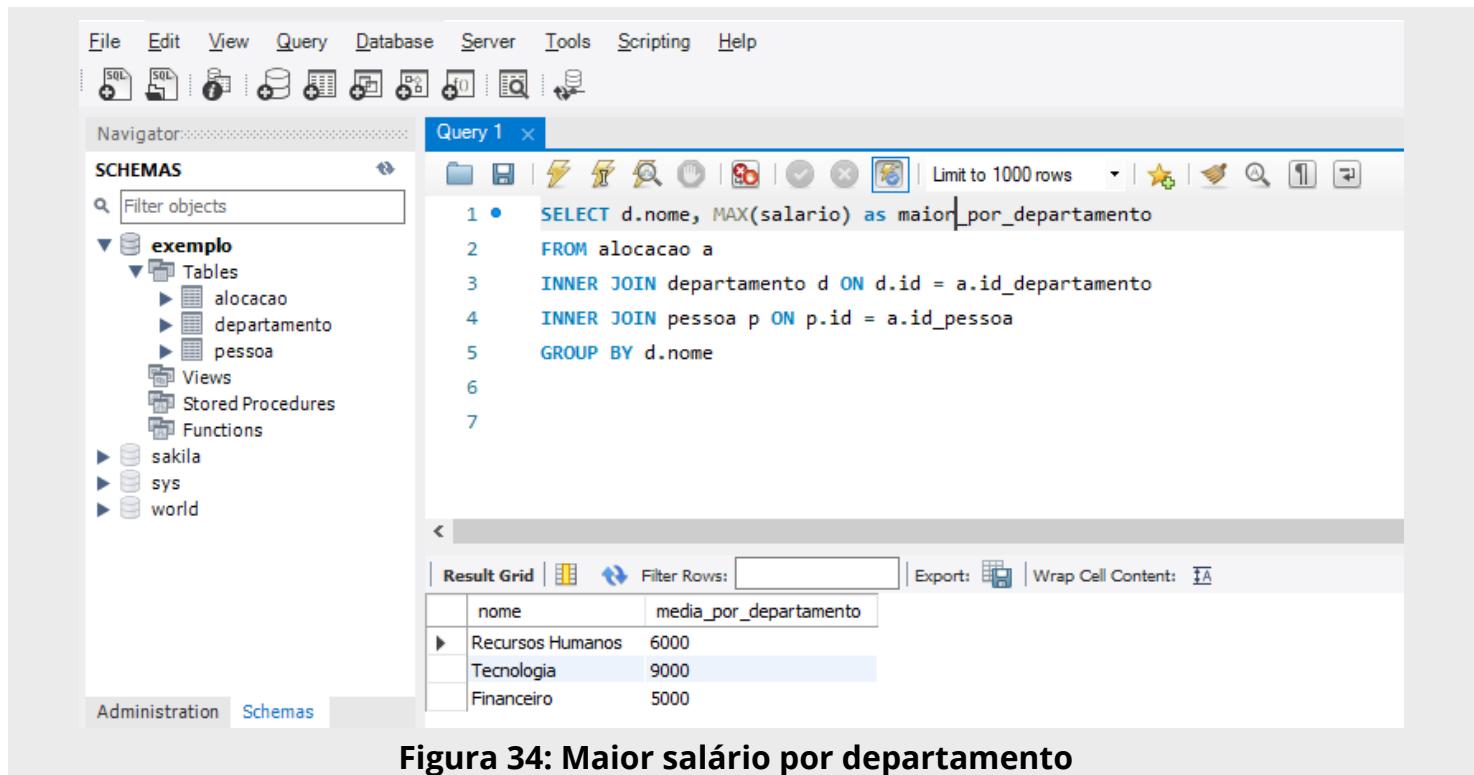


```

File Edit View Query Database Server Tools Scripting Help
Navigator: Query 1
SCHEMAS
Filter objects
exemplo
Tables
    alocacao
    departamento
    pessoa
Views
Stored Procedures
Functions
sakila
sys
world
Administration Schemas
1 • SELECT d.nome, AVG(salario) as media_por_departamento
2 FROM alocacao a
3 INNER JOIN departamento d ON d.id = a.id_departamento
4 INNER JOIN pessoa p ON p.id = a.id_pessoa
5 GROUP BY d.nome
6
7
Result Grid | Filter Rows: Export: Wrap Cell Content:
nome      total_por_departamento
Recursos Humanos 4666.6667
Tecnologia 5666.6667
Financeiro 5000.0000
    
```

Figura 33: Média de salários por departamento

MAX: Traz o maior valor de uma determinada coluna.

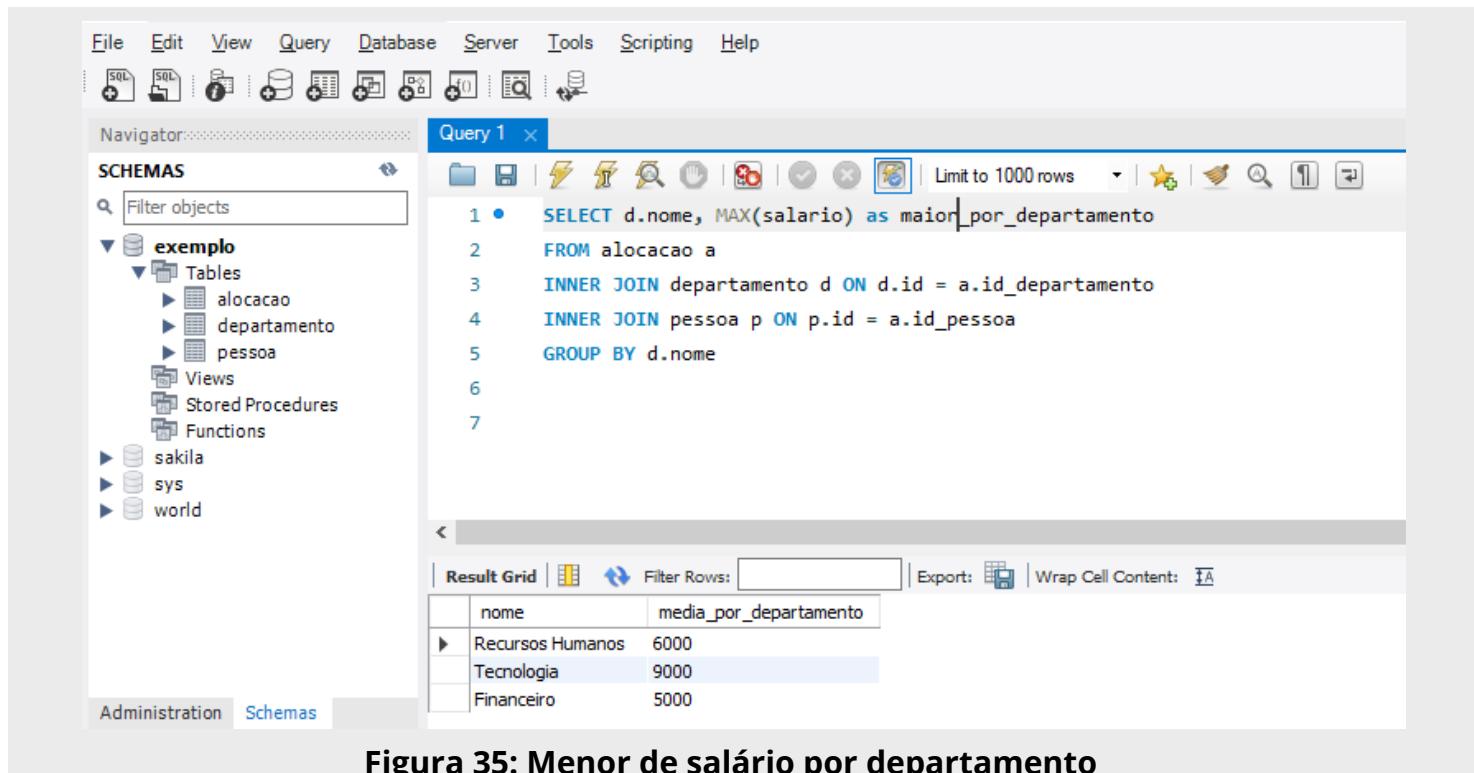


```

File Edit View Query Database Server Tools Scripting Help
SQL SQL+ Scripts Database Tools Help
Navigator
SCHEMAS
Filter objects
exemplo
Tables
alocacao
departamento
pessoa
Views
Stored Procedures
Functions
sakila
sys
world
Administration Schemas
Query 1
SELECT d.nome, MAX(salario) as maior_por_departamento
FROM alocacao a
INNER JOIN departamento d ON d.id = a.id_departamento
INNER JOIN pessoa p ON p.id = a.id_pessoa
GROUP BY d.nome
Result Grid Filter Rows: Export: Wrap Cell Content:
nome media_por_departamento
Reursos Humanos 6000
Tecnologia 9000
Financeiro 5000
    
```

Figura 34: Maior salário por departamento

MIN: Retorna o menor valor de uma determinada coluna.



```

File Edit View Query Database Server Tools Scripting Help
SQL SQL+ Scripts Database Tools Help
Navigator
SCHEMAS
Filter objects
exemplo
Tables
alocacao
departamento
pessoa
Views
Stored Procedures
Functions
sakila
sys
world
Administration Schemas
Query 1
SELECT d.nome, MIN(salario) as menor_por_departamento
FROM alocacao a
INNER JOIN departamento d ON d.id = a.id_departamento
INNER JOIN pessoa p ON p.id = a.id_pessoa
GROUP BY d.nome
Result Grid Filter Rows: Export: Wrap Cell Content:
nome menor_por_departamento
Reursos Humanos 6000
Tecnologia 9000
Financeiro 5000
    
```

Figura 35: Menor de salário por departamento

Referências Bibliográficas

ELMASRI, R.; NAVATHE, S. **Sistemas de Banco de Dados [BV:PE]**.. 7. ed.. São Paulo: Pearson, 2018.

HEUSER, C. **Projeto de Banco de Dados. [BV:MB]**. 6^a. Ed.. Porto Alegre: ARTMED,2009.

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **Banco de Dados: implementação em SQL, PL/SQL e Oracle 11g [BV:PE]**. 1. ed.. São Paulo: Pearson, 2013.

RAMARKRISHMAN, R. **Sistemas de gerenciamento de banco de dados.[BE:MB]**. 3^a Ed.. Porto Alegre: McGraw-Hill, 2008.

MACHADO, Felipe N. R. **Banco de Dados - Projeto e Implementação [BV:MB]**. 3. ed.. São Paulo: Érica, 2014.

FONSECA, Cleber Costa da. **Implementação de banco de dados. Banco de Dados [BV:RE]**. 1. ed.. Rio de Janeiro: SESES, 2016.

ALVES, William Pereira. **Banco de Dados [BV:MB]**. 1º Ed. São Paulo: Érica, 2014.

NETO, Geraldo H. **MODELAGEM DE DADOS. [BV:RE]**. 1. ed.. Rio de Janeiro:SESES, 2015.

BALIEIRO, R. **Banco de Dados** [BV:RE]. 1.Ed. Rio de Janeiro: SESES, 2015.



Digital College

ENSINO DE HABILIDADES DIGITAIS

digitalcollege.com.br