

Grupo 3:

CÍCERO CALIL SALIM
DANILO JOSÉ DE SOUZA ARAÚJO
GUSTAVO PREZOTTI MORELLI
HEITOR RODRIGUES ARAUJO
THOMÁS KRIGER DE SOUZA

Universidade Vila Velha
Curso: Ciência da Computação
Disciplina: Banco de Dados II
Professor: Jean-Rémi Bourguet

Vila Velha
2025
Banco de Dados II — Projeto Segundo Bimestre
Modelos e Consultas SQL
Trabalho apresentado à disciplina de Banco de Dados II

Segue link do repositorio Github com projeto completo:
<https://github.com/KrigerThomas/agenda-contatos-bd2>

1. Criação do Banco de Dados

Uma agenda de contatos é composta por diversos usuários. Cada usuário possui nome completo, e-mail (único) e data de nascimento. Um usuário pode cadastrar vários contatos, e cada contato pertence a um único usuário. Cada contato possui uma data de criação, pode ser marcado como favorito, e pode ter diversos telefones, com número e tipo (celular, fixo, trabalho ou outro). Além disso, um contato pode ter um endereço associado, contendo rua, número, complemento, bairro, cidade, estado e CEP.

Os contatos também podem ser classificados em uma ou mais categorias (como família, trabalho, amigos), por meio de uma tabela associativa. Uma categoria pode estar vinculada a vários contatos. Isso permite uma organização mais flexível e personalizada da agenda.

```
1 •   CREATE DATABASE IF NOT EXISTS agenda_contatos;
2 •   USE agenda_contatos;
3
4 •   CREATE TABLE IF NOT EXISTS usuario (
5     id_usuario INT AUTO_INCREMENT PRIMARY KEY,
6     nome_completo VARCHAR(100) NOT NULL,
7     email VARCHAR(100) UNIQUE,
8     data_nascimento DATE
9   );
10
11 •  CREATE TABLE IF NOT EXISTS contato (
12     id_contato INT AUTO_INCREMENT,
13     favorito BOOLEAN NOT NULL DEFAULT FALSE,
14     data_criacao DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
15     Usuario_idUsuario INT,
16     PRIMARY KEY(id_contato, Usuario_idUsuario),
17     FOREIGN KEY (Usuario_idUsuario) REFERENCES Usuario (id_usuario)
18   );
```

```
20 • Ⓜ CREATE TABLE IF NOT EXISTS telefone (
21     id_telefone INT AUTO_INCREMENT PRIMARY KEY,
22     id_contato INT NOT NULL,
23     numero VARCHAR(20) NOT NULL,
24     tipo ENUM('celular', 'fixo', 'trabalho', 'outro') NOT NULL,
25     FOREIGN KEY (id_contato) REFERENCES contato(id_contato)
26         ON DELETE CASCADE
27 );
28
29 • Ⓜ CREATE TABLE IF NOT EXISTS endereco (
30     id_endereco INT AUTO_INCREMENT PRIMARY KEY,
31     id_contato INT NOT NULL,
32     rua VARCHAR(100) NOT NULL,
33     numero VARCHAR(10),
34     complemento VARCHAR(50),
35     bairro VARCHAR(50),
36     cidade VARCHAR(50) NOT NULL,
37     estado CHAR(2) NOT NULL,
38     cep CHAR(9) NOT NULL,
39     FOREIGN KEY (id_contato) REFERENCES contato(id_contato)
40         ON DELETE CASCADE
41 );
42
43 • Ⓜ CREATE TABLE IF NOT EXISTS categoria (
44     id_categoria INT AUTO_INCREMENT PRIMARY KEY,
45     nome_categoria VARCHAR(50) NOT NULL UNIQUE
46 );
47
48 • Ⓜ CREATE TABLE IF NOT EXISTS contato_categoria (
49     id_contato INT NOT NULL,
50     id_categoria INT NOT NULL,
51     PRIMARY KEY (id_contato, id_categoria),
52     FOREIGN KEY (id_contato) REFERENCES contato(id_contato)
53         ON DELETE CASCADE,
54     FOREIGN KEY (id_categoria) REFERENCES categoria(id_categoria)
55         ON DELETE CASCADE
56 );
```


PROCEDURES E API PARA POVOAMENTO DO BANCO DE DADOS

Para a composição e teste do banco de dados em desenvolvimento, foi adotada a estratégia de utilização de múltiplas procedures responsáveis pelo povoamento de dados simulados. Cada procedure foi cuidadosamente elaborada para implementar uma lógica específica, visando a geração de dados consistentes, variados e coerentes com as regras de negócio previamente estabelecidas.

Essa abordagem permitiu automatizar o processo de inserção de registros em diferentes tabelas, assegurando não apenas a integridade referencial, mas também a representatividade das diversas situações que poderão ser encontradas em ambiente produtivo. A utilização de procedimentos armazenados distintos para cada cenário de inserção contribuiu para modularizar o código, facilitar a manutenção e possibilitar testes direcionados de cada uma das lógicas implementadas.

2.1. Povoar Contatos

Esta procedure chamada povoar_contato tem como finalidade inserir automaticamente um número especificado de registros simulados na tabela contato. Ela recebe como parâmetro de entrada o número de contatos que devem ser inseridos e, a partir disso, executa uma repetição controlada por uma variável contadora chamada i, que é inicializada com o valor 1.

Enquanto o valor de i for menor ou igual ao número total de contatos indicado, a procedure realiza uma inserção na tabela contato. Para isso, preenche o campo favorito com um valor aleatório entre 0 e 1, obtido através da função RAND() que gera um número aleatório de ponto flutuante, e da função ROUND() que arredonda esse valor para que fique restrito a um desses dois inteiros.

Além disso, para o campo Usuario_idUsuario, a procedure seleciona aleatoriamente um usuário existente na tabela usuario. Isso é feito utilizando a instrução SELECT id_usuario FROM usuario ORDER BY RAND() LIMIT 1, que embaralha os registros e seleciona um único identificador de usuário.

Após a inserção, a variável i é incrementada e o processo se repete até que o número de contatos inseridos seja igual ao especificado inicialmente. Assim, a procedure automatiza a criação de dados simulados, garantindo variedade e aleatoriedade, tanto no campo booleano favorito quanto na associação a usuários já cadastrados, sendo uma ferramenta eficiente para testes e validação do banco de dados.

```
39      -- Stored procedure tabela contato (300)
40      DELIMITER $$ 
41 •  CREATE PROCEDURE povoar_contato(IN num_contatos INT)
42 •  BEGIN
43      DECLARE i INT DEFAULT 1;
44
45      WHILE i <= num_contatos DO
46          INSERT INTO contato (favorito, Usuario_idUsuario)
47          VALUES (
48              ROUND(RAND()),
49              (SELECT id_usuario FROM usuario ORDER BY RAND() LIMIT 1)
50          );
51          SET i = i + 1;
52      END WHILE;
53  END $$ 
54  DELIMITER ;
55
56 •  CALL povoar_contato(300);
57
58 •  SELECT *
59   FROM Contato;
```

2.2. Povoar Categoria

A procedure povoar_categoria foi criada com o objetivo de inserir automaticamente um número determinado de registros na tabela categoria. Ela recebe como parâmetro de entrada o valor num_categorias, que define quantos registros devem ser inseridos. Inicialmente, é declarada uma variável inteira chamada i, que começa com o valor 1 e serve como contador para controlar as inserções.

Enquanto o valor de i for menor ou igual ao número total de categorias especificado, a procedure executa uma inserção na tabela categoria. O campo nome_categoria é preenchido com uma string que resulta da concatenação da palavra "Categoria" com o número correspondente ao contador i. Essa concatenação é feita utilizando a função CONCAT('Categoria ', i), gerando nomes como "Categoria 1", "Categoria 2" e assim por diante, até que o número de categorias desejado seja atingido.

Após cada inserção, a variável i é incrementada em uma unidade para que o processo continue até completar todas as inserções necessárias. Dessa forma, a procedure automatiza a criação de múltiplos registros com nomes distintos e numerados de forma sequencial, o que facilita a geração de dados simulados para testes e validações no banco de dados.

```
84      -- Stored procedure tabela categoria (300)
85      DELIMITER $$ 
86 •  CREATE PROCEDURE povoar_categoria(IN num_categorias INT)
87      BEGIN
88          DECLARE i INT DEFAULT 1;
89
90          WHILE i <= num_categorias DO
91              INSERT INTO categoria(nome_categoria)
92                  VALUES (
93                      CONCAT('Categoria ', i)
94                  );
95              SET i = i + 1;
96          END WHILE;
97      END $$ 
98      DELIMITER ;
99
100 •   CALL povoar_categoria(300);
101
102 •   SELECT *
103     FROM categoria;
```

2.3. Povoar Telefone

A procedure povoar_telefone foi criada com o objetivo de inserir automaticamente uma quantidade determinada de registros na tabela telefone. Ela recebe como parâmetro o valor num_telefones, que define quantos registros devem ser gerados. Inicialmente, declara-se uma variável inteira i, com valor inicial 1, que será utilizada como contador para controlar o número de inserções realizadas.

Enquanto o valor de i for menor ou igual ao número total de telefones especificado, a procedure executa uma inserção na tabela telefone, preenchendo três campos: id_contato, numero e tipo. O campo id_contato é preenchido por meio da seleção aleatória de um registro existente na tabela contato, utilizando a instrução SELECT id_contato FROM contato ORDER BY RAND() LIMIT 1, que embaralha os registros e seleciona apenas um. O campo numero é preenchido por meio da função CONCAT, que concatena o código de área fixo (27) 9 com um número aleatório de oito dígitos, gerado por LPAD(FLOOR(RAND() * 100000000), 8, '0'), garantindo assim que cada número de telefone seja composto adequadamente com o DDD e a quantidade correta de dígitos.

Já o campo tipo é preenchido com um dos quatro valores possíveis: 'celular', 'fixo', 'trabalho' ou 'outro'. A escolha entre essas opções é feita aleatoriamente pela função ELT(FLOOR((RAND() * 4) + 1), 'celular', 'fixo', 'trabalho', 'outro'), que gera um número aleatório entre 1 e 4 e, com base nele, seleciona um dos quatro tipos disponíveis. Após a inserção de cada registro, o valor da variável i é incrementado em uma unidade, permitindo que o loop prossiga até que todas as inserções sejam concluídas. Assim, a procedure automatiza a geração de dados variados e realistas para a tabela telefone, assegurando a diversidade necessária para a realização de testes e validações no sistema.

```
61      -- Stored procedure tabela telefone (350)
62      DELIMITER $$ 
63 •   CREATE PROCEDURE povoar_telefone(IN num_telefones INT)
64  BEGIN
65      DECLARE i INT DEFAULT 1;
66
67      WHILE i <= num_telefones DO
68          INSERT INTO telefone (id_contato, numero, tipo)
69          VALUES (
70              (SELECT id_contato FROM contato ORDER BY RAND() LIMIT 1),
71              CONCAT('(27) 9', LPAD(FLOOR(RAND() * 100000000), 8, '0')),
72              ELT(FLOOR((RAND() * 4) + 1), 'celular', 'fixo', 'trabalho', 'outro')
73          );
74          SET i = i + 1;
75      END WHILE;
76  END $$ 
77  DELIMITER ;
78
79 •   CALL povoar_telefone(350);
80
81 •   SELECT *
82     FROM telefone;
```

2.4. Povoar Entidade Associativa entre Contato e Categoria

A procedure completar_contato_categoria foi criada com o objetivo de inserir automaticamente um número determinado de registros na tabela contato_categoria, que faz a associação entre contatos e categorias. Ela recebe como parâmetro o valor num_contato_categorias, que define quantas associações devem ser criadas. Inicialmente, é declarada uma variável inteira chamada i, que é inicializada com o valor 1 e funciona como contador para controlar o número de inserções realizadas.

Enquanto o valor de i for menor ou igual ao número total de associações indicado, a procedure executa uma inserção na tabela contato_categoria, preenchendo os campos id_contato e id_categoria. O campo id_contato é preenchido com a seleção aleatória de um contato existente na tabela contato, utilizando a instrução SELECT id_contato FROM contato ORDER BY RAND() LIMIT 1, que embaralha os registros e escolhe apenas um. De forma análoga, o campo id_categoria é preenchido com a seleção aleatória de um registro da tabela categoria, utilizando a instrução SELECT id_categoria FROM categoria ORDER BY RAND() LIMIT 1, garantindo que a associação seja realizada com uma categoria existente e também de forma aleatória.

Após cada inserção, a variável i é incrementada para que o processo continue até que todas as associações desejadas sejam realizadas. Assim, essa procedure automatiza a criação de relacionamentos entre contatos e categorias de maneira eficiente, gerando dados variados e realistas, o que facilita a realização de testes, validações e simulações no banco de dados.

```
106      -- Stored procedure tabela contato_categoria (300)
107      DELIMITER $$*
108  •  CREATE PROCEDURE completar_contato_categoria(IN num_contato_categorias INT)
109  •  BEGIN
110      DECLARE i INT DEFAULT 1;
111
112      WHILE i <= num_contato_categorias DO
113          INSERT INTO contato_categoria(id_contato, id_categoria)
114          VALUES (
115              (SELECT id_contato FROM contato ORDER BY RAND() LIMIT 1),
116              (SELECT id_categoria FROM categoria ORDER BY RAND() LIMIT 1)
117          );
118          SET i = i + 1;
119      END WHILE;
120  END $$*
121  DELIMITER ;
122
123  •  CALL completar_contato_categoria(300);
124
125  •  SELECT *
126      FROM contato_categoria;
```

2.5. Povoar Usuários

A procedure povoar_usuarios foi criada com o objetivo de inserir automaticamente um número determinado de registros na tabela usuario. Ela recebe como parâmetro o valor num_usuarios, que define quantos usuários devem ser gerados. Inicialmente, declara-se uma variável inteira i, com valor inicial igual a 1, que será utilizada como contador para controlar a quantidade de inserções realizadas.

Enquanto o valor de i for menor ou igual ao número total de usuários especificado, a procedure executa uma inserção na tabela usuario, preenchendo os campos nome_completo, email e data_nascimento. O campo nome_completo é preenchido com a concatenação da palavra "Usuário" seguida do número correspondente ao contador i, utilizando a função CONCAT('Usuário ', i), garantindo assim a criação de nomes distintos como "Usuário 1", "Usuário 2" e assim por diante. O campo email é preenchido de forma semelhante, concatenando a palavra "usuario" com o número do contador i e o domínio "@exemplo.com", resultando em e-mails como "usuario1@exemplo.com".

Já o campo data_nascimento é preenchido por meio da função DATE_SUB, que subtrai um número de anos da data atual, definida por CURDATE(). A quantidade de anos subtraída é calculada pela expressão (18 + MOD(i, 30)), ou seja, é sempre no mínimo 18 anos, variando até 47 anos, conforme o resto da divisão do contador i por 30. Assim, para cada novo usuário, é gerada uma data de nascimento diferente dentro de um intervalo coerente de idades. Após a inserção de cada registro, a variável i é incrementada em uma unidade, permitindo que o processo continue até que todas as inserções sejam concluídas. Dessa forma, a procedure automatiza a geração de dados variados e realistas para a tabela usuario, facilitando testes e validações no banco de dados.

```
1      -- Exemplos de Stored Procedures para popular o banco
2
3      -- Stored Procedure tabela Usuario / INSERINDO UM MILHÃO DE REGISTROS PARA TESTAR
4  DELIMITER $$|
5 • CREATE PROCEDURE povoar_usuarios(IN num_usuarios INT)
6 BEGIN
7     DECLARE i INT DEFAULT 1;
8
9     WHILE i <= num_usuarios DO
10        INSERT INTO usuario (nome_completo, email, data_nascimento)
11        VALUES (
12            CONCAT('Usuário ', i),
13            CONCAT('usuario', i, '@exemplo.com'),
14            DATE_SUB(CURDATE(), INTERVAL (18 + MOD(i, 30)) YEAR)
15        );
16        SET i = i + 1;
17    END WHILE;
18 END $$|
19 DELIMITER ;
20
21 • SHOW VARIABLES LIKE 'secure_file_priv';
```

2.6: Importação de Dados para CSV

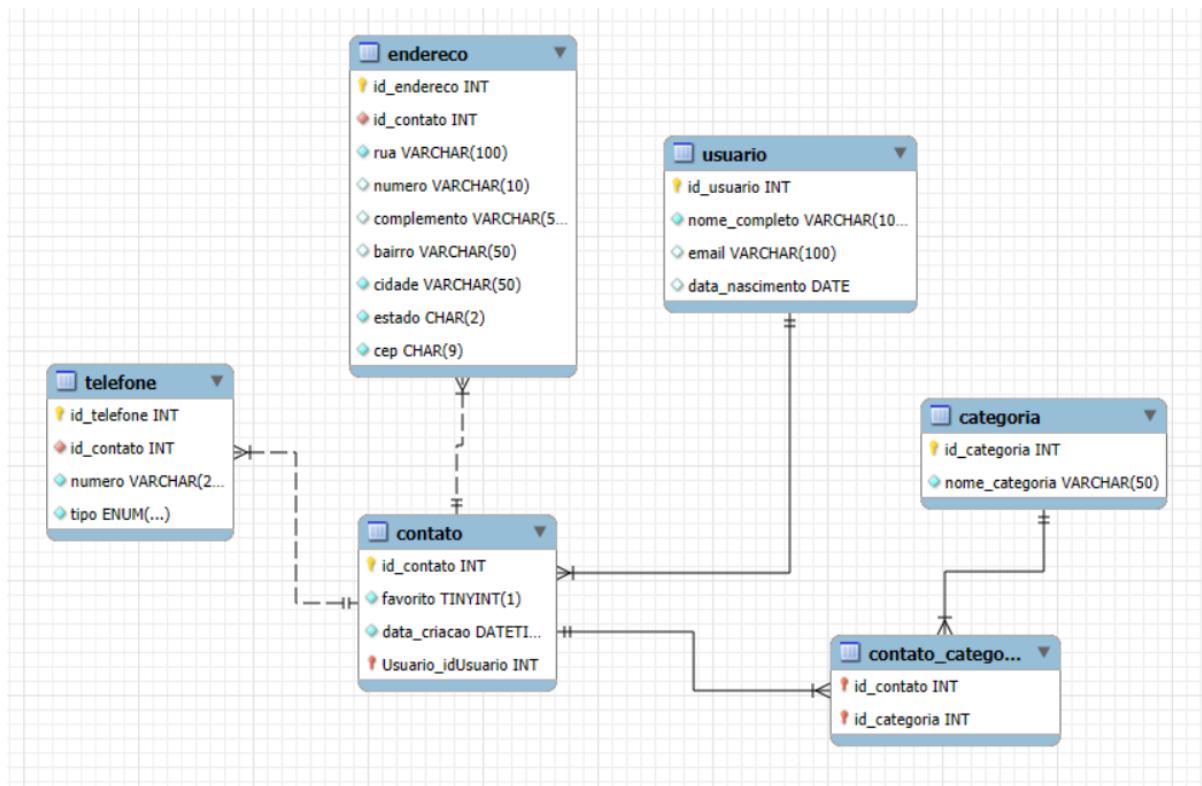
Foi desenvolvido um script em Python com o objetivo de realizar a exportação de todos os dados inseridos na tabela usuario, que totalizam um milhão de registros, para um arquivo no formato CSV. A utilização deste script teve como principal finalidade facilitar a reinserção e a manipulação desses dados em diferentes contextos, como novos testes, migrações de ambiente ou análises externas. A escolha pelo formato CSV se deu pela sua simplicidade, ampla compatibilidade com diversas ferramentas de processamento de dados e eficiência no armazenamento de grandes volumes de informações. Assim, o script automatiza a leitura dos registros diretamente do banco de dados e gera um arquivo estruturado e pronto para ser utilizado sempre que necessário, evitando a necessidade de repovoar a tabela a partir de procedures, o que poderia demandar mais tempo e recursos computacionais.

```
1 import csv
2 import mysql.connector
3
4 # Conexão com o banco de dados
5 conexao = mysql.connector.connect(
6     host='localhost',          # ou outro host se for remoto
7     user='root',              # substitua pelo seu usuário
8     password='Senha',         # substitua pela sua senha
9     database='agenda_contatos' # nome do seu banco
10 )
11
12 cursor = conexao.cursor()
13
14 # Consulta para buscar os dados da tabela usuario
15 query = "SELECT id_usuario, nome_completo, email, data_nascimento FROM usuario"
16 cursor.execute(query)
17
18 # Caminho do arquivo CSV de saída
19 caminho_csv = r"C:/csv/usuarios.csv"
20
21 # Escreve os dados no arquivo CSV
22 with open(caminho_csv, "w", newline='', encoding='utf-8') as arquivo_csv:
23     escritor = csv.writer(arquivo_csv)
24     escritor.writerow(['id_usuario', 'nome_completo', 'email', 'data_nascimento']) # cabecalhos
25
26     for linha in cursor:
27         escritor.writerow(linha)
28
29 # Finaliza conexões
30 cursor.close()
31 conexao.close()
32
33 print("Exportação concluída com sucesso para:", caminho_csv)
```

O comando LOAD DATA LOCAL INFILE é usado para importar dados de um arquivo CSV chamado usuarios.csv, localizado no diretório C:/csv/, diretamente para a tabela usuario no banco de dados. Os campos no arquivo CSV estão separados por vírgula, delimitados por aspas simples, e cada linha é terminada por uma quebra de linha. A cláusula IGNORE 1 ROWS indica que a primeira linha do arquivo (normalmente o cabeçalho) será ignorada. Os dados são inseridos nas colunas id_usuario, nome_completo, email e data_nascimento. Após esse processo, o comando DELETE FROM usuario apaga todos os registros da tabela, e o SELECT * FROM usuario exibe o conteúdo da tabela, que, nesse ponto, estará vazia devido ao DELETE anterior.

```
24      -- CALL povoar_usuarios(1000000); INSERINDO UM MILHÃO DE REGISTROS NA TABELA USUARIO
25
26 •  LOAD DATA LOCAL INFILE 'C:/csv/usuarios.csv'
27   INTO TABLE usuario
28   FIELDS TERMINATED BY ','
29   ENCLOSED BY ""
30   LINES TERMINATED BY '\n'
31   IGNORE 1 ROWS
32   (id_usuario, nome_completo, email, data_nascimento);
33
34 •  DELETE FROM usuario;
35 •  SELECT *
36   FROM usuario;
```

Modelo Lógico



QUERRIES

```
1 •  SELECT * FROM USUARIO;
2
3      -- Consultas_Simples_Heitor
4
5      -- Consulta 1:
6
7 •  SELECT * FROM usuario
8      WHERE data_nascimento > '2000-01-01'
9      AND nome_completo LIKE 'Usuario 9%';
10
11     -- Consulta 2:
12
13 •  SELECT YEAR(data_nascimento) as ano, COUNT(*) as total
14      FROM usuario
15      GROUP BY YEAR(data_nascimento)
16      HAVING COUNT(*) > 5000;
```

```
18      -- Consulta 3: (Muito rápida)
19
20 •   SELECT * FROM usuario
21     WHERE email LIKE 'usuario8888%'
22     ORDER BY data_nascimento;
23
24      -- Consulta 4:
25
26 •   SELECT nome_completo, email
27     FROM usuario
28     WHERE id_usuario BETWEEN 100000 AND 200000;
29
30      -- Consulta 5:
31
32 •   SELECT id_usuario, nome_completo
33     FROM usuario
34     WHERE email LIKE '%99999';
```

```
38      -- Consulta 1
39 •  CREATE VIEW vw_contatos_favoritos_completos AS
40    SELECT
41      u.id_usuario,
42      u.nome_completo AS nome_usuario,
43      u.email,
44      c.id_contato,
45      c.data_criacao,
46      t.numero AS telefone,
47      t.tipo AS tipo_telefone,
48      e.rua,
49      e.numero AS numero_endereco,
50      e.complemento,
51      e.bairro,
52      e.cidade,
53      e.estado,
54      e.cep,
55      GROUP_CONCAT(cat.nome_categoria SEPARATOR ', ') AS categorias
```

```
56   FROM contato c
57   JOIN usuario u ON u.id_usuario = c.Usuario_idUsuario
58   LEFT JOIN telefone t ON t.id_contato = c.id_contato
59   LEFT JOIN endereco e ON e.id_contato = c.id_contato
60   LEFT JOIN contato_categoria cc ON cc.id_contato = c.id_contato
61   LEFT JOIN categoria cat ON cat.id_categoria = cc.id_categoria
62   WHERE c.favorito = TRUE
63   GROUP BY
64       u.id_usuario, u.nome_completo, u.email,
65       c.id_contato, c.data_criacao,
66       t.numero, t.tipo,
67       e.rua, e.numero, e.complemento, e.bairro, e.cidade, e.estado, e.cep;
68
69 •  SELECT *
70   FROM vw_contatos_favoritos_completos;
```

```
72      -- Consulta 2
73
74 •  SELECT
75      id_usuario,
76      nome_completo,
77      email,
78      LOWER(email) AS email_lower,
79      FLOOR(DATEDIFF(CURDATE(), data_nascimento) / 365.25) AS idade,
80      (SELECT COUNT(*) FROM contato c WHERE c.usuario_idusuario = u.id_usuario) AS total_contatos
81  FROM usuario u
82  WHERE YEAR(data_nascimento) BETWEEN 1970 AND 2005
83      AND LOWER(email) LIKE '%@exemplo.com'
84  ORDER BY idade DESC, CHAR_LENGTH(nome_completo) DESC;
```

```
88      -- Consulta 3
89
90 •  SELECT
91      id_usuario,
92      nome_completo,
93      email,
94      DATE_FORMAT(data_nascimento, '%W, %M %Y') AS data_formatada,
95      (SELECT COUNT(*) FROM contato c WHERE c.usuario_idusuario = u.id_usuario) AS total_contatos
96  FROM usuario u
97  WHERE LOWER(email) LIKE '%@exemplo.com'
98      AND MONTH(data_nascimento) IN (1, 2, 3, 4, 5, 6)
99  ORDER BY total_contatos DESC, data_formatada;
```

INDEX

```
1   -- Index primeira consulta:  
2 • CREATE INDEX idx_usuarios_data_nome ON usuario(data_nascimento, nome_completo);  
3  
4   -- Index segunda consulta:  
5 • CREATE INDEX idx_usuarios_ano_nascimento ON usuario(data_nascimento);  
6  
7   -- Index terceira consulta:  
8 • CREATE INDEX idx_usuarios_email_data ON usuario(email, data_nascimento);  
9  
10  -- Index quarta consulta:  
11 • CREATE INDEX idx_usuarios_faixa_id ON usuario(id_usuario);  
12  
13  -- Index quinta consulta:  
14 • CREATE INDEX idx_usuarios_email ON usuario(email);  
15  
16  -- Índices para joins  
17 • CREATE INDEX idx_contato_usuario ON contato(Usuario_idUsuario);  
18 • CREATE INDEX idx_telefone_contato ON telefone(id_contato);  
19 • CREATE INDEX idx_endereco_contato ON endereco(id_contato);  
20 • CREATE INDEX idx_contato_categoria_contato ON contato_categoria(id_contato);  
21 • CREATE INDEX idx_contato_categoria_categoria ON contato_categoria(id_categoria);
```

```
23      -- Outros Index:  
24 •  CREATE INDEX idx_usuario_data_email  
25    ON usuario (data_nascimento, email);  
26  
27 •  CREATE INDEX idx_usuario_data_nascimento ON usuario (data_nascimento);
```

TRIGGERS

```
1  -- Parte 1: Função UDF
2  DELIMITER $$ 
3 • CREATE FUNCTION verificar_maioridade(data_nascimento DATE)
4  RETURNS BOOLEAN
5  DETERMINISTIC
6  BEGIN
7      DECLARE idade INT;
8
9      -- Checa se data_nascimento é NULL ou '0000-00-00'
10     IF data_nascimento IS NULL OR data_nascimento = '0000-00-00' THEN
11         RETURN FALSE;
12     END IF;
13
14     SET idade = FLOOR(DATEDIFF(CURDATE(), data_nascimento) / 365.25);
15     RETURN idade >= 18;
16 END $$ 
17 DELIMITER ;
```

```
20 •      -- Parte 2: Tabela de Log
21   CREATE TABLE IF NOT EXISTS log_insercao_contato (
22     id_log INT AUTO_INCREMENT PRIMARY KEY,
23     id_contato INT NOT NULL,
24     data_insercao DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
25     mensagem VARCHAR(255)
26   );
```

```
28      -- Parte 3: Trigger
29      DELIMITER $$*
30  •   CREATE TRIGGER trigger_insercao_contato
31      AFTER INSERT ON contato
32      FOR EACH ROW
33  •   BEGIN
34      INSERT INTO log_insercao_contato (id_contato, mensagem)
35          VALUES (NEW.id_contato, CONCAT('Contato inserido com ID: ', NEW.id_contato));
36  END $$*
37  DELIMITER ;
38
39  •   SELECT id_usuario, nome_completo, verificar_maioridade(data_nascimento) AS maior
40      FROM usuario;
41
42  •   INSERT INTO contato (favorito, usuario_idusuario) VALUES (FALSE, 1);
43
44  •   SELECT * FROM log_insercao_contato;
```

RETORNO DAS QUERYS

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows: |

id_usuario	nome_completo	email	data_nascimento
90	Usuário 90	usuario90@exemplo.com	2007-05-23
91	Usuário 91	usuario91@exemplo.com	2006-05-23
92	Usuário 92	usuario92@exemplo.com	2005-05-23
93	Usuário 93	usuario93@exemplo.com	2004-05-23
94	Usuário 94	usuario94@exemplo.com	2003-05-23
95	Usuário 95	usuario95@exemplo.com	2002-05-23
96	Usuário 96	usuario96@exemplo.com	2001-05-23
97	Usuário 97	usuario97@exemplo.com	2000-05-23
900	Usuário 900	usuario900@exemplo.com	2007-05-23
901	Usuário 901	usuario901@exemplo.com	2006-05-23
902	Usuário 902	usuario902@exemplo.com	2005-05-23
903	Usuário 903	usuario903@exemplo.com	2004-05-23
904	Usuário 904	usuario904@exemplo.com	2003-05-23
905	Usuário 905	usuario905@exemplo.com	2002-05-23
906	Usuário 906	usuario906@exemplo.com	2001-05-23
907	Usuário 907	usuario907@exemplo.com	2000-05-23
930	Usuário 930	usuario930@exemplo.com	2007-05-23
931	Usuário 931	usuario931@exemplo.com	2006-05-23

usuario 1 × Apply

Output:

Action Output

#	Time	Action	Message
1	23:47:03	USE agenda_contatos	0 row(s) affected
2	23:47:34	SELECT * FROM usuario WHERE data_nascimento > '2000-01-01' AND nome_completo LIKE 'Usuário 9%'	29656 row(s) returned

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

ano	total
2006	33334
2005	33334
2004	33334
2003	33334
2002	33334
2001	33334
2000	33334
1999	33334
1998	33334
1997	33334
1996	33333
1995	33333
1994	33333
1993	33333
1992	33333
1991	33333
1990	33333
1989	33333

Result 2 × Read Only Context Help Snippets

Output:

Action Output

#	Time	Action	Message	Duration / Fetch
4	23:48:21	EXPLAIN FORMAT=JSON SELECT * FROM usuario WHERE data_nascimento > '2000-01-01' AND nome_compl... OK		0.000 sec
5	23:48:45	SELECT YEAR(data_nascimento) as ano, COUNT(*) as total FROM usuario GROUP BY YEAR(data_nascimento) H... 30 row(s) returned		0.391 sec / 0.000 sec

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows: |

id_usuario	nome_completo	email	data_nascimento
888029	Usuário 888029	usuario888029@exemplo.com	1978-05-23
888059	Usuário 888059	usuario888059@exemplo.com	1978-05-23
888089	Usuário 888089	usuario888089@exemplo.com	1978-05-23
888119	Usuário 888119	usuario888119@exemplo.com	1978-05-23
888149	Usuário 888149	usuario888149@exemplo.com	1978-05-23
888179	Usuário 888179	usuario888179@exemplo.com	1978-05-23
888209	Usuário 888209	usuario888209@exemplo.com	1978-05-23
888239	Usuário 888239	usuario888239@exemplo.com	1978-05-23
888269	Usuário 888269	usuario888269@exemplo.com	1978-05-23
88829	Usuário 88829	usuario88829@exemplo.com	1978-05-23
888299	Usuário 888299	usuario888299@exemplo.com	1978-05-23
888329	Usuário 888329	usuario888329@exemplo.com	1978-05-23
888359	Usuário 888359	usuario888359@exemplo.com	1978-05-23
888389	Usuário 888389	usuario888389@exemplo.com	1978-05-23
888419	Usuário 888419	usuario888419@exemplo.com	1978-05-23
888449	Usuário 888449	usuario888449@exemplo.com	1978-05-23
888479	Usuário 888479	usuario888479@exemplo.com	1978-05-23
888509	Usuário 888509	usuario888509@exemplo.com	1978-05-23

usuário 3 ×

Output

#	Time	Action	Message	Duration / Fetch
7	23:49:04	EXPLAIN FORMAT=JSON SELECT YEAR(data_nascimento) as ano, COUNT(*) as total FROM usuario GROUP BY...	OK	0.000 sec
8	23:49:29	SELECT * FROM usuario WHERE email LIKE 'usuario888%'; ORDER BY data_nascimento	1111 row(s) returned	0.047 sec / 0.000 sec

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan | Context Help | Snippets | Apply | Revert

nome_completo email

usuário 100000	usuario100000@exemplo.com
usuário 100001	usuario100001@exemplo.com
usuário 100002	usuario100002@exemplo.com
usuário 100003	usuario100003@exemplo.com
usuário 100004	usuario100004@exemplo.com
usuário 100005	usuario100005@exemplo.com
usuário 100006	usuario100006@exemplo.com
usuário 100007	usuario100007@exemplo.com
usuário 100008	usuario100008@exemplo.com
usuário 100009	usuario100009@exemplo.com
usuário 100010	usuario100010@exemplo.com
usuário 100011	usuario100011@exemplo.com
usuário 100012	usuario100012@exemplo.com
usuário 100013	usuario100013@exemplo.com
usuário 100014	usuario100014@exemplo.com
usuário 100015	usuario100015@exemplo.com
usuário 100016	usuario100016@exemplo.com
usuário 100017	usuario100017@exemplo.com

usuário 4 < Context Help Snippets

Action Output

#	Time	Action	Message	Duration / Fetch
10	23:49:52	EXPLAIN FORMAT=JSON SELECT * FROM usuario WHERE email LIKE 'usuario88%' ORDER BY data_nascimento	OK	0.000 sec
11	23:50:12	SELECT nome_completo, email FROM usuario WHERE id_usuario BETWEEN 100000 AND 200000	100001 row(s) returned	0.000 sec / 0.062 sec

Result Grid

id_usuario	nome_completo
9999	Usuário 9999
19999	Usuário 19999
29999	Usuário 29999
39999	Usuário 39999
49999	Usuário 49999
59999	Usuário 59999
69999	Usuário 69999
79999	Usuário 79999
89999	Usuário 89999
99990	Usuário 99990
99991	Usuário 99991
99992	Usuário 99992
99993	Usuário 99993
99994	Usuário 99994
99995	Usuário 99995
99996	Usuário 99996
99997	Usuário 99997
99998	Usuário 99998

usuario 5 <

Output:

Action Output

#	Time	Action	Message	Duration / Fetch
13	23:50:31	EXPLAIN FORMAT=JSON SELECT nome_completo, email FROM usuario WHERE id_usuario BETWEEN 100000...	OK	0.000 sec
14	23:50:57	SELECT id_usuario, nome_completo FROM usuario WHERE email LIKE "%999%".	280 row(s) returned	0.453 sec / 0.000 sec

Context Help Snippets

Result Grid

Form Editor

Field Types

Query Stats

Execution Plan

vw_contatos_favoritos_completos

Output

Action Output

#	Time	Action	Message	Duration / Fetch
16	23:51:17	EXPLAIN FORMAT=JSON SELECT id_usuario, nome_completo FROM usuario WHERE email LIKE "%9999%"	OK	0.000 sec
17	23:51:49	SELECT * FROM vw_contatos_favoritos_completos	229 row(s) returned	0.000 sec / 0.000 sec

Result Grid

Form Editor

Field Types

Query Stats

Execution Plan

vw_contatos_favoritos_completos

Output

Action Output

#	Time	Action	Message	Duration / Fetch
19	23:52:03	EXPLAIN FORMAT=JSON SELECT * FROM vw_contatos_favoritos_completos	OK	0.000 sec
20	23:52:49	SELECT id_usuario, nome_completo, email, LOWER(email) AS email_lower, FLOOR(DATEDIFF(CUR...	933333 row(s) returned	1.172 sec / 1.735 sec

Result 8 ×

	id_usuario	nome_completo	email	data_formatada	total_contatos
▶	923517	Usuário 923517	usuario923517@exemplo.com	Friday, May 1980	1
	881667	Usuário 881667	usuario881667@exemplo.com	Friday, May 1980	1
	589347	Usuário 589347	usuario589347@exemplo.com	Friday, May 1980	1
	55317	Usuário 55317	usuario55317@exemplo.com	Friday, May 1980	1
	163857	Usuário 163857	usuario163857@exemplo.com	Friday, May 1980	1
	459597	Usuário 459597	usuario459597@exemplo.com	Friday, May 1980	1
	438057	Usuário 438057	usuario438057@exemplo.com	Friday, May 1980	1
	806271	Usuário 806271	usuario806271@exemplo.com	Friday, May 1986	1
	794481	Usuário 794481	usuario794481@exemplo.com	Friday, May 1986	1
	674241	Usuário 674241	usuario674241@exemplo.com	Friday, May 1986	1
	620781	Usuário 620781	usuario620781@exemplo.com	Friday, May 1986	1
	885081	Usuário 885081	usuario885081@exemplo.com	Friday, May 1986	1
	810291	Usuário 810291	usuario810291@exemplo.com	Friday, May 1986	1
	826041	Usuário 826041	usuario826041@exemplo.com	Friday, May 1986	1
	503691	Usuário 503691	usuario503691@exemplo.com	Friday, May 1986	1
	539601	Usuário 539601	usuario539601@exemplo.com	Friday, May 1986	1
	522171	Usuário 522171	usuario522171@exemplo.com	Friday, May 1986	1
	252921	Usuário 252921	usuario252921@exemplo.com	Friday, May 1986	1
	15931	Usuário 15931	usuario15931@exemplo.com	Friday, May 1986	1

Output

Action Output

#	Time	Action	Message	Duration / Fetch
22	23:53:16	EXPLAIN FORMAT=JSON SELECT	id_usuario, nome_completo, email, LOWER(email) AS email_lower, ... OK	0.000 sec
23	23:53:37	SELECT	id_usuario, nome_completo, email, DATE_FORMAT(data_nascimento, "%W, %M %Y") AS data_nascimento, ... 1000000 row(s) returned	2.921 sec / 0.235 sec

Result Grid

- Form Editor
- Field Types
- Query Stats
- Execution Plan

Read Only Context Help Snippets