|  |  |  |
| --- | --- | --- |
|  | **CENTRO UNIVERSITÁRIO DE PATOS DE MINAS – UNIPAM**  **BACHARELADO EM SISTEMAS DE INFORMAÇÃO**  **SISTEMAS DE BANCO DE DADOS – 4º PERÍODO**  **PROFESSOR RAFAEL MARINHO E SILVA** | |
| **ALUNO (A): Caio César Alves de Freitas** | |
| **VALOR: 4 PONTOS** | **DATA: 07/11/2025** |
| A lista de exercícios deverá ser resolvida manualmente e entregue em formato PDF pelo Portal UNIPAM até o dia 07/11/2025. | | |

**LISTA DE EXERCÍCIOS**

Com base nos conceitos e conteúdos sobre Banco de Dados estudados em aulas expositivas e na resolução de exercícios, utilize os exemplos e materiais das aulas práticas, bem como o livro Sistemas de Banco de Dados, 7ª edição (Capítulos 6 e 7), disponível na biblioteca do UNIPAM, para responder a todas as questões a seguir.

**Instruções gerais**

* Conecte-se ao banco plano\_saude fornecido no roteiro e execute os comandos solicitados.
* Para cada comando SQL solicitado anexe um print contendo (i) o comando executado e (ii) o resultado retornado pelo SGBD.
* **Entrega: arquivo PDF**.

**QUESTÕES TEÓRICAS**

|  |
| --- |
| Para cada pergunta abaixo responda consultando o livro.   1. O que é SQL?   **SQL é a linguagem padrão para Sistemas de Gerenciamento de Banco de Dados Relacionais, fornecendo comandos para estruturar o banco de dados (DDL) e comandos para consultar e manipular os dados armazenados (DML).**   1. Explique o conceito de cada categoria DDL, DML, DQL, DTL e DCL.  * **Linguagem de Definição de Dados (DDL): usada para definir o esquema e estruturar o banco de dados.** * **Linguagem de Manipulação de Dados (DML): usada para manipular os dados do banco.** * **Linguagem de Consulta de Dados (DQL): usada para, especificamente, consultar os dados armazenados, sendo parte da DML.** * **Linguagem de Controle de Dados (DCL): usada para gerenciar a segurança e as permissões de acesso do banco de dados, controlando quem pode ver e modificar os dados.** * **Linguagem de Transação de Dados (DTL): usada para gerenciar as transações do banco de dados, fazendo com que as mudanças sejam permanentes ou desfeitas.**  1. Explique o conceito de normalização: motivos, formas normais (1ª, 2ª, 3ª, BCNF) e como aplicá-las.   **A normalização é um processo usado para analisar esquemas de tabelas (relações) com base em suas dependências funcionais e chaves primárias, com o objetivo de decompor tabelas grandes em tabelas menores e bem estruturadas, que quando reunidas, formam os dados originais sem perdas.**  **O principal motivo é evitar a redundância dos dados e a distorção das informações, acontecendo devido a anomalias:**   * **Anomalia de inserção: ocorre quando não é possível adicionar um dado sem que outro dado também esteja presente.** * **Anomalia de Exclusão: ocorre quando a exclusão de um registro acarreta na perda não intencional de outras informações.** * **Anomalia de Modificação: ocorre quando a alteração de um dado exige a atualização de múltiplas linhas.**   **Para entender as formas normais, é necessário entender o conceito de dependência funcional.** Dependência Funcional é uma regra ou restrição que define um relacionamento entre atributos dentro de uma tabela. Dizemos que um Atributo B tem uma dependência funcional de um Atributo A se, para cada valor do Atributo A, existir *exatamente um único* valor correspondente do Atributo B (ou escreve-se Atributo A -> Atributo B). Isso significa que, se soubermos o valor de A, conseguimos descobrir o valor de B, por exemplo, se soubermos o CPF de uma pessoa, logo conseguimos descobrir o nome dela.  **Algumas formas normais são:**   * **Primeira Forma Normal: uma tabela está na primeira forma se todos os seus atributos forem atômicos. Isso significa que nenhum atributo pode conter múltiplos valores ou um conjunto de valores. Por exemplo, se uma tabela Funcionário tem um atributo Dependentes que armazena uma lista de nomes, violando a norma. Para corrigir cria-se uma tabela Dependentes e a relacionamos com Funcionário.** * **Segunda Forma Normal: uma tabela está na segunda forma se ela estiver na primeira forma e todo atributo não chave é totalmente dependente da chave primária, proibindo dependência parcial. Por exemplo, se uma tabela Inscrição tem a chave primária CPF\_Aluno, ID\_Curso e um atributo Nome\_Aluno, violando a norma. Para corrigir cria-se uma tabela Aluno com CPF\_Aluno e Nome\_Aluno e manter a Inscrição com a chave primária.** * **Terceira Forma Normal: uma tabela está na terceira forma se ela estiver na segunda forma e nenhum atributo não chave é transitivamente dependente da chave primária. Por exemplo, se uma tabela Funcionário com CPF, Nome, ID\_Departamento e Nome\_Departamento, temos que para conseguir o ID\_Departamento dependemos do CPF e para conseguir o Nome\_Departamento dependemos de ID\_Departamento, assim Nome\_Departamento depende transitivamente de CPF. Para corrigir cria-se uma tabela Departamento com ID\_Departamento e Nome\_Departamento e Funcionário com CPF, Nome e ID\_Departamento.** * **Forma Normal de Boyce-Codd (BCNF): uma tabela está na BCNF se ela estiver na terceira forma e, para cada dependência funcional que existir, o Atributo A for uma superchave. Ela é uma versão mais rigorosa da 3FN. Por exemplo, se uma tabela Matrícula tem os atributos CPF\_Aluno, ID\_Disciplina e Nome\_Professor, onde a chave primária é (CPF\_Aluno, ID\_Disciplina), mas também existe uma regra de que "cada professor ensina apenas uma disciplina" (o que cria a dependência Nome\_Professor -> ID\_Disciplina), a norma é violada. A violação ocorre porque Nome\_Professor determina um atributo (ID\_Disciplina), mas Nome\_Professor não é uma superchave. Para corrigir, decompomos em duas tabelas: Professor\_Disciplina (com Nome\_Professor e ID\_Disciplina) e Matricula\_Aluno (com CPF\_Aluno e Nome\_Professor).**   **Para normalizar um banco de dados é necessário:**   1. **Identificar as dependências funcionais.** 2. **Verificar se a tabela viola a primeira forma. Se sim, decompor.** 3. **Verificar se a tabela viola a segunda forma. Se sim, decompor.** 4. **Verificar se a tabela viola a terceira forma. Se sim, decompor.** 5. **Verificar se a tabela viola a BCNF. Se sim, decompor.** 6. Defina subclasses, superclasses e herança em modelos relacionais/OR-mapeamento.  * **Supercalsse: é um tipo de entidade que inclui um ou mais subgrupos distintos que são significativos para o modelo e precisam ser representados. As superclasses contêm todos os atributos e relacionamentos comuns as subclasses.** * **Subclasse: é um subconjunto de entidades de uma superclasse aos outros membros da supercalsse.** * **Herança: é o mecanismo que conecta a subclasse à superclasses, com a subclasse herdando todos os atributos da superclasse.**   **Para simular esses conceitos usando tabelas e chaves existem algumas opções:**   * **Criar uma tabela que representa as superclasse e tabelas separadas para cada subclasse. Na tabela da subclasse, a chave primária da tabela da superclasse é usada como chave estrangeira, conectando-as.** * **Criar tabelas para as subclasses com todos os atributos que seriam herdados nelas, sem uma tabela para a superclasse.** * **Criar uma tabela para a superclasse e nela colocar atributos que vão identificar a qual subclasse elas pertencem.**  1. Explique as alternativas de implementação “tabela por hierarquia”, “tabela por classe concreta” e “tabela por subtipo”. Cite vantagens e desvantagens de cada abordagem.   **Tabela por hierarquia:**  **Toda a hierarquia de classes (superclasse e todas as subclasses) é mapeada para uma única tabela no banco de dados. Essa tabela contém colunas para todos os atributos (comuns da superclasse e específicos de todas as subclasses). Uma coluna especial, chamada discriminadora, é usada para identificar qual classe aquela linha representa (ex: 'ENGENHEIRO' ou 'SECRETARIA').**  **Vantagens:**   * **Desempenho em Consultas Polimórficas: É a abordagem mais rápida para buscar todos os tipos de funcionários de uma vez. Uma consulta SELECT \* FROM FUNCIONARIO traz todo mundo, sem a necessidade de JOINs ou UNIONs.** * **Simplicidade: O esquema é simples de entender à primeira vista, sendo apenas uma tabela.**   **Desvantagens:**   * **Muitos NULLs: Esta é a principal desvantagem, com colunas de uma classe não possuindo valor (NULL). Isso desperdiça espaço e pode tornar os dados confusos.** * **Violação de Normalização: A tabela armazena atributos que não se aplicam a todas as linhas, o que pode ser visto como uma forma de violação de normalização.** * **Restrições de NOT NULL: Não é possível aplicar uma restrição NOT NULL em um atributo específico da subclasse, porque as linhas da subclasse terão que ter NULL nesse campo.** * **Manutenção: Adicionar um novo atributo a uma subclasse (ou uma nova subclasse) exige um ALTER TABLE na tabela inteira, o que pode ser uma operação pesada e arriscada em bancos de dados grandes.**   **Tabela por classe concreta (TPC):**  **Tabelas são criadas apenas para as classes concretas (geralmente as subclasses). A superclasse (se for abstrata) não ganha uma tabela própria. Cada tabela de subclasse armazena todos os atributos: os seus próprios (específicos) e os que ela herdou (comuns).**  **Vantagens:**   * **Desempenho em Consultas Específicas: Consultar apenas engenheiros é muito rápido (SELECT \* FROM ENGENHEIRO), pois não há JOINs.** * **Sem NULLs Desnecessários: Cada tabela armazena apenas os dados que lhe competem.**   **Desvantagens:**   * **Pior Desempenho Polimórfico: Consultar todos os funcionários é difícil. É necessário usar UNION em todas as tabelas (SELECT \* FROM ENGENHEIRO UNION SELECT \* FROM SECRETARIA), o que é muito ineficiente.** * **Redundância de Dados: Os atributos comuns (Nome, Salario) são fisicamente duplicados em todas as tabelas.** * **Problemas de Manutenção: Se for necessário adicionar um novo atributo comum à superclasse FUNCIONARIO (ex: DataContratacao), você terá que aplicar um ALTER TABLE em todas as tabelas de subclasses.** * **Integridade Referencial: É muito difícil criar uma chave estrangeira que aponte para "um funcionário" (de qualquer tipo). Se a tabela DEPARTAMENTO tem um Gerente\_CPF, para qual tabela ela deve apontar? Isso complica muito o modelo.**   **Tabela por subtipo:**  **Há uma tabela que representa a superclasse que contém todos os atributos compartilhados entre as entidades, e são criadas tabelas para todas as subclasses com os atributos específicos de cada uma e uma chave estrangeira referenciando a tabela da superclasse. Por exemplo, haveria as tabelas FUNCIONÁRIO, ENGENHEIRO e SECRETÁRIA, nas tabelas ENGENEHIRO e SECRETÁRIA teriam uma chave estrangeira referenciando a tabela FUNCIONÁRIO.**  **Vantagens:**   * **Melhor Normalização: Esta é a solução academicamente mais correta e normalizada (3FN). Não há redundância e não há NULLs desnecessários.** * **Clareza e Flexibilidade: O esquema do banco de dados reflete perfeitamente a hierarquia de classes. Adicionar uma nova subclasse é fácil: basta criar uma nova tabela sem afetar as existentes.** * **Integridade Referencial: Qualquer tabela pode ter uma chave estrangeira apontando para FUNCIONARIO.CPF e funcionará para todos os tipos.** * **Restrições NOT NULL: É possível colocar NOT NULL em atributos específicos em suas respectivas tabelas.**   **Desvantagens:**   * **Desempenho com JOINs: Qualquer consulta que precise dos dados completos de uma subclasse sempre exigirá um JOIN entre FUNCIONARIO e a subclasse.** * **Complexidade em Consultas Polimórficas: Buscar todos os funcionários com seus dados específicos pode exigir múltiplos LEFT JOINs (um para cada subclasse), o que pode ser complexo de escrever e executar.**  1. Explique especialização e generalização. Quando usar cada uma no ciclo de projeto de banco de dados?   **Especialização:**  **É o processo de identificar vários subgrupos a partir de uma entidade genérica, com os subgrupos possuindo características (atributos) ou relacionamentos específicos. É usado quando existe uma entidade genérica definida e se tem a necessidade adicionar mais detalhes ou tratar exceções de um subgrupo específico, sendo ideal para refinamento do banco de dados.**  **Generalização:**  **É o processo de identificar uma entidade genérica a partir de várias entidades que possuem o mesmo tipo de atributo, compartilhando de atributos e relacionamentos em comum. É usado quando se começa a identificar as entidades de um sistema e percebe-se que as várias entidades podem pertencer a um único grupo devido à semelhança dos atributos e relacionamentos.**   1. Fundamentos de consultas JOIN: explique os conceitos de cada tipo de consulta, INNER JOIN, LEFT JOIN e RIGHT JOIN. Explique o que cada um retorna e quando usar.   **Inner Join:**  **É um comando que retorna os registros com base em uma comparação de colunas de uma ou mais tabelas com valores que são correspondentes. Ao buscar as informações, ele analisa a condição de junção e retorna apenas os registros que possuem uma correspondência exata em ambas as tabelas. Por exemplo, considerando duas tabelas:**  **FUNCIONÁRIO (**  **ID\_Funcionario (chave primária),**  **Nome,**  **CPF**  **);**  **MÉDICO (**  **ID\_Funcionario (chave estrangeira),**  **CRM**  **);**  **Ao fazer um Inner Join entre elas para retornar o nome, CPF e CRM dos médicos, é comparado as colunas ID\_Funcionario das duas tabelas para listar os médicos que existem, se houver um funcionário em que o seu ID\_Funcionario não corresponda com o ID\_Funcionario de um médico, ele não será listado, pois não é um médico.**  **Por essa natureza restritiva, ele é usado exatamente quando se deseja obter um conjunto de dados coeso e completo, onde apenas as informações que se relacionam mutuamente são relevantes, como ao listar clientes e os pedidos que eles efetivamente fizeram, ignorando clientes que nunca compraram ou pedidos sem cliente.**  **Left Join:**  **É um comando que “coloca” duas ou mais tabelas lado a lado e busca os dados das tabelas a esquerda combinando com os dados existentes das tabelas a direita. Ao buscar as informações, ele analisa a condição de junção e retorna todos os registros da tabela da esquerda (a primeira tabela da consulta) e os registros correspondentes da tabela da direita. Se não houver uma correspondência para um registro da esquerda, as colunas da tabela da direita serão retornadas como NULL (vazias). Por exemplo considerando as tabelas:**  **FUNCIONÁRIO (**  **ID\_Funcionario (chave primária),**  **Nome,**  **CPF**  **);**  **MÉDICO (**  **ID\_Funcionario (chave estrangeira),**  **CRM**  **);**  **Ao fazer um Left Join entre elas (com FUNCIONÁRIO sendo a tabela da esquerda) para retornar o nome, CPF e CRM de todos os funcionários, é comparado as colunas ID\_Funcionario. Se houver um funcionário em que o seu ID\_Funcionario não corresponda com o ID\_Funcionario de um médico, ele ainda assim será listado, mas a coluna CRM ficará como NULL.**  **Por essa natureza focada na tabela da esquerda, ele é usado exatamente quando se deseja obter um conjunto de dados completo da tabela principal, mas ainda assim ver as informações correspondentes da segunda tabela, se elas existirem, como ao listar todos os clientes, independentemente de terem feito pedidos ou não, e mostrando os pedidos apenas para aqueles que compraram.**  **Right Join:**  **É um comando que “coloca” duas ou mais tabelas lado a lado e busca os dados das tabelas a direita combinando com os dados existentes das tabelas a esquerda. Ao buscar as informações, ele analisa a condição de junção e retorna todos os registros da tabela da direita (a segunda tabela da consulta) e os registros correspondentes da tabela da esquerda. Se não houver uma correspondência para um registro da direita, as colunas da tabela da esquerda serão retornadas como NULL (vazias). Por exemplo considerando as tabelas:**  **FUNCIONÁRIO (**  **ID\_Funcionario (chave primária),**  **Nome,**  **CPF**  **);**  **MÉDICO (**  **ID\_Funcionario (chave estrangeira),**  **CRM**  **);**  **Ao fazer um Right Join entre elas (com MÉDICO sendo a tabela da direita) para retornar o Nome, CPF e CRM de todos os médicos, é comparado as colunas ID\_Funcionario. Se houver um médico em que o seu ID\_Funcionario não corresponda com o ID\_Funcionario de um funcionário (o que seria incomum, mas serve para o exemplo), ele ainda assim será listado, mas as colunas Nome e CPF ficarão como NULL.**  **Por essa natureza focada na tabela da direita, ele é usado exatamente quando se deseja obter um conjunto de dados completo da tabela da direita, mas ainda assim ver as informações correspondentes da tabela da esquerda, se elas existirem, como ao listar todos os pedidos, independentemente de terem clientes ou não, e mostrando os clientes apenas para aqueles pedidos que possuem um cliente.** |

**QUESTÕES PRÁTICAS**

|  |
| --- |
| 1. Liste todos os pacientes (nome e data\_cadastro)      1. Liste todos os médicos (nome, crm, especialidade)      1. Criar nova tabela endereco e migrar os dados:  * Criar tabela endereco com colunas atômicas (id, logradouro, numero, complemento, bairro, cidade, uf, cep).      * Popular endereco com os dados existentes (faça uma transformação simplificada se o formato atual for Rua X, 100).      * Alterar pessoa removendo colunas de endereço e adicionando endereco\_id FK.      * Atualizar pessoa.endereco\_id para referenciar [endereco.id](http://endereco.id).      * Mostrar SELECT p.nome, e.logradouro, e.numero FROM pessoa p JOIN endereco e ON p.endereco\_id = e.id; com print.      1. LEFT JOIN vs RIGHT JOIN: escreva e execute duas queries que relacionem paciente e consulta com LEFT JOIN e RIGHT JOIN (na sequência); capture os resultados e explique a diferença entre eles, indicando quais registros aparecem em cada resultado e por quê.       **A diferença entre as querys foi em qual tabela é considerada a “principal” na execução. Na query com LEFT JOIN a tabela pessoa é considerada a principal, então o comando retorna os dados de todas as pessoas que existem juntamente com os dados das consultas relacionadas a elas se esses dados existirem, podendo acontecer de aparecer dados NULL por existir pessoas que não fizeram consultas. Na query com RIGHT JOIN a tabela consulta é considerada a principal, então o comando retorna os dados de todas as consultas que existem juntamente com os dados das pessoas relacionadas a elas se esses dados existirem, porém, como para ter um registro na tabela consulta é necessário passar o paciente\_id, é impossível haver uma consulta sem um paciente, logo não é retornado nenhuma consulta que não tenha um paciente (dados em NULL).**   1. INNER JOIN vs RIGHT JOIN (igualdade de resultados): escreva e execute queries INNER JOIN e RIGHT JOIN para medico + consulta e paciente + consulta, explique em quais circunstâncias os resultados serão idênticos e por quê.     **Executando essas duas querys, uma utilizando o INNER JOIN e outra o RIGHT JOIN, os resultados são os mesmos, isso devido a estrutura das tabelas. No comando com INNER JOIN é retornado os registros das tabelas medico e consulta onde o valor do atributo pessoa\_id de médico é igual ao valor do atributo medico\_id de consulta, então se houvesse um médico sem consulta ou uma consulta sem médico (impossível de acontecer, pois para criar um registro em consulta é necessário ter um médico) eles não apareceriam. No comando com RIGHT JOIN a tabela consulta é considerada a principal e é retornado todas as consultas juntamente com os dados do médico, mas como é impossível existir um registro de consulta sem um medico\_id, o resultado acaba sendo o mesmo, se por acaso existisse uma consulta sem medico\_id, ela iria aparecer no resultado com os dados do médico como NULL.**      **Executando essas duas querys, uma utilizando o INNER JOIN e outra o RIGHT JOIN, os resultados são os mesmos, isso devido a estrutura das tabelas. No comando com INNER JOIN é retornado os registros das tabelas paciente e consulta onde o valor do atributo pessoa\_id de paciente é igual ao valor do atributo paciente\_id de consulta, então se houvesse um paciente sem consulta ou uma consulta sem paciente (impossível de acontecer, pois para criar um registro em consulta é necessário ter um paciente) eles não apareceriam. No comando com RIGHT JOIN a tabela consulta é considerada a principal e é retornado todas as consultas juntamente com os dados do paciente, mas como é impossível existir um registro de consulta sem um paciente\_id, o resultado acaba sendo o mesmo, se por acaso existisse uma consulta sem paciente\_id, ela iria aparecer no resultado com os dados do paciente como NULL.** |

**Boa Atividade!!!**