**Documento de Requisitos do Sistema - Banco Malvader**

**(SRS - Software Requirements Specification)**

Este documento será um complemento ao trabalho, detalhando os requisitos funcionais, não funcionais e as especificações do sistema para orientar os grupos no desenvolvimento.

**1.1 Objetivo**

Este documento descreve os requisitos para o desenvolvimento do sistema uma aplicação bancária que gerencia contas, funcionários e clientes com persistência em um banco de dados MySQL. O sistema deve implementar funcionalidades avançadas de gerenciamento bancário, utilizando conceitos de Programação Orientada a Objetos (POO) e um banco de dados relacional. Este SRS será o guia para que seu grupo possa entender o escopo, as funcionalidades e os critérios de aceitação.

**1.2 Escopo**

O sistema permitirá:

* Autenticação segura de usuários (funcionários e clientes) com senha e OTP (One-Time Password).
* Gerenciamento de contas bancárias (Poupança, Corrente e Investimento) com operações como abertura, encerramento, depósito, saque, transferência e consulta.
* Registro de funcionários com hierarquia e controle de permissões.
* Geração de relatórios financeiros detalhados com exportação para múltiplos formatos.
* Persistência de dados em um banco de dados MySQL com gatilhos, procedimentos armazenados, visões e auditoria.

**1.3 Definições, Acrônimos e Abreviações**

* SRS: Software Requirements Specification (Especificação de Requisitos de Software).
* POO: Programação Orientada a Objetos.
* MySQL: Sistema de gerenciamento de banco de dados relacional.
* OTP: One-Time Password (Senha de Uso Único).
* DAO: Data Access Object (Objeto de Acesso a Dados).
* CPF: Cadastro de Pessoa Física.
* CC: Conta Corrente.
* CP: Conta Poupança.
* CI: Conta Investimento.

**1.4 Referências**

* Documentação oficial do MySQL (<https://dev.mysql.com/doc/>).
* Padrões de design de banco de dados relacional e POO.

**2. Descrição Geral**

**2.1 Perspectiva do Produto**

Um sistema bancário completo que simula operações reais de uma instituição financeira. Ele integra uma interface gráfica com um banco de dados MySQL, oferecendo funcionalidades para funcionários (gerenciamento de contas e relatórios) e clientes (operações financeiras).

**2.2 Funções do Produto**

* Autenticação: Login seguro com senha e OTP gerado dinamicamente.
* Gerenciamento de Contas: Abertura, encerramento, consulta e alteração de contas (CP, CC, CI).
* Operações Financeiras: Depósito, saque, transferência e extrato.
* Relatórios: Geração de relatórios financeiros com exportação para Excel e PDF.
* Auditoria: Registro de todas as ações críticas no sistema.
* Segurança: Validação de senhas fortes, limites de transações e permissões hierárquicas.

**2.3 Características dos Usuários**

* Funcionários: Usuários com conhecimento básico de sistemas bancários, responsáveis por gerenciar contas e relatórios. Incluem estagiários, atendentes e gerentes com diferentes níveis de permissão.
* Clientes: Usuários finais que realizam operações financeiras, com conhecimento mínimo de tecnologia.

**2.4 Restrições**

* O banco de dados deve ser implementado em MySQL.
* Todas as regras de negócio críticas (ex.: limites, validações) devem ser implementadas no banco de dados.
* O sistema deve suportar ao menos 100 contas e 1.000 transações sem degradação significativa de desempenho.

**2.5 Premissas e Dependências**

* Premissa: Os alunos têm acesso a um ambiente MySQL configurado.
* Dependência: Uso do driver JDBC para conexão com MySQL (ou equivalente na linguagem escolhida).

**3. Requisitos Específicos**

**3.1 Requisitos Funcionais**

***RF1 - Autenticação***

* ***RF1.1***: O sistema deve exibir uma tela de login com opções "Funcionário", "Cliente" e "Sair".
* ***RF1.2***: O login exige senha e OTP gerado pelo banco (válido por 5 minutos).
* ***RF1.3***: Tentativas de login devem ser registradas na tabela auditoria com data, hora, usuário e resultado (sucesso/falha).
* Critério de Aceitação: Login bem-sucedido redireciona ao menu correspondente; falhas após 3 tentativas bloqueiam o usuário por 10 minutos.

***RF2 - Menu Funcionário***

* RF2.1 - Abertura de Conta:
  + Suporte a CP, CC e CI com campos específicos (ver estrutura do banco).
  + Número da conta gerado automaticamente com dígito verificador.
  + Gatilho registra a abertura em auditoria.
* RF2.2 - Encerramento de Conta:
  + Requer senha de administrador e OTP.
  + Bloqueia se saldo < 0 ou dívidas pendentes.
  + Registra motivo em uma tabela de histórico.
* RF2.3 - Consulta de Dados:
  + Submenu com visões para Conta, Funcionário e Cliente.
  + Exibe dados calculados (ex.: score de crédito, projeção de rendimentos).
* RF2.4 - Alteração de Dados:
  + Alterações em Conta, Funcionário e Cliente com senha de administrador.
  + Registro de alterações em auditoria com valores antigos/novos.
* RF2.5 - Cadastro de Funcionários:
  + Requer permissão de gerente.
  + Código único gerado pelo banco.
  + Gatilho verifica limite de funcionários por agência (máximo 20).
* RF2.6 - Geração de Relatórios:
  + Relatórios de movimentações, inadimplência e desempenho de funcionários.
  + Exportação para Excel e PDF via front-end.
  + Dados obtidos de uma view específica.

***RF3 - Menu Cliente***

* RF3.1 - Operações de Conta:
  + Saldo: Exibe saldo e projeção de rendimentos (CP/CI) com senha e OTP.
  + Depósito: Limite diário de R$10.000 via trigger.
  + Saque: Verifica saldo + limite; aplica taxa para saques excessivos (>5/mês).
  + Transferência: Entre contas ou terceiros com validação e taxa.
  + Extrato: Últimas 50 transações ou por período, exportável.
  + Consultar Limite: Mostra limite atual e projeção.
* RF3.2 - Encerrar Sessão: Logout registrado em auditoria.

**3.2 Requisitos Não Funcionais**

* ***RNF1*** - Desempenho: Consultas devem retornar em < 2 segundos com 1.000 transações.
* ***RNF2*** - Escalabilidade: Suporte a 100 contas e 10.000 transações com índices adequados.
* ***RNF3*** - Segurança: Senhas criptografadas (MD5 ou superior), OTP e auditoria de ações.
* ***RNF4*** - Usabilidade: Interface gráfica intuitiva, com mensagens claras de erro/sucesso.
* ***RNF5*** - Portabilidade: Banco de dados MySQL compatível com versões 8.x; front-end independente de plataforma.

**3.3 Requisitos de Interface**

* ***RI1***: Tela de login com campos para senha e OTP.
* ***RI2***: Menus com botões para cada funcionalidade (Funcionário e Cliente).
* ***RI3***: Formulários para cadastro e alteração com validação de entrada.
* ***RI4***: Exibição de relatórios e extratos em formato tabular, com opção de exportação.

**3.4 Requisitos de Banco de Dados**

* ***RBD1***: Estrutura conforme seção 4, com tabelas, gatilhos, procedures e visões.
* ***RBD2***: Índices em campos de busca frequente (ex.: numero\_conta, data\_hora).
* ***RBD3***: Transações SQL para operações críticas (ex.: transferências).
* ***RBD4***: Validações no banco (ex.: limite de depósito, senha forte).

**4. Estrutura do Banco de Dados**

**4.1 Tabelas**

* usuario: id\_usuario (PK), nome, cpf (UNIQUE), data\_nascimento, telefone, tipo\_usuario, senha\_hash, otp\_ativo, otp\_expiracao.
* funcionario: id\_funcionario (PK), id\_usuario (FK), codigo\_funcionario (UNIQUE), cargo, id\_supervisor (FK).
* cliente: id\_cliente (PK), id\_usuario (FK), score\_credito.
* endereco: id\_endereco (PK), id\_usuario (FK), cep, local, numero\_casa, bairro, cidade, estado, complemento.
* agencia: id\_agencia (PK), nome, codigo\_agencia (UNIQUE), endereco\_id (FK).
* conta: id\_conta (PK), numero\_conta (UNIQUE), id\_agencia (FK), saldo, tipo\_conta, id\_cliente (FK), data\_abertura, status.
* conta\_poupanca: id\_conta\_poupanca (PK), id\_conta (FK, UNIQUE), taxa\_rendimento, ultimo\_rendimento.
* conta\_corrente: id\_conta\_corrente (PK), id\_conta (FK, UNIQUE), limite, data\_vencimento, taxa\_manutencao.
* conta\_investimento: id\_conta\_investimento (PK), id\_conta (FK, UNIQUE), perfil\_risco, valor\_minimo, taxa\_rendimento\_base.
* transacao: id\_transacao (PK), id\_conta\_origem (FK), id\_conta\_destino (FK), tipo\_transacao, valor, data\_hora, descricao.
* auditoria: id\_auditoria (PK), id\_usuario (FK), acao, data\_hora, detalhes.
* relatorio: id\_relatorio (PK), id\_funcionario (FK), tipo\_relatorio, data\_geracao, conteudo.

**4.2 Gatilhos**

* Atualização de saldo após transações.
* Validação de senha forte em usuario.
* Limite de depósito diário em transacao.

**4.3 Procedimentos Armazenados**

* gerar\_otp(id\_usuario): Gera OTP de 6 dígitos.
* calcular\_score\_credito(id\_cliente): Atualiza score com base em transações.

**4.4 Visões**

* vw\_resumo\_contas: Resumo de contas por cliente.
* vw\_movimentacoes\_recentes: Movimentações dos últimos 90 dias.

**5. Requisitos de Design**

* Arquitetura: MVC (Model-View-Controller) ou equivalente.
* Pacotes:
  + dao: Acesso ao banco de dados.
  + model: Representação das tabelas.
  + view: Interface gráfica (livre escolha).
  + controller: Lógica de negócios.
  + util: Conexão e utilitários.
* Conexão: Driver JDBC ou similar, com pool de conexões recomendado.

**6. Critérios de Aceitação**

* Todas as funcionalidades listadas em RF2 e RF3 implementadas e testadas.
* Banco de dados configurado com todas as tabelas, gatilhos, procedures e visões funcionando.
* Interface gráfica funcional, com validação de entrada e exportação de relatórios.
* Documentação com diagrama ER e descrição das regras de negócio.

**7. Entrega**

* Código: Organizado em pacotes, com comentários detalhando lógica e SQL.
* Documentação: Diagrama ER, instruções de instalação e descrição do sistema.
* Apresentação: Demonstração prática (5-10 minutos) mostrando autenticação, operações e relatórios.

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**Revisão dos requisitos técnicos**

**Acesso ao Programa**

* Autenticação Inicial:
  + Tela de login com autenticação multifator: senha e um código temporário gerado pelo banco de dados (OTP - One-Time Password).
  + Menu principal com opções: "Funcionário", "Cliente" e "Sair".
  + Registro de tentativas de login (com data, hora e resultado) em uma tabela de auditoria.

**Menu Funcionário**

1. Abertura de Conta:
   * Suporte a múltiplos tipos de conta: Conta Poupança (CP), Conta Corrente (CC) e Conta Investimento (CI).
   * Dados obrigatórios:
     + CP: agência, número da conta, nome, CPF, data de nascimento, telefone, endereço completo, senha, taxa de rendimento personalizada.
     + CC: mesma base + limite dinâmico (calculado com base no histórico de transações do cliente), data de vencimento e taxa de manutenção.
     + CI: mesma base + perfil de risco (baixo, médio, alto) e valor mínimo de investimento.
   * Validação no banco: CPF único, número de conta gerado automaticamente com dígito verificador (ex.: algoritmo de Luhn).
   * Gatilho para registrar a abertura em uma tabela de log.
2. **Encerramento de Conta:**
   * Requer senha de administrador e OTP.
   * Verificar saldo pendente e bloquear encerramento se houver dívidas.
   * Registrar motivo do encerramento (ex.: solicitação do cliente, inadimplência) em uma tabela de histórico.
   * Atualizar status da conta para "Encerrada" via procedure.
3. **Consulta de Dados:**
   * Submenu:
     + Conta: Tipo, nome, CPF, saldo atual, limite disponível, data de vencimento, histórico de transações (últimos 90 dias) e projeção de rendimentos (para CP e CI).
     + Funcionário: Código, cargo, nome, CPF, data de nascimento, telefone, endereço, número de contas abertas por ele e desempenho (média de valores movimentados).
     + Cliente: Nome, CPF, data de nascimento, telefone, endereço, score de crédito (calculado via procedure), e lista de contas ativas/inativas.
   * Consultas devem usar visões (views) para otimizar e proteger dados sensíveis.
4. **Alteração de Dados:**
   * Submenu:
     + Conta: Alterar limite (com validação de score de crédito), data de vencimento e taxa de rendimento/manutenção.
     + Funcionário: Alterar cargo (com restrição de nível hierárquico), telefone e endereço.
     + Cliente: Alterar telefone, endereço e senha (com validação de força da senha via trigger).
   * Todas as alterações registradas em uma tabela de auditoria com data, usuário responsável e valores antigos/novos.
5. **Cadastro de Funcionários:**
   * Dados: código único (gerado pelo banco), cargo (hierarquia: estagiário, atendente, gerente), nome, CPF, data de nascimento, telefone, endereço, senha criptografada (ex.: hash MD5 no banco).
   * Requer permissão de administrador (nível gerente ou superior).
   * Gatilho para verificar limite de funcionários por agência.
6. **Geração de Relatórios:**
   * Relatórios avançados:
     + Movimentações por período (filtro por data, tipo de transação e agência).
     + Clientes inadimplentes (saldo negativo ou limite excedido).
     + Desempenho de funcionários (contas abertas, valores movimentados).
   * Exportação para Excel e PDF via front-end, com dados obtidos de uma view específica.
   * Procedure para calcular métricas (ex.: total movimentado, média de transações por cliente).
7. **Sair: Retorna ao menu principal.**

**Menu Cliente**

1. **Operações de Conta:**
   * Submenu:
     + Saldo: Exibir saldo atual e projeção de rendimentos (CP e CI), com senha e OTP.
     + Depósito: Permitir depósitos com validação de valor máximo diário (ex.: R$10.000) via trigger.
     + Saque: Verificar saldo + limite (CC) ou saldo disponível (CP/CI), com registro de taxa para saques excessivos.
     + Transferência: Entre contas do mesmo cliente ou para terceiros (com validação de conta destino e taxa).
     + Extrato: Últimas 50 transações ou por período, exportável para Excel/PDF.
     + Consultar Limite: Limite atual e projeção de aumento com base no score de crédito.
   * Todas as operações registradas em uma tabela de transações com gatilho para atualizar o saldo.
2. **Encerrar Programa**: Finaliza a sessão do cliente com logout registrado no banco.

**Estrutura do Banco de Dados**

**Tabelas**

1. usuario
   * id\_usuario (PK, INT, AUTO\_INCREMENT)
   * nome (VARCHAR(100), NOT NULL)
   * cpf (VARCHAR(11), UNIQUE, NOT NULL)
   * data\_nascimento (DATE, NOT NULL)

telefone (VARCHAR(15), NOT NULL)

* + tipo\_usuario (ENUM('FUNCIONARIO', 'CLIENTE'), NOT NULL)
  + senha\_hash (VARCHAR(32), NOT NULL) - Armazena hash MD5 da senha
  + otp\_ativo (VARCHAR(6)) - Código OTP temporário
  + otp\_expiracao (DATETIME) - Data/hora de expiração do OTP

1. **funcionario**
   * id\_funcionario (PK, INT, AUTO\_INCREMENT)
   * id\_usuario (FK, INT, REFERENCES usuario(id\_usuario))
   * codigo\_funcionario (VARCHAR(20), UNIQUE, NOT NULL) - Gerado automaticamente
   * cargo (ENUM('ESTAGIARIO', 'ATENDENTE', 'GERENTE'), NOT NULL)
   * id\_supervisor (FK, INT, REFERENCES funcionario(id\_funcionario)) - Hierarquia
2. **cliente**
   * id\_cliente (PK, INT, AUTO\_INCREMENT)
   * id\_usuario (FK, INT, REFERENCES usuario(id\_usuario))
   * score\_credito (DECIMAL(5,2), DEFAULT 0) - Calculado dinamicamente
3. **endereco**
   * id\_endereco (PK, INT, AUTO\_INCREMENT)
   * id\_usuario (FK, INT, REFERENCES usuario(id\_usuario))
   * cep (VARCHAR(10), NOT NULL)
   * local (VARCHAR(100), NOT NULL)
   * numero\_casa (INT, NOT NULL)
   * bairro (VARCHAR(50), NOT NULL)
   * cidade (VARCHAR(50), NOT NULL)
   * estado (CHAR(2), NOT NULL)
   * complemento (VARCHAR(50)) - Opcional
   * Índice em cep para buscas rápidas
4. **agencia**
   * id\_agencia (PK, INT, AUTO\_INCREMENT)
   * nome (VARCHAR(50), NOT NULL)
   * codigo\_agencia (VARCHAR(10), UNIQUE, NOT NULL)
   * endereco\_id (FK, INT, REFERENCES endereco(id\_endereco))
5. **conta**
   * id\_conta (PK, INT, AUTO\_INCREMENT)
   * numero\_conta (VARCHAR(20), UNIQUE, NOT NULL) - Inclui dígito verificador
   * id\_agencia (FK, INT, REFERENCES agencia(id\_agencia))
   * saldo (DECIMAL(15,2), NOT NULL DEFAULT 0)
   * tipo\_conta (ENUM('POUPANCA', 'CORRENTE', 'INVESTIMENTO'), NOT NULL)
   * id\_cliente (FK, INT, REFERENCES cliente(id\_cliente))
   * data\_abertura (DATETIME, NOT NULL DEFAULT CURRENT\_TIMESTAMP)
   * status (ENUM('ATIVA', 'ENCERRADA', 'BLOQUEADA'), NOT NULL DEFAULT 'ATIVA')
   * Índice em numero\_conta
6. **conta\_poupanca**
   * id\_conta\_poupanca (PK, INT, AUTO\_INCREMENT)
   * id\_conta (FK, INT, REFERENCES conta(id\_conta), UNIQUE)
   * taxa\_rendimento (DECIMAL(5,2), NOT NULL) - Personalizada por conta
   * ultimo\_rendimento (DATETIME) - Última aplicação do rendimento
7. **conta\_corrente**
   * id\_conta\_corrente (PK, INT, AUTO\_INCREMENT)
   * id\_conta (FK, INT, REFERENCES conta(id\_conta), UNIQUE)
   * limite (DECIMAL(15,2), NOT NULL DEFAULT 0)
   * data\_vencimento (DATE, NOT NULL)
   * taxa\_manutencao (DECIMAL(5,2), NOT NULL DEFAULT 0)
8. **conta\_investimento**
   * id\_conta\_investimento (PK, INT, AUTO\_INCREMENT)
   * id\_conta (FK, INT, REFERENCES conta(id\_conta), UNIQUE)
   * perfil\_risco (ENUM('BAIXO', 'MEDIO', 'ALTO'), NOT NULL)
   * valor\_minimo (DECIMAL(15,2), NOT NULL)
   * taxa\_rendimento\_base (DECIMAL(5,2), NOT NULL)
9. **transacao**
   * id\_transacao (PK, INT, AUTO\_INCREMENT)
   * id\_conta\_origem (FK, INT, REFERENCES conta(id\_conta))
   * id\_conta\_destino (FK, INT, REFERENCES conta(id\_conta)) - Para transferências
   * tipo\_transacao (ENUM('DEPOSITO', 'SAQUE', 'TRANSFERENCIA', 'TAXA', 'RENDIMENTO'), NOT NULL)
   * valor (DECIMAL(15,2), NOT NULL)
   * data\_hora (TIMESTAMP, NOT NULL DEFAULT CURRENT\_TIMESTAMP)
   * descricao (VARCHAR(100)) - Detalhe da transação
   * Índice em data\_hora
10. **auditoria**
    * id\_auditoria (PK, INT, AUTO\_INCREMENT)
    * id\_usuario (FK, INT, REFERENCES usuario(id\_usuario))
    * acao (VARCHAR(50), NOT NULL) - Ex.: "LOGIN", "ALTERACAO", "ENCERRAMENTO"
    * data\_hora (TIMESTAMP, NOT NULL DEFAULT CURRENT\_TIMESTAMP)
    * detalhes (TEXT) - JSON ou texto com valores antigos/novos
11. **relatorio**
    * id\_relatorio (PK, INT, AUTO\_INCREMENT)
    * id\_funcionario (FK, INT, REFERENCES funcionario(id\_funcionario))
    * tipo\_relatorio (VARCHAR(50), NOT NULL)
    * data\_geracao (TIMESTAMP, NOT NULL DEFAULT CURRENT\_TIMESTAMP)
    * conteudo (TEXT, NOT NULL) - Dados em formato JSON ou CSV

**Gatilhos (Triggers)**

1. Atualização de Saldo: Após cada transação (INSERT em transacao), atualizar o saldo da conta correspondente.

**sql**

DELIMITER $$

CREATE TRIGGER atualizar\_saldo AFTER INSERT ON transacao

FOR EACH ROW

BEGIN

IF NEW.tipo\_transacao = 'DEPOSITO' THEN

UPDATE conta SET saldo = saldo + NEW.valor WHERE id\_conta = NEW.id\_conta\_origem;

ELSEIF NEW.tipo\_transacao IN ('SAQUE', 'TAXA') THEN

UPDATE conta SET saldo = saldo - NEW.valor WHERE id\_conta = NEW.id\_conta\_origem;

ELSEIF NEW.tipo\_transacao = 'TRANSFERENCIA' THEN

UPDATE conta SET saldo = saldo - NEW.valor WHERE id\_conta = NEW.id\_conta\_origem;

UPDATE conta SET saldo = saldo + NEW.valor WHERE id\_conta = NEW.id\_conta\_destino;

END IF;

END $$

DELIMITER ;

1. Validação de Senha Forte: Antes de atualizar senha\_hash em usuario, verificar se a senha tem pelo menos 8 caracteres, 1 letra maiúscula, 1 número e 1 caractere especial.

**sql**

DELIMITER $$

CREATE TRIGGER validar\_senha BEFORE UPDATE ON usuario

FOR EACH ROW

BEGIN

IF NEW.senha\_hash REGEXP '^[0-9a-f]{32}$' THEN -- Assume MD5

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Senha deve ser atualizada via procedure com validação';

END IF;

END $$

DELIMITER ;

1. Limite de Depósito Diário: Bloquear depósitos acima de R$10.000 por dia por cliente.

sql

DELIMITER $$

CREATE TRIGGER limite\_deposito BEFORE INSERT ON transacao

FOR EACH ROW

BEGIN

DECLARE total\_dia DECIMAL(15,2);

SELECT SUM(valor) INTO total\_dia

FROM transacao

WHERE id\_conta\_origem = NEW.id\_conta\_origem

AND tipo\_transacao = 'DEPOSITO'

AND DATE(data\_hora) = DATE(NEW.data\_hora);

IF (total\_dia + NEW.valor) > 10000 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Limite diário de depósito excedido';

END IF;

END $$

DELIMITER ;

**Procedimentos Armazenados (Stored Procedures)**

1. **Gerar OTP: Cria um código temporário de 6 dígitos válido por 5 minutos.**

**sql**

DELIMITER $$

CREATE PROCEDURE gerar\_otp(IN id\_usuario INT)

BEGIN

DECLARE novo\_otp VARCHAR(6);

SET novo\_otp = LPAD(FLOOR(RAND() \* 1000000), 6, '0');

UPDATE usuario SET otp\_ativo = novo\_otp, otp\_expiracao = NOW() + INTERVAL 5 MINUTE

WHERE id\_usuario = id\_usuario;

SELECT novo\_otp;

END $$

DELIMITER ;

1. **Calcular Score de Crédito: Atualiza o score do cliente com base no histórico de transações.**

**sql**

DELIMITER $$

CREATE PROCEDURE calcular\_score\_credito(IN id\_cliente INT)

BEGIN

DECLARE total\_trans DECIMAL(15,2);

DECLARE media\_trans DECIMAL(15,2);

SELECT SUM(valor), AVG(valor) INTO total\_trans, media\_trans

FROM transacao t

JOIN conta c ON t.id\_conta\_origem = c.id\_conta

WHERE c.id\_cliente = id\_cliente AND t.tipo\_transacao IN ('DEPOSITO', 'SAQUE');

UPDATE cliente SET score\_credito = LEAST(100, (total\_trans / 1000) + (media\_trans / 100))

WHERE id\_cliente = id\_cliente;

END $$

DELIMITER ;

Visões (Views)

1. Resumo de Contas por Cliente:

sql

CREATE VIEW vw\_resumo\_contas AS

SELECT c.id\_cliente, u.nome, COUNT(co.id\_conta) AS total\_contas, SUM(co.saldo) AS saldo\_total

FROM cliente c

JOIN usuario u ON c.id\_usuario = u.id\_usuario

JOIN conta co ON c.id\_cliente = co.id\_cliente

GROUP BY c.id\_cliente, u.nome;

1. Movimentações Recentes:

sql

CREATE VIEW vw\_movimentacoes\_recentes AS

SELECT t.\*, c.numero\_conta, u.nome AS cliente

FROM transacao t

JOIN conta c ON t.id\_conta\_origem = c.id\_conta

JOIN cliente cl ON c.id\_cliente = cl.id\_cliente

JOIN usuario u ON cl.id\_usuario = u.id\_usuario

WHERE t.data\_hora >= NOW() - INTERVAL 90 DAY;

**+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**Estrutura do Projeto**

* **Pacotes:**
  + dao: DAOs para cada tabela com métodos complexos (ex.: findByScoreRange, getTransactionSummary).
  + model: Classes refletindo as tabelas com validações internas.
  + view: Interface gráfica livre (Swing, JavaFX, HTML/CSS/JS com framework web, etc.).
  + controller: Lógica de negócios com chamadas a procedures e gatilhos.
  + util: Conexão MySQL e manipulação de JSON/CSV para relatórios.
* **Requisitos Adicionais:**
  + Implementar índices para otimizar consultas em transacao (data\_hora) e conta (numero\_conta).
  + Usar transações SQL para operações críticas (ex.: transferências entre contas).
  + Exportar relatórios em pelo menos dois formatos (ex.: Excel e PDF).

**+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**Entrega e Avaliação**

* Entrega do Código: Código organizado em pacotes, com comentários detalhando gatilhos, procedures e visões.
* Documentação: Diagrama ER completo, explicação das regras de negócio no banco e instruções de configuração.
* Apresentação: Demonstração prática do sistema, destacando consultas complexas e relatórios.

**Critérios de Avaliação**

1. Complexidade do Banco (30%): Uso correto de gatilhos, procedures, visões e índices.
2. Integridade e Consistência (25%): Regras de negócio implementadas no banco (ex.: limites, validações).
3. Funcionalidade (25%): Todas as operações funcionando com persistência.
4. Interface e Usabilidade (20%): Interface funcional e intuitiva, independentemente da tecnologia escolhida.

**Bibliografia Básica**

1. DATE, C. J. *Introdução a Sistemas de Bancos de Dados*. 8. ed. São Paulo: Pearson Addison Wesley, 2004.
2. SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. *Sistemas de Banco de Dados*. 6. ed. São Paulo: Pearson, 2013.
3. DEITEL, Paul; DEITEL, Harvey. *Java: Como Programar*. 10. ed. São Paulo: Pearson, 2016.

**Bibliografia Complementar**

1. CORONEL, Carlos; MORRIS, Steven; ROB, Peter. *Database Systems: Design, Implementation, & Management*. 13th ed. Boston: Cengage Learning, 2019.
2. FOWLER, Martin. *UML Essencial: Um Breve Guia para a Linguagem Padrão de Modelagem de Objetos*. 3. ed. Porto Alegre: Bookman, 2005.
3. SOMMERVILLE, Ian. *Engenharia de Software*. 10. ed. São Paulo: Pearson, 2019.
4. ELMASRI, Ramez; NAVATHE, Shamkant. *Fundamentos de Sistemas de Banco de Dados*. 7. ed. São Paulo: Pearson, 2016.
5. BATES, Chris. *Desenvolvimento de Websites com MySQL e PHP*. 5. ed. São Paulo: Cengage Learning, 2017. (Útil como referência para alunos que optarem por front-end web).