PRATICANDO

BANCO DE DADOS

Apresentação

Praticar é fundamental para o seu aprendizado. Sentir-se desafiado, lidar com a frustração e aplicar conceitos são essenciais para fixar conhecimentos. No ambiente Praticando, você terá a oportunidade de enfrentar desafios específicos e estudos de caso, criados para ampliar suas competências e para a aplicação prática dos conhecimentos adquiridos.

Objetivo

Ampliar competências e consolidar conhecimentos através de desafios específicos e estudos de caso práticos.

Otimização de Consultas SQL em PostgreSQL

Caso Prático

João, um programador de uma empresa de tecnologia, foi designado para otimizar o desempenho de uma consulta SQL que está impactando negativamente a performance do banco de dados PostgreSQL utilizado pela empresa. A consulta em questão realiza operações complexas em várias tabelas, incluindo junções e funções de agregação, e é executada frequentemente, o que resulta em uma sobrecarga significativa no sistema. O problema está ocorrendo no ambiente de produção da empresa, onde as consultas lentas estão afetando diretamente o tempo de resposta das aplicações e, consequentemente, a experiência do usuário. O desempenho da consulta começou a degradar à medida que o volume de dados aumentou significativamente, algo que não foi considerado no design inicial da consulta. João precisa identificar os gargalos e propor uma solução que melhore a eficiência dessa consulta.

Diante dessa situação, analise as possíveis estratégias que João poderia adotar para otimizar a consulta SQL no PostgreSQL. Considere técnicas de indexação, reescrita da consulta, e uso de estruturas avançadas de dados ou funções do PostgreSQL que poderiam melhorar o desempenho. Justifique sua resposta com base no conteúdo estudado sobre consultas SQL e otimização de desempenho em sistemas de banco de dados.

Chave de resposta

João pode adotar várias estratégias para otimizar a consulta SQL. Primeiramente, ele deve considerar a criação de índices nas colunas utilizadas em junções e nas cláusulas WHERE, o que pode acelerar a recuperação dos dados. Além disso, João pode reescrever a consulta para reduzir a complexidade, possivelmente dividindo-a em subconsultas menores para melhorar a legibilidade e a eficiência. Outra técnica seria a análise da distribuição de dados e a aplicação de particionamento em tabelas grandes para melhorar a eficiência das operações de leitura. João também deve revisar a necessidade de todas as operações realizadas na consulta, eliminando qualquer processamento desnecessário. A combinação

dessas estratégias deve resultar em uma consulta otimizada, reduzindo o tempo de execução e melhorando a performance geral do sistema.

Para saber mais sobre esse conteúdo, acesse:

Tema: Consultas em uma Tabela no PostgreSQL

Tema: Sistema de Banco de Dados

Tema: Projeto de Banco de Dados: Modelagem Lógica e Física

Sistema de Banco de Dados

Desafio 1

Você é um administrador de banco de dados recém-contratado por uma grande corporação e sua primeira tarefa é identificar o melhor sistema de banco de dados para as novas necessidades da empresa, que envolvem o gerenciamento de grandes volumes de dados não estruturados e a necessidade de escalabilidade. Considerando a evolução dos sistemas de banco de dados e suas características, qual tipo de SGBD seria mais apropriado para atender a essas necessidades?

A

Sistema de Banco de Dados Relacional com suporte SQL.

B

Sistema de Banco de Dados Hierárquico com suporte a grandes volumes de dados.

C

Sistema de Banco de Dados NoSQL, especializado em dados não estruturados.

D

Sistema de Arquivos Tradicional com capacidade de escalabilidade.

E

Sistema de Banco de Dados Navegacional baseado em estruturas de grafos.

A alternativa C está correta.

A) Sistema de Banco de Dados Relacional com suporte SQL: Incorreta. Embora os SGBDs relacionais sejam amplamente utilizados devido à sua estrutura baseada em tabelas e uso da linguagem SQL, eles não são a melhor escolha para dados não estruturados e grandes volumes de dados que exigem escalabilidade. Esses sistemas são ideais para dados estruturados e normalizados, mas podem ter limitações significativas ao lidar com as necessidades mencionadas.

B) Sistema de Banco de Dados Hierárquico com suporte a grandes volumes de dados: Parcialmente incorreta. Os bancos de dados hierárquicos eram comuns nos primeiros dias da computação, especialmente para dados com uma estrutura rígida e clara. No entanto, eles não são adequados para lidar com dados não estruturados ou para escalar de maneira eficiente em um ambiente moderno de grandes volumes de dados.

C) Sistema de Banco de Dados NoSQL, especializado em dados não estruturados: Correta. Sistemas de banco de dados NoSQL são projetados especificamente para gerenciar grandes volumes de dados não

estruturados, oferecendo flexibilidade e escalabilidade que os SGBDs tradicionais não conseguem fornecer. Eles são ideais para aplicações modernas que lidam com Big Data, Internet das Coisas (IoT) e outras situações onde a estrutura dos dados não é rígida.

D) Sistema de Arquivos Tradicional com capacidade de escalabilidade: Incorreta. Sistemas de arquivos tradicionais são limitados na maneira como gerenciam dados, não oferecendo as funcionalidades avançadas que SGBDs modernos proporcionam, especialmente em termos de escalabilidade e gerenciamento de grandes volumes de dados não estruturados.

E) Sistema de Banco de Dados Navegacional baseado em estruturas de grafos: Parcialmente incorreta. Enquanto bancos de dados navegacionais, especialmente aqueles baseados em grafos, podem ser úteis para certos tipos de dados e relações, eles não são a melhor escolha para gerenciar grandes volumes de dados não estruturados com eficiência e escalabilidade.

Para saber mais sobre esse conteúdo, acesse:

Módulo 1

OUTROS MODELOS DE SGBDS

“Entretanto, existem aplicações em sistemas de informação que requerem muito mais recursos de armazenamento e manipulação de dados do que as tabelas do modelo relacional, em especial aplicações Web e de cunho científico que processam grandes quantidades de dados em formatos diversos, com as atuais tendências como Big Data, Internet of Things e Data Science. Assim, vários modelos de banco de dados não relacionais vêm surgindo no mercado, sendo denominados de NoSQL...”

Desafio 2

Como administrador de um banco de dados, você foi informado sobre a necessidade de realizar alterações no esquema lógico de um banco de dados sem afetar o esquema externo ou os dados físicos armazenados. Com base na arquitetura dos sistemas de banco de dados, qual das seguintes características seria fundamental para realizar essa tarefa?

A

Independência física dos dados.

B

Independência de plataforma de hardware.

C

Independência lógica dos dados.

D

Compartilhamento de dados entre múltiplos usuários.

E

Natureza autocontida dos dados.

A alternativa C está correta.

A) Independência física dos dados: Parcialmente incorreta. A independência física dos dados se refere à capacidade de modificar a estrutura física do banco de dados (como a reorganização dos arquivos) sem alterar a estrutura lógica do banco de dados. Embora importante, essa característica não aborda diretamente a alteração do esquema lógico sem afetar o esquema externo.

B) Independência de plataforma de hardware: Incorreta. Essa característica permitiria que o banco de dados operasse em diferentes plataformas de hardware, mas não está relacionada à capacidade de alterar o esquema lógico sem impacto nas visões dos usuários finais ou na estrutura física.

C) Independência lógica dos dados: Correta. A independência lógica dos dados permite que o esquema lógico do banco de dados seja alterado sem afetar o esquema externo, que é a visão dos usuários finais, ou a estrutura física dos dados. Essa característica é fundamental para garantir flexibilidade na manutenção e evolução do banco de dados sem prejudicar as aplicações que dependem dele.

D) Compartilhamento de dados entre múltiplos usuários: Incorreta. Essa característica está relacionada à capacidade de múltiplos usuários acessarem e manipularem os dados simultaneamente, mas não à alteração do esquema lógico sem afetar outras camadas.

E) Natureza autocontida dos dados: Parcialmente incorreta. A natureza autocontida dos dados refere-se ao fato de que os dados contêm tanto as informações quanto a descrição dessas informações, mas não aborda diretamente a independência lógica do esquema.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

DIFERENÇAS ENTRE SISTEMA DE ARQUIVOS E SISTEMA DE BANCO DE DADOS

“Independência lógica de dados consiste na capacidade de se alterar o esquema conceitual lógico, por exemplo, acrescentando um item de dado, sem alterar o esquema conceitual externo, isto é, as visões externas dos usuários através dos programas de aplicação.”

Desafio 3

Você é responsável por gerenciar a segurança e a integridade dos dados em um sistema de banco de dados relacional utilizado em uma empresa multinacional. Durante uma auditoria, foi solicitado que você explicasse como o banco de dados garante que todas as transações sejam completadas com sucesso ou revertidas completamente em caso de falha. Qual princípio fundamental dos sistemas de banco de dados você descreveria?

A

Durabilidade dos dados.

B

Atomicidade das transações.

C

Isolamento das transações.

D

Consistência dos dados.

E

Controle de redundância de dados.

A alternativa B está correta.

A) Durabilidade dos dados: Parcialmente incorreta. A durabilidade garante que, uma vez que uma transação é concluída, suas alterações permanecem no banco de dados mesmo após uma falha de sistema. Embora importante, esse conceito não aborda a questão de garantir que uma transação seja completada totalmente ou revertida em caso de falha.

B) Atomicidade das transações: Correta. A atomicidade é um dos princípios fundamentais da propriedade ACID (Atomicidade, Consistência, Isolamento, Durabilidade) em sistemas de banco de dados, garantindo que cada transação seja tratada como uma unidade indivisível. Isso significa que, se qualquer parte da transação falhar, toda a transação é revertida, garantindo que o banco de dados nunca permaneça em um estado parcial ou inconsistente.

C) Isolamento das transações: Parcialmente incorreta. O isolamento garante que as transações sejam executadas de forma independente, sem interferências umas das outras. Embora importante para a integridade dos dados, não aborda diretamente a garantia de completude ou reversão das transações.

D) Consistência dos dados: Parcialmente incorreta. A consistência garante que uma transação leve o banco de dados de um estado válido para outro estado válido. No entanto, a consistência sozinha não garante a completude ou reversão de uma transação em caso de falha.

E) Controle de redundância de dados: Incorreta. Esse conceito está relacionado à minimização da duplicação de dados no banco de dados e não aborda o comportamento das transações.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

VANTAGENS E DESVANTAGENS DA ABORDAGEM DE BANCO DE DADOS

“Atomicidade (Atomicity) Cada transação é tratada como uma unidade composta de uma sequência de operações, de modo que deve executar completamente com sucesso ou falhar completamente.”

Desafio 4

Você está trabalhando como analista de banco de dados em uma empresa de comércio eletrônico que processa um grande volume de transações diariamente. Um dos desafios que você enfrenta é garantir que todas as transações sejam consistentes, mesmo quando ocorrem falhas durante o processamento. Qual propriedade do banco de dados garante que, após a conclusão de uma transação, todas as mudanças feitas pelo processo sejam permanentemente gravadas, independentemente de falhas subsequentes?

A

Atomicidade.

B

Consistência.

C

Durabilidade.

D

Isolamento.

E

Independência lógica.

A alternativa C está correta.

A) Atomicidade: Parcialmente incorreta. A atomicidade garante que uma transação seja executada completamente ou não seja executada de forma alguma, mas não se refere à permanência das mudanças após a conclusão da transação, que é o foco da durabilidade.

B) Consistência: Parcialmente incorreta. A consistência assegura que uma transação leve o banco de dados de um estado válido para outro, mas não garante a permanência das mudanças feitas por uma transação após sua conclusão.

C) Durabilidade: Correta. A durabilidade é uma propriedade fundamental dos sistemas de banco de dados, garantindo que, uma vez que uma transação é confirmada (committed), todas as alterações feitas são permanentemente gravadas no banco de dados, mesmo que ocorram falhas subsequentes. Isso assegura que as mudanças persistam e estejam disponíveis para futuras operações.

D) Isolamento: Parcialmente incorreta. O isolamento assegura que as transações concorrentes sejam executadas de forma que não interfiram entre si, mas não está relacionado à permanência das mudanças feitas por uma transação após sua conclusão.

E) Independência lógica: Incorreta. A independência lógica refere-se à capacidade de alterar o esquema lógico do banco de dados sem afetar as visões externas ou os dados físicos, mas não está relacionada à garantia de permanência das mudanças feitas por uma transação.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

VANTAGENS E DESVANTAGENS DA ABORDAGEM DE BANCO DE DADOS

"Durabilidade (Durability): Uma vez que a transação é aceita (committed), o que significa que seu resultado foi gravado em memória não volátil, esse resultado permanecerá válido mesmo em caso de falhas do sistema.

Projeto de Banco de Dados Modelagem Conceitual

Desafio 1

Você foi recentemente contratado por uma empresa de tecnologia para trabalhar como analista de dados. Durante um projeto, sua equipe precisa modelar um banco de dados que atenda aos requisitos de uma escola de treinamentos. Durante a modelagem, um dos seus colegas sugere representar os relacionamentos entre cursos e alunos usando um diagrama de entidade e relacionamento (DER). No entanto, surgem dúvidas sobre a maneira correta de representar esses relacionamentos.

Qual a forma correta de representar o relacionamento entre cursos e alunos no diagrama de entidade e relacionamento?

A

Utilizando atributos adicionais dentro da entidade curso.

B

Criando uma entidade associativa que capture as inscrições dos alunos nos cursos.

C

Ligando diretamente a entidade aluno à entidade curso sem intermediários.

D

Criando uma única entidade para representar tanto cursos quanto alunos.

E

Utilizando um atributo multivalorado para representar todos os cursos em cada entidade de aluno.

A alternativa B está correta.

A) Incorreta. Utilizar atributos adicionais dentro da entidade curso para representar as inscrições dos alunos não é a abordagem correta, pois isso implicaria em uma má estruturação dos dados. A relação entre alunos e cursos é do tipo muitos-para-muitos, o que requer a criação de uma entidade associativa.

B) Correta. A melhor forma de representar um relacionamento muitos-para-muitos entre cursos e alunos é criar uma entidade associativa. Essa entidade associativa pode capturar detalhes importantes sobre a inscrição, como a data em que o aluno se inscreveu no curso. Isso permite uma modelagem precisa e evita redundâncias de dados.

C) Incorreta. Ligar diretamente a entidade aluno à entidade curso não capturaria adequadamente o relacionamento muitos-para-muitos entre essas entidades. Esse método não permitiria registrar informações adicionais sobre a relação, como a data de inscrição.

D) Incorreta. Criar uma única entidade para representar tanto cursos quanto alunos é uma abordagem incorreta, pois cada uma dessas entidades possui características e atributos distintos. Misturá-las em uma única entidade violaria os princípios da modelagem de dados.

E) Incorreta. Utilizar um atributo multivalorado para representar todos os cursos dentro de uma entidade aluno seria uma má prática de modelagem, pois dificultaria a consulta e a manutenção dos dados. A entidade associativa é a abordagem correta para este caso.

Para saber mais sobre esse conteúdo, acesse:

Módulo 1

PROJETO CONCEITUAL

“O projeto conceitual usa um diagrama gráfico, conhecido por Diagrama de Entidade e Relacionamento (DER), que possui três elementos essenciais: entidades, relacionamentos e atributos. [...] A partir desse diagrama, é possível concluir que o modelo possui duas entidades (CLIENTE e CURSO) cuja função é armazenar os dados dos clientes e dos cursos da escola. Além disso, essas entidades possuem uma relação entre si, de maneira que um cliente pode fazer inscrição em um ou mais cursos.”

Desafio 2

Você foi contratado para desenvolver um sistema de banco de dados para uma universidade. Um dos requisitos é que o sistema permita a criação de turmas associadas a disciplinas, com um controle preciso sobre o número de vagas disponíveis em cada turma. Como essa funcionalidade deve ser modelada no DER?

A

Criar uma entidade "Turma" separada, que se relaciona diretamente com "Disciplina", e incluir atributos como código da turma e número de vagas.

B

Representar "Turma" como um atributo multivalorado na entidade "Disciplina", especificando o número de vagas.

C

Utilizar um relacionamento ternário entre "Aluno", "Disciplina" e "Turma" para capturar as informações necessárias.

D

Adicionar as turmas como sub-entidades de "Disciplina", facilitando o controle sobre vagas.

E

Registrar "Turma" como um atributo composto em "Disciplina", detalhando as especificações de cada turma.

A alternativa A está correta.

A) Correta. Criar uma entidade separada para "Turma" é a abordagem mais correta, pois permite capturar todos os detalhes específicos de cada turma, como o número de vagas e o código da turma, sem sobrecarregar a entidade "Disciplina". Isso oferece flexibilidade e precisão na gestão das informações.

B) Incorreta. Modelar "Turma" como um atributo multivalorado na entidade "Disciplina" não é apropriado, pois dificultaria o rastreamento e a consulta de informações específicas sobre cada turma. Essa abordagem limita a capacidade do banco de dados de lidar com informações detalhadas e distintas sobre turmas.

C) Incorreta. Um relacionamento ternário entre "Aluno", "Disciplina" e "Turma" complicaria desnecessariamente o modelo de dados. A complexidade adicional não é justificada pela necessidade de capturar informações básicas como o número de vagas.

D) Incorreta. Adicionar turmas como sub-entidades de "Disciplina" não é a abordagem mais limpa. Isso poderia dificultar a manutenção e a expansão do banco de dados à medida que mais turmas e disciplinas são adicionadas.

E) Incorreta. Registrar "Turma" como um atributo composto em "Disciplina" não permitiria o detalhamento necessário para o controle eficaz das vagas, complicando a manipulação dos dados e tornando as consultas mais difíceis.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

ENTIDADE

“De acordo com Heuser (2009), a entidade corresponde a uma representação do conjunto de objetos da realidade modelada sobre os quais se deseja manter informações no banco de dados.”

Desafio 3

Você está trabalhando no desenvolvimento de um sistema de banco de dados para uma instituição acadêmica, onde é necessário gerenciar as orientações de projetos realizadas por professores com seus alunos. Cada orientação de projeto envolve um professor, um aluno e um projeto específico.

Como você modelaria essa situação em um Diagrama de Entidade e Relacionamento (DER) para capturar corretamente todas as associações entre essas entidades?

A

Criar uma entidade "Projeto" que se relaciona diretamente com "Professor" e "Aluno" de maneira separada, sem um vínculo direto entre os três.

B

Utilizar uma entidade "Orientação" para conectar "Professor", "Aluno" e "Projeto", criando uma relação ternária entre essas entidades.

C

Modelar "Aluno" como uma sub-entidade de "Professor", vinculando "Projeto" apenas ao "Aluno" através de uma relação binária.

D

Construir um relacionamento binário entre "Professor" e "Projeto", e outro entre "Aluno" e "Projeto", omitindo a relação direta entre "Professor" e "Aluno".

E

Representar "Orientação" como um atributo composto dentro da entidade "Projeto", para registrar os professores e alunos envolvidos.

A alternativa B está correta.

A) Incorreta. Criar uma entidade "Projeto" que se relaciona separadamente com "Professor" e "Aluno" não capturaria adequadamente a complexidade da relação entre os três elementos. Cada orientação envolve simultaneamente um professor, um aluno e um projeto, e esses elementos precisam estar diretamente conectados.

B) Correta. A criação de uma entidade "Orientação" que conecta "Professor", "Aluno" e "Projeto" em uma relação ternária é a melhor forma de modelar essa situação. Isso permite que o sistema capture corretamente cada instância onde um professor orienta um aluno em um projeto específico, preservando a integridade e a clareza do modelo de dados.

C) Incorreta. Modelar "Aluno" como uma sub-entidade de "Professor" criaria uma hierarquia que não reflete a realidade das relações. Alunos e professores são entidades independentes que se associam no contexto de um projeto, mas não devem ser subordinados um ao outro no modelo de dados.

D) Incorreta. Construir relacionamentos binários separadamente entre "Professor" e "Projeto" e entre "Aluno" e "Projeto" falha em capturar a interdependência dessas entidades dentro da orientação de um projeto. Isso poderia levar a dados inconsistentes e difíceis de gerenciar.

E) Incorreta. Representar "Orientação" como um atributo composto dentro da entidade "Projeto" simplificaria demais o modelo, deixando de registrar a relação dinâmica e única entre "Professor", "Aluno" e "Projeto" em cada orientação.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

RELACIONAMENTO TERNÁRIO

“Vamos modelar orientações de alunos em projetos, realizadas por docentes. Há três tipos de informações: projeto, aluno e docente. Estamos, portanto, diante de um relacionamento ternário. [...] Cada ocorrência do relacionamento ORIENTAÇÃO vincula três ocorrências de entidade: um projeto, um aluno a ser orientado e um docente orientador.”

Desafio 4

Você está desenvolvendo um sistema de banco de dados para uma universidade que precisa gerenciar diferentes tipos de funcionários, como professores e técnicos administrativos. Esses funcionários compartilham alguns atributos, como nome e telefone, mas cada tipo tem características específicas, como a graduação dos professores e a área de atuação dos técnicos. Como você deve modelar essa situação em um Diagrama de Entidade e Relacionamento (DER)?

A

Criar entidades separadas para "Professor" e "Técnico", sem qualquer conexão, para simplificar o modelo.

B

Modelar uma entidade única "Funcionário" sem diferenciar entre professores e técnicos, armazenando todos os atributos em uma tabela.

C

Utilizar uma entidade "Funcionário" genérica e criar sub-entidades "Professor" e "Técnico", cada uma com seus atributos específicos, usando o mecanismo de especialização/generalização.

D

Adicionar todos os atributos de professores e técnicos na entidade "Funcionário", marcando os não aplicáveis como nulos, para evitar a criação de sub-entidades.

E

Criar uma entidade "Funcionário" que se relaciona diretamente com outras entidades para armazenar as diferenças entre professores e técnicos.

A alternativa C está correta.

A) Criar entidades completamente separadas para "Professor" e "Técnico" pode parecer simplificar o modelo, mas falha em reconhecer que ambos são tipos de funcionários que compartilham características comuns. Isso levaria a duplicação de dados e a um modelo menos eficiente.

B) Modelar uma única entidade "Funcionário" sem diferenciar entre os tipos específicos não permite capturar as particularidades dos professores e técnicos. Isso resultaria em um modelo de dados que não reflete a realidade, tornando mais difícil o gerenciamento de informações específicas a cada tipo de funcionário.

C) Correta. O uso de uma entidade genérica "Funcionário" com sub-entidades "Professor" e "Técnico" é a maneira correta de modelar essa situação. Isso permite que os atributos comuns sejam armazenados na entidade "Funcionário", enquanto as características específicas de cada tipo de funcionário são tratadas nas sub-entidades, mantendo a integridade e a clareza do modelo de dados.

D) Adicionar todos os atributos de professores e técnicos na entidade "Funcionário" e marcar os não aplicáveis como nulos é uma má prática, pois aumenta a complexidade da tabela e pode levar a problemas de manutenção e desempenho. A especialização/generalização é uma abordagem mais limpa e eficiente.

E) Criar uma entidade "Funcionário" que se relaciona diretamente com outras entidades para armazenar as diferenças não é necessário e complicaria desnecessariamente o modelo. A especialização/generalização oferece uma solução mais adequada e alinhada com os princípios de modelagem de dados.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

ESPECIALIZAÇÃO/ GENERALIZAÇÃO

“O mecanismo de especialização/generalização é representado por um triângulo, com a entidade mais genérica localizada na parte superior e a(s) entidade(s) especializada(s) na parte inferior. [...] Todo funcionário possui um código único, além de nome e pelo menos um telefone. Há duas entidades especializadas: DOCENTE e ANALISTA. A entidade DOCENTE possui um atributo obrigatório GRADUAÇÃO.”

Projeto de Banco de Dados Modelagem Lógica e Física

Desafio 1

Você foi contratado para revisar o design de um banco de dados existente para uma empresa de tecnologia. Durante sua análise, você percebe que o banco de dados possui diversas tabelas com colunas repetitivas e dados redundantes. Qual seria o processo correto para eliminar essas redundâncias e melhorar o desempenho?

A

Aplicar a desnormalização, que visa adicionar redundâncias controladas para otimizar consultas.

B

Implementar o processo de normalização até a 2FN, o que elimina todas as dependências funcionais transitivas.

C

Adotar a técnica de normalização até a 3FN, o que assegura que não haja dependências parciais ou transitivas nas tabelas.

D

Definir chaves estrangeiras em todas as tabelas, garantindo a integridade referencial entre elas.

E

Criar índices em todas as colunas, otimizando o desempenho das consultas e eliminando redundâncias.

A alternativa C está correta.

A) Está incorreta, pois a desnormalização, embora útil em alguns casos para melhorar o desempenho de consultas, não é adequada para eliminar redundâncias; na verdade, ela adiciona redundâncias controladas.

B) Está incorreta porque, embora a 2FN elimine dependências parciais, ela ainda permite dependências transitivas, que só são eliminadas na 3FN.

C) Está correta, pois a normalização até a terceira forma normal (3FN) é crucial para eliminar dependências parciais e transitivas, garantindo que cada tabela esteja otimizada para evitar redundâncias e preservar a integridade dos dados. A 3FN assegura que todas as colunas em uma tabela dependam unicamente da chave primária, sem introduzir colunas que dependam de outros atributos não-chave. Isso evita problemas de anomalias nas inserções, atualizações e exclusões de dados, proporcionando um modelo de banco de dados mais eficiente e confiável.

D) Está incorreta, pois confunde integridade referencial com normalização; a criação de chaves estrangeiras é fundamental para relacionamentos entre tabelas, mas não elimina dependências funcionais ou redundâncias.

E) Está incorreta porque a criação de índices, embora melhore o desempenho das consultas, não tem impacto na eliminação de redundâncias e pode até aumentar o consumo de espaço e afetar a performance de inserções e atualizações.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

NORMALIZAÇÃO

“A normalização é um processo baseado no conceito de forma normal (FN), que pode ser vista como uma regra, a qual deve ser observada na semântica de uma tabela, para que a considerem bem projetada. (...) Para fins práticos, no contexto da maior parte dos projetos de banco de dados relacionais, costumamos executar o processo de normalização até a 3FN.”

Desafio 2

Você está trabalhando como desenvolvedor de sistemas em uma empresa que utiliza um banco de dados relacional para armazenar informações críticas de clientes. Durante uma análise de performance, foi identificado que algumas tabelas apresentam redundâncias significativas nos dados. Para otimizar o sistema, você decide aplicar técnicas de normalização para melhorar a estrutura do banco de dados.

Qual etapa do processo de normalização deve ser aplicada para eliminar dependências funcionais parciais, garantindo que não haja redundância em colunas não chave de uma tabela?

A

Aplicar a Terceira Forma Normal (3FN).

B

Aplicar a Primeira Forma Normal (1FN).

C

Aplicar a Segunda Forma Normal (2FN).

D

Aplicar a Forma Normal de Boyce-Codd (FNBC).

E

Aplicar a Quarta Forma Normal (4FN).

A alternativa C está correta.

A) Aplicar a Terceira Forma Normal (3FN): Incorreta. A 3FN é aplicada para eliminar dependências transitivas, ou seja, quando uma coluna não chave depende de outra coluna não chave. No entanto, antes de alcançar a 3FN, é necessário que a tabela já esteja na 2FN, que trata especificamente das dependências parciais.

B) Aplicar a Primeira Forma Normal (1FN): Incorreta. A 1FN elimina atributos compostos e multivalorados, mas não trata das dependências funcionais parciais. É um passo inicial na normalização, mas insuficiente para resolver problemas de redundância derivados de dependências parciais.

C) Aplicar a Segunda Forma Normal (2FN): Correta. A 2FN é aplicada para eliminar dependências funcionais parciais, onde uma coluna não chave depende de parte da chave primária composta. Essa forma normal garante que todas as colunas não chave dependam integralmente da chave primária, evitando redundâncias que podem ocorrer em uma tabela que ainda está na 1FN.

D) Aplicar a Forma Normal de Boyce-Codd (FNBC): Incorreta. A FNBC é uma generalização da 3FN que lida com certos tipos de anomalias não cobertas pela 3FN, mas, assim como a 3FN, pressupõe que a tabela já esteja na 2FN.

E) Aplicar a Quarta Forma Normal (4FN): Incorreta. A 4FN trata de multivalorização e é aplicada após a FNBC, mas não é relevante para eliminar dependências funcionais parciais, que são abordadas na 2FN.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

SEGUNDA FORMA NORMAL (2FN)

“Uma tabela está na 2FN, caso esteja na 1FN e não haja dependências funcionais parciais. Se analisarmos com um pouco mais de atenção cada coluna não chave da tabela, iremos perceber que, além da coluna NOME, as colunas CATEGORIA e SALARIO também só dependem da coluna CODIGODOCENTE. Para ficar de acordo com a 2FN, será necessário eliminarmos as dependências parciais, conforme os passos a seguir...”

Desafio 3

Ao trabalhar no projeto de um banco de dados para uma empresa de recursos humanos, você percebe a necessidade de representar dependentes de funcionários no sistema. Cada funcionário pode ter vários dependentes, mas cada dependente está vinculado a um único funcionário. Qual seria a melhor prática de design de banco de dados para representar essa relação de forma eficiente e garantir a integridade dos dados?

A

Criar uma tabela separada para dependentes e adicionar uma chave estrangeira para a tabela de funcionários.

B

Incluir os dados dos dependentes diretamente na tabela de funcionários, adicionando colunas para cada dependente.

C

Criar uma tabela para dependentes e uma chave composta que inclua o código do funcionário e um identificador de dependente.

D

Adicionar os dados dos dependentes como uma lista de valores dentro de uma única coluna na tabela de funcionários.

E

Usar um campo de texto na tabela de funcionários para armazenar informações sobre os dependentes.

A alternativa A está correta.

A) Criar uma tabela separada para dependentes e adicionar uma chave estrangeira para a tabela de funcionários: Correta.

Essa é a prática recomendada de design de banco de dados porque permite uma organização clara e eficiente dos dados, mantendo a integridade referencial. Criar uma tabela separada para dependentes, com uma chave estrangeira que referencia a tabela de funcionários, garante que cada dependente esteja sempre associado ao funcionário correto. Essa estrutura evita duplicação de dados, facilita a manutenção e permite que a relação entre funcionários e dependentes seja facilmente escalável, permitindo que cada funcionário tenha vários dependentes sem precisar modificar a estrutura da tabela.

B) Incluir os dados dos dependentes diretamente na tabela de funcionários, adicionando colunas para cada dependente: Incorreta.

Adicionar colunas para cada dependente na tabela de funcionários não é uma prática recomendada porque limita a flexibilidade do design do banco de dados. Isso criaria redundâncias e anomalias, como a necessidade de modificar a estrutura da tabela toda vez que um novo dependente é adicionado, o que não é escalável. Além disso, essa abordagem resulta em uma tabela de funcionários inflada e difícil de gerenciar, especialmente em casos onde o número de dependentes varia significativamente entre os funcionários.

C) Criar uma tabela para dependentes e uma chave composta que inclua o código do funcionário e um identificador de dependente: Incorreta.

Embora a criação de uma tabela separada para dependentes seja correta, o uso de uma chave composta é desnecessário e pode complicar a estrutura do banco de dados sem necessidade. O identificador único para cada dependente e uma chave estrangeira para o funcionário são suficientes para garantir a integridade e a eficiência do sistema. Uma chave composta não cumpre os requisitos do sistema uma vez que um dependente esteja ligado a mais de um funcionário.

D) Adicionar os dados dos dependentes como uma lista de valores dentro de uma única coluna na tabela de funcionários: Incorreta.

Armazenar os dados dos dependentes como uma lista de valores em uma única coluna na tabela de funcionários é uma prática inadequada, pois viola os princípios da normalização de bancos de dados. Isso torna a consulta e a manutenção dos dados difíceis, além de impedir o uso eficaz de índices e chaves estrangeiras. Esse design compromete a integridade dos dados e dificulta a realização de operações de busca, filtragem e manipulação.

E) Usar um campo de texto na tabela de funcionários para armazenar informações sobre os dependentes: Incorreta.

Usar um campo de texto para armazenar informações sobre os dependentes é uma abordagem inadequada, pois trata os dados de forma não estruturada, dificultando a consulta e o controle da integridade dos dados. Essa prática não permite relacionamentos claros entre tabelas, torna a manutenção dos dados complexa e propensa a erros, além de violar os princípios de normalização. Isso compromete a eficiência do sistema e a qualidade das consultas SQL.

Para saber mais sobre esse conteúdo, acesse:

Módulo 1

COMPONENTES DE UMA TABELA

“A primeira linha da tabela de exemplo contém os seguintes campos ou cabeçalhos de coluna: (...) Quando criarmos uma tabela, devemos definir se o valor da coluna é opcional ou obrigatório (...) Quando escolhemos uma coluna para ser chave primária, dizemos que estamos diante de uma chave simples. Se escolhermos mais de uma coluna, a chave é dita composta.”

Desafio 4

Você está trabalhando em um projeto de banco de dados para uma organização que lida com grandes volumes de dados relacionados a transações financeiras. Durante o processo de normalização, é necessário identificar e eliminar todas as dependências funcionais parciais. Qual procedimento deve ser adotado para garantir que o banco de dados esteja na Segunda Forma Normal (2FN)?

A

Eliminar todos os atributos compostos e multivalorados para garantir que a tabela esteja na Primeira Forma Normal (1FN).

B

Remover todas as dependências transitivas, assegurando que os atributos dependam somente da chave primária.

C

Garantir que todas as tabelas estejam em 1FN e que não haja dependências funcionais parciais, onde os atributos dependam completamente da chave primária.

D

Implementar a 3FN diretamente, sem passar pelas etapas de 1FN e 2FN, para eliminar qualquer dependência funcional.

E

Adicionar índices em todas as colunas para acelerar as consultas e garantir a integridade referencial.

A alternativa C está correta.

A) Incorreta porque eliminar atributos compostos e multivalorados é um passo necessário para alcançar a 1FN, mas não garante que a tabela estará em 2FN, já que a 2FN também lida com dependências funcionais parciais.

B) Incorreta porque remover dependências transitivas é um requisito para a 3FN, não para a 2FN.

C) Correta, pois a Segunda Forma Normal (2FN) exige que a tabela esteja primeiramente na Primeira Forma Normal (1FN) e, em seguida, assegura que todos os atributos não-chave dependam completamente da chave primária, eliminando quaisquer dependências funcionais parciais. Isso significa que, para cada atributo em uma tabela, deve existir uma relação direta e completa com a chave primária, sem depender apenas de parte dela, o que garante a integridade e eficiência do modelo de dados.

D) Incorreta porque pular diretamente para a 3FN sem passar pelas etapas de 1FN e 2FN pode levar à perda de etapas importantes de normalização, resultando em um modelo incompleto ou incorreto.

E) Incorreta porque a criação de índices pode melhorar o desempenho de consultas, mas não afeta diretamente a normalização ou a eliminação de dependências funcionais.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

SEGUNDA FORMA NORMAL (2FN)

"Uma tabela está na 2FN, caso esteja na 1FN e não haja dependências funcionais parciais. Se analisarmos com um pouco mais de atenção cada coluna não chave da tabela PROJETODOCENTE, iremos perceber que, além da coluna NOME, as colunas CATEGORIA e SALARIO também só dependem da coluna CODIGODOCENTE. (...) Manter no modelo cada tabela que possua chave primária simples; identificar cada dependência parcial; Criar uma tabela para cada dependência parcial identificada.".

Criação e Manipulação de Objetos no POSTGRESQL

Desafio 1

Você é responsável pela manutenção de um banco de dados PostgreSQL em uma empresa de logística. Durante uma auditoria, foi identificado que algumas tabelas contêm registros que já não são mais relevantes, e você foi instruído a removê-los para liberar espaço e melhorar o desempenho do sistema. No entanto, é crucial garantir que a remoção desses registros não afete a integridade dos dados restantes. Qual é a melhor prática para remover uma tabela, levando em consideração que ela possui dependências de chaves estrangeiras com outras tabelas?

A

Usar o comando DROP DATABASE

B

Remover as colunas de chave estrangeira primeiro

C

Utilizar o comando DROP TABLE CASCADE

D

Alterar a tabela para remover as restrições de chave estrangeira

E

Executar o comando DELETE FROM antes de DROP TABLE

A alternativa C está correta.

A) Usar o comando DROP DATABASE: Incorreta. O comando DROP DATABASE remove o banco de dados inteiro, incluindo todas as tabelas, funções, e dados associados. Isso é muito mais drástico do que a situação exige e resultaria na perda de todos os dados no banco de dados, não apenas da tabela específica, o que seria um erro catastrófico.

B) Remover as colunas de chave estrangeira primeiro: Parcialmente Incorreta. Remover as colunas de chave estrangeira antes de remover a tabela pode ser uma abordagem para evitar erros de dependência, mas não é a prática recomendada, pois essa ação pode gerar inconsistências nos dados de outras tabelas que dependem dessas colunas. Além disso, essa abordagem não é prática quando há muitas dependências complexas.

C) Utilizar o comando DROP TABLE CASCADE: Correta. O comando DROP TABLE CASCADE é a melhor prática para remover uma tabela com dependências de chave estrangeira no PostgreSQL. Ele remove automaticamente todas as dependências associadas, garantindo que a integridade do banco de dados seja mantida. Esse comando é especialmente útil quando há várias dependências, pois, ele simplifica o processo e evita erros manuais.

D) Alterar a tabela para remover as restrições de chave estrangeira: Parcialmente Incorreta. Alterar a tabela para remover as restrições de chave estrangeira antes de removê-la pode ser uma abordagem, mas é mais propenso a erros e pode ser mais complicado do que usar o CASCADE. Além disso, essa prática requer mais passos manuais, aumentando o risco de deixar o banco de dados em um estado inconsistente.

E) Executar o comando DELETE FROM antes de DROP TABLE: Incorreta. Executar DELETE FROM pode remover todos os registros de uma tabela, mas não remove a tabela em si. Além disso, isso não resolve o problema das dependências de chave estrangeira e não garante que a integridade dos dados seja mantida durante a remoção da tabela.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

CRIAÇÃO E ALTERAÇÃO DE TABELAS RELACIONAIS

“Suponha que temos interesse em remover a tabela NIVEL... O SGBD não removerá a tabela NIVEL e retornará uma mensagem de erro, informando que há um objeto (tabela CURSO) que depende da tabela NIVEL... Nesse caso, poderíamos usar o comando SQL a seguir: DROP TABLE NIVEL CASCADE;... Internamente, o comando altera a estrutura da tabela CURSO, removendo a restrição de chave estrangeira da coluna CODIGONIVEL. Em seguida, a tabela NIVEL é removida do banco de dados.”

Desafio 2

Como especialista em bancos de dados, você foi chamado para corrigir um problema em um sistema de gestão acadêmica. Um novo curso foi inserido com um erro no código do curso, e agora você precisa atualizar essa informação na tabela de cursos. Qual seria a abordagem correta para atualizar o código do curso?

A

UPDATE curso SET codigo_curso = 10;

B

DELETE FROM curso WHERE codigo_curso = 1;

C

ALTER TABLE curso DROP codigo_curso;

D

UPDATE curso SET codigo_curso = 10 WHERE codigo_curso = 1;

E

ALTER TABLE curso ADD CONSTRAINT ON UPDATE CASCADE;

A alternativa D está correta.

A) UPDATE curso SET codigo_curso = 10; Incorreta.

Este comando atualiza a coluna codigo_curso para 10 em todos os registros da tabela curso, sem especificar uma condição. Isso resultaria na alteração do código de todos os cursos para 10, o que é claramente incorreto. A falta de uma cláusula WHERE torna esta operação perigosa, pois pode comprometer a integridade dos dados, alterando mais registros do que o necessário.

B) DELETE FROM curso WHERE codigo_curso = 1: Incorreta.

Este comando exclui o registro do curso onde codigo_curso é igual a 1. Embora isso remova o curso com o código incorreto, não é o objetivo da tarefa. Deletar o registro resultaria na perda de todos os dados associados ao curso, o que não é desejado. A tarefa é corrigir o erro no código do curso, não excluir o curso.

C) ALTER TABLE curso DROP codigo_curso: Incorreta.

O comando ALTER TABLE ... DROP é utilizado para remover colunas inteiras de uma tabela. Usar esse comando removeria a coluna codigo_curso de todos os registros, o que não é o objetivo da correção. Este comando causaria perda de dados crítica, pois a coluna inteira seria eliminada da tabela, prejudicando a integridade do banco de dados.

D) UPDATE curso SET codigo_curso = 10 WHERE codigo_curso = 1: Correta.

Essa é a abordagem correta para corrigir o código do curso. O comando UPDATE permite modificar os valores em uma ou mais colunas de uma tabela existente. Aqui, você está atualizando o valor da coluna codigo_curso para 10, mas apenas para as linhas onde o valor atual de codigo_curso é 1. Isso corrige o erro sem afetar outras colunas ou registros na tabela. Esse comando é eficiente e direto, garantindo que apenas o código específico do curso errado seja alterado.

E) ALTER TABLE curso ADD CONSTRAINT ON UPDATE CASCADE: Incorreta.

Este comando ALTER TABLE ... ADD CONSTRAINT adiciona uma restrição à tabela, mas a cláusula ON UPDATE CASCADE é geralmente usada em conjunto com chaves estrangeiras para garantir que atualizações em uma tabela sejam refletidas automaticamente em tabelas relacionadas. Ele não corrige o erro existente no código do curso. Além disso, o comando não é suficiente para resolver o problema específico de atualizar o código de um curso já existente.

Para saber mais sobre esse conteúdo, acesse:

Módulo 3

ATUALIZAÇÃO DE LINHAS EM TABELA

“A atualização de linhas em tabela é realizada com o auxílio do comando UPDATE da SQL. Sua sintaxe básica está expressa a seguir:..”

Desafio 3

Você é responsável pelo gerenciamento de um banco de dados PostgreSQL utilizado por uma empresa de comércio eletrônico. Devido a um aumento nas transações, foi decidido que a tabela de histórico de pedidos deve ser reorganizada para melhorar o desempenho das consultas. Como parte desse processo, você precisa adicionar uma nova coluna para registrar a data de modificação de cada registro. Qual seria o comando SQL mais apropriado para realizar essa alteração?

A

ALTER TABLE pedidos ADD COLUMN data_modificacao time;

B

ALTER TABLE pedidos DROP COLUMN data_modificacao;

C

ALTER TABLE pedidos ADD COLUMN data_modificacao date;

D

ALTER TABLE pedidos ADD CONSTRAINT CHECK (data_modificacao > current_date);

E

ALTER TABLE pedidos ALTER COLUMN data_modificacao SET DEFAULT current_date;

A alternativa C está correta.

A) ALTER TABLE pedidos ADD COLUMN data_modificacao time;: Incorreta. O tipo de dado time é utilizado para armazenar apenas a hora do dia, sem a data. Isso não atende ao requisito de registrar a data completa de modificação dos registros, o que poderia resultar em perda de informações essenciais sobre quando uma modificação foi feita.

B) ALTER TABLE pedidos DROP COLUMN data_modificacao;: Incorreta. Este comando remove uma coluna, o que vai contra o objetivo de adicionar uma nova coluna para registrar a data de modificação. Remover uma coluna não só falha em cumprir a tarefa, mas também pode resultar na perda de dados críticos, se a coluna já existisse.

C) ALTER TABLE pedidos ADD COLUMN data_modificacao date;: Correta. O comando ALTER TABLE com ADD COLUMN permite a adição de uma nova coluna na tabela pedidos. Definir o tipo de dado como date é apropriado para armazenar a data completa de modificação dos registros, cumprindo o requisito de registrar a data de modificação. Essa prática é comum em sistemas que necessitam rastrear alterações em registros.

D) ALTER TABLE pedidos ADD CONSTRAINT CHECK (data_modificacao > current_date);: Parcialmente Incorreta. Embora a cláusula CHECK possa ser útil para validar os valores inseridos na coluna data_modificacao, ela não adiciona uma nova coluna à tabela, o que é o objetivo principal do comando necessário para esta situação. Além disso, validar que data_modificacao seja maior que current_date pode não ser adequado em todos os contextos.

E) ALTER TABLE pedidos ALTER COLUMN data_modificacao SET DEFAULT current_date;: Parcialmente Incorreta. Este comando modifica uma coluna existente para definir um valor padrão, mas não adiciona uma nova coluna à tabela. Além disso, definir o valor padrão como current_date pode ser útil, mas isso não é suficiente para adicionar a coluna em primeiro lugar, que é o que a questão pede.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

ALTERAÇÃO DE TABELAS

“Suponha que surgiu a necessidade de modelar a informação sobre a data de primeiro reconhecimento do curso. Podemos, então, alterar a estrutura da tabela CURSO, adicionando uma coluna opcional denominada DTRECONH. O comando ALTER TABLE é útil no contexto dessa tarefa. A seguir, sintaxe básica do comando ALTER TABLE: ALTER TABLE <NOMETABELA> ADD <COLUNA> <TIPODEDADOS> ;”

Desafio 4

Você é o responsável pelo gerenciamento de um banco de dados em uma empresa de investimentos e está realizando uma série de operações financeiras em uma única transação. Durante o processo, um erro foi detectado em uma das operações, e é necessário desfazer todas as mudanças feitas desde o início da transação para evitar inconsistências no banco de dados. Qual comando SQL você deve utilizar para garantir que nenhuma alteração seja aplicada ao banco de dados, revertendo todas as operações realizadas até o momento?

A

COMMIT;

B

SAVEPOINT;

C

ROLLBACK;

D

START TRANSACTION;

E

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

A alternativa C está correta.

A) COMMIT;: Incorreta. O comando COMMIT finaliza uma transação aplicando permanentemente todas as mudanças feitas ao banco de dados. No entanto, no caso de um erro durante a transação, COMMIT não deve ser usado, pois isso confirmaria as mudanças, mesmo as incorretas.

B) SAVEPOINT;: Parcialmente Incorreta. SAVEPOINT permite definir pontos de salvamento dentro de uma transação, aos quais você pode voltar se algo der errado, mas não reverte toda a transação. SAVEPOINT é útil em operações complexas, mas, sozinho, não reverte todas as mudanças realizadas na transação desde o início.

C) ROLLBACK;: Correta. O comando ROLLBACK reverte todas as operações feitas desde o início da transação, garantindo que nenhuma alteração seja aplicada ao banco de dados. Isso é crucial quando um erro é detectado e você precisa garantir que o banco de dados volte ao estado anterior ao início da transação, mantendo a integridade dos dados.

D) START TRANSACTION;: Parcialmente Incorreta. START TRANSACTION é usado para iniciar uma transação, agrupando operações subsequentes para serem confirmadas ou revertidas como uma unidade. Contudo, após um erro, esse comando não desfaz as operações realizadas. Em vez disso, ele deve preceder um ROLLBACK para que a transação possa ser desfeita se necessário.

E) SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;: Incorreta. Esse comando ajusta o nível de isolamento de uma transação, definindo como as transações são executadas em relação umas às outras, mas não reverte operações. Embora útil para evitar problemas como leituras sujas ou não repetíveis, ele não reverte alterações feitas na transação.

Para saber mais sobre esse conteúdo, acesse:

Módulo 4

TRANSAÇÕES NO POSTGRESQL

“O comando ROLLBACK é utilizado para reverter todas as operações realizadas em uma transação desde o seu início, assegurando que nenhuma mudança seja aplicada ao banco de dados. Este comando é essencial para manter a integridade dos dados em caso de falhas durante a execução de uma transação.”

Consultas em uma Tabela no POSTGRESQL

Desafio 1

Você está desenvolvendo uma aplicação que precisa exibir informações detalhadas de uma tabela de produtos em um banco de dados PostgreSQL. Seu desafio é criar uma consulta SQL que retorne todas as colunas da tabela produtos, mas você também precisa criar um alias para a coluna preço, renomeando-a para "Preço Unitário" na exibição dos resultados. Como você deve estruturar a consulta SQL?

A

SELECT * FROM produtos WHERE preco AS "Preço Unitário";

B

SELECT ALL FROM produtos AS "Preço Unitário";

C

SELECT preco AS "Preço Unitário", * FROM produtos;

D

SELECT *, preco AS "Preço Unitário" FROM produtos;

E

SELECT preco AS "Preço Unitário" FROM produtos;

A alternativa D está correta.

A) Incorreta. A cláusula WHERE é utilizada para especificar condições de filtragem de registros, mas não serve para criar aliases de colunas. Além disso, não é a maneira correta de renomear uma coluna para exibição.

B) Incorreta. SELECT ALL não é uma sintaxe válida para selecionar todas as colunas de uma tabela. Além disso, AS está sendo mal utilizado ao tentar renomear a tabela em vez de uma coluna.

C) Incorreta. Embora esta consulta esteja próxima, a ordem das colunas no SELECT é importante. O uso do * antes de preco AS "Preço Unitário" causaria ambiguidade e não exibe corretamente o alias junto com todas as colunas.

D) Correta. A consulta SELECT *, preco AS "Preço Unitário" FROM produtos; está corretamente estruturada. O * seleciona todas as colunas da tabela, e a cláusula AS renomeia a coluna preco para "Preço Unitário", que será exibido no resultado da consulta.

E) Incorreta. Esta consulta apenas seleciona a coluna preco renomeada como "Preço Unitário", mas não inclui as demais colunas da tabela produtos, o que não atende ao requisito de exibir todas as colunas.

Para saber mais sobre esse conteúdo, acesse:

Módulo 1

ESTRUTURA BÁSICA DE UM COMANDO SELECT

“O comando SELECT é usado para exibir dados resultantes de uma consulta. Os dados podem ser colunas físicas de uma tabela, colunas calculadas ou mesmo resultado do uso de expressões e funções. Uma sintaxe básica para o comando SELECT está expressa a seguir: SELECT COLUNA1 [[AS] APELIDOCOLUNA1], COLUNA2 [[AS] APELIDOCOLUNA2], ... COLUNAN [[AS] APELIDOCOLUNAN] FROM TABELA;”

Desafio 2

Você foi contratado para trabalhar em uma empresa de análise de dados financeiros. Seu gerente solicita que você crie um relatório que mostre a quantidade de transações por tipo de transação em um banco de dados PostgreSQL. Além disso, ele quer que o relatório exiba apenas os tipos de transação que tiveram mais de 100 ocorrências. Como você deve estruturar a consulta SQL para atender a essa necessidade?

A

SELECT tipo_transacao, COUNT(*) FROM transacoes WHERE COUNT(*) > 100 GROUP BY tipo_transacao;

B

SELECT tipo_transacao, COUNT(*) FROM transacoes GROUP BY tipo_transacao HAVING COUNT(*) > 100;

C

SELECT tipo_transacao, COUNT(*) FROM transacoes GROUP BY tipo_transacao WHERE COUNT(*) > 100;

D

SELECT tipo_transacao, COUNT(*) FROM transacoes HAVING COUNT(*) > 100 GROUP BY tipo_transacao;

E

SELECT tipo_transacao, COUNT(*) FROM transacoes GROUP BY tipo_transacao ORDER BY COUNT(*) > 100;

A alternativa B está correta.

A) Incorreta. A cláusula WHERE é utilizada para filtrar registros antes de qualquer agregação ocorrer. Neste caso, WHERE não pode ser usada para filtrar grupos após a aplicação da função de agregação COUNT(*). A cláusula correta para esse propósito é HAVING.

B) Correta. A cláusula GROUP BY é usada para agrupar os registros por tipo_transacao, e a cláusula HAVING é usada para filtrar os grupos com mais de 100 ocorrências após a aplicação da agregação. Esta é a estrutura correta para a consulta solicitada.

C) Incorreta. A cláusula WHERE não pode ser usada para filtrar grupos após a agregação, como neste caso em que se deseja filtrar pelo resultado da função COUNT(*). A cláusula correta para isso é HAVING.

D) Incorreta. Embora a consulta utilize HAVING, a ordem das cláusulas está incorreta. O GROUP BY deve preceder a cláusula HAVING, pois os dados precisam ser agrupados antes de serem filtrados.

E) Incorreta. A cláusula ORDER BY é usada para ordenar os resultados, mas não é adequada para filtrar grupos com base em agregações. Além disso, a comparação COUNT(*) > 100 na ORDER BY não é válida.

Para saber mais sobre esse conteúdo, acesse:

Módulo 3

GRUPO DE DADOS COM GROUP BY E HAVING

“Até o momento, utilizamos a cláusula WHERE para programar filtros em consultas, com condições simples ou compostas envolvendo colunas da tabela ou funções de data. Contudo, você vai vivenciar situações onde será necessário estabelecer algum tipo de filtro, tendo como base um cálculo originado a partir de uma função de agregação, não sendo possível usar a cláusula WHERE. Nesses casos, utilizamos a cláusula HAVING, que serve justamente para esse propósito.”

Desafio 3

Você está trabalhando como desenvolvedor em uma empresa que precisa realizar uma análise detalhada dos dados de vendas. Seu gerente solicita que você crie uma consulta SQL que retorne todos os registros de vendas feitas no mês de julho, com valores acima de R$10.000.

Como você deve estruturar a cláusula WHERE para filtrar os registros corretamente?

A

SELECT * FROM vendas WHERE valor > 10000 AND data_venda BETWEEN '2024-07-01' AND '2024-07-31';

B

SELECT * FROM vendas WHERE valor >= 10000 OR data_venda = '2024-07';

C

SELECT * FROM vendas WHERE valor = 10000 AND data_venda IN ('2024-07-01', '2024-07-31');

D

SELECT * FROM vendas WHERE valor > 10000 AND data_venda LIKE '2024-07';

E

SELECT * FROM vendas WHERE valor > 10000 ORDER BY data_venda = '2024-07';

A alternativa A está correta.

A) Correta. A consulta usa a cláusula WHERE corretamente, filtrando os registros com valor superior a R$10.000 e que ocorreram entre as datas '2024-07-01' e '2024-07-31', representando todo o mês de julho. O operador BETWEEN é apropriado para selecionar um intervalo de datas.

B) Incorreta. O operador OR neste contexto pode causar resultados indesejados, pois retornará registros onde o valor é maior ou igual a 10.000 ou qualquer venda feita em julho, mas não necessariamente ambas as condições. Além disso, data_venda = '2024-07' não é uma comparação válida para uma data específica.

C) Incorreta. A comparação valor = 10000 retorna apenas registros com exatamente R$10.000, o que não atende ao requisito de valores superiores. Além disso, a comparação de data_venda IN ('2024-07-01', '2024-07-31') só verifica se a data é um dos dois valores específicos, não abrangendo todo o mês de julho.

D) Incorreta. O operador LIKE é geralmente usado para buscas de padrões em strings e não é adequado para comparar datas. data_venda LIKE '2024-07' não é uma comparação válida para filtrar datas no formato YYYY-MM-DD.

E) Incorreta. A cláusula ORDER BY é utilizada para ordenar resultados, não para filtrar. A expressão ORDER BY data_venda = '2024-07' não é uma sintaxe válida para ordenação ou filtragem de registros com base em datas.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

CLÁUSULA WHERE E OPERADORES DA SQL

“Em nossas consultas, usaremos como base a tabela ALUNO... No WHERE, especificamos alguma condição, simples ou composta, para filtrar registros que serão recuperados pelo SGBD. É importante frisar que a cláusula WHERE realiza a operação de restrição da Álgebra Relacional, também conhecida como seleção.”

Desafio 4

Você está trabalhando em um sistema de cadastro de clientes e precisa gerar um relatório que exiba todos os clientes que ainda não forneceram um endereço de e-mail. O banco de dados PostgreSQL utilizado permite que o campo de e-mail seja opcional, o que significa que alguns registros podem ter essa informação em branco ou nula.

Como você deve estruturar a consulta SQL para recuperar os registros desses clientes?

A

SELECT * FROM clientes WHERE email IS NOT NULL;

B

SELECT * FROM clientes WHERE email = NULL;

C

SELECT * FROM clientes WHERE email IS NULL;

D

SELECT * FROM clientes WHERE email = '';

E

SELECT * FROM clientes WHERE email LIKE '%NULL%';

A alternativa C está correta.

A) Incorreta. A consulta WHERE email IS NOT NULL retornaria os registros onde o campo de e-mail está preenchido, ou seja, onde o e-mail não é nulo. Isso não atende ao objetivo de listar clientes que não forneceram e-mail.

B) Incorreta. Comparar diretamente um campo com NULL usando = não é correto em SQL. O operador = não pode ser usado para verificar valores nulos, pois NULL não é considerado um valor comparável.

C) Correta. A consulta WHERE email IS NULL é a maneira correta de recuperar registros onde o campo de e-mail é nulo, ou seja, onde o cliente não forneceu um e-mail. Essa consulta filtra corretamente os clientes que ainda não têm e-mail cadastrado.

D) Incorreta. A condição WHERE email = '' verifica se o campo de e-mail está vazio, mas não verifica se o campo é nulo. Embora possa capturar e-mails em branco, ela não detecta casos onde o valor é realmente NULL.

E) Incorreta. A consulta WHERE email LIKE '%NULL%' tenta buscar uma string "NULL" dentro do campo de e-mail, mas isso não é adequado para identificar valores realmente nulos no banco de dados. LIKE é usado para buscas de padrões em strings, não para comparar com NULL.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

RECUPERANDO DADOS COM O USO DO OPERADOR NULL

“Quando uma coluna é opcional, significa que existe possibilidade de que algum registro não possua valor cadastrado para a coluna em questão. Nessa hipótese, há entendimento de que o valor da coluna é 'desconhecido' ou 'não aplicável'. Para testar se uma coluna possui valor cadastrado, usa-se a expressão COLUNA IS NOT NULL.”

Consulta com várias Tabelas no POSTGRESQL

Desafio 1

Você foi contratado para otimizar as consultas em um banco de dados de uma empresa que possui diversas tabelas relacionadas, como clientes e pedidos. Para garantir que os relatórios gerados incluam apenas os dados válidos, qual operação de junção você deve utilizar?

A

LEFT JOIN

B

FULL OUTER JOIN

C

CROSS JOIN

D

INNER JOIN

E

RIGHT JOIN

A alternativa D está correta.

A) Incorreta. LEFT JOIN retornaria todos os registros da tabela de clientes, incluindo aqueles sem correspondência na tabela de pedidos, o que não atende à necessidade de exibir apenas os registros com uma relação válida entre as tabelas.

B) Incorreta. FULL OUTER JOIN incluiria todos os registros de ambas as tabelas, independentemente de correspondência, o que resultaria em dados extras desnecessários para a consulta desejada.

C) Incorreta. CROSS JOIN gera o produto cartesiano entre as tabelas, criando combinações de todos os registros, o que não é útil para filtrar relações válidas entre clientes e pedidos.

D) Correta. INNER JOIN é a operação correta para exibir apenas os registros que possuem uma correspondência válida entre as tabelas envolvidas. Neste caso, mostra apenas os clientes que têm pedidos relacionados, garantindo a precisão dos relatórios.

E) Incorreta. RIGHT JOIN retornaria todos os pedidos, incluindo aqueles que não têm um cliente associado, o que também resultaria em dados irrelevantes para a necessidade apresentada.

Para saber mais sobre esse conteúdo, acesse:

Módulo 1

JUNÇÃO INTERNA

"Perceba que foram recuperadas as linhas que de fato relacionam cursos aos seus respectivos níveis. No entanto, a forma mais usada para retornar os mesmos resultados é com o auxílio da cláusula de junção interna, o qual possui sintaxe básica conforme a seguir: SELECT *

FROM TABELA1 [INNER] JOIN TABELA2 ON (CONDIÇÃOJUNÇÃO) [USING (COLUNA_DE_JUNÇÃO)] ..."

Desafio 2

Você está desenvolvendo um sistema de gerenciamento de funcionários em PostgreSQL e precisa exibir uma lista de funcionários que ganham mais do que a média salarial de seus respectivos departamentos. Para isso, você decide utilizar uma subconsulta. Qual das abordagens abaixo é a mais adequada para obter o resultado desejado?

A

Utilizar uma subconsulta no SELECT para calcular o salário máximo por departamento.

B

Utilizar uma subconsulta no WHERE para comparar o salário de cada funcionário com a média salarial do departamento.

C

Utilizar uma subconsulta no FROM para gerar uma tabela temporária com os salários de todos os funcionários.

D

Utilizar uma subconsulta no JOIN para combinar as tabelas de funcionários e departamentos.

E

Utilizar uma subconsulta no GROUP BY para agrupar os funcionários por departamento.

A alternativa B está correta.

A) Incorreta. Embora calcular o salário máximo por departamento possa ser útil em outros contextos, não atende ao objetivo de comparar o salário de cada funcionário com a média salarial do departamento, que é necessário para identificar aqueles que ganham acima da média.

B) Correta. A subconsulta no WHERE é a abordagem ideal para essa situação. A subconsulta permite calcular a média salarial do departamento, e o WHERE a utiliza para comparar cada salário individualmente. Isso garante que o resultado inclua apenas os funcionários que ganham mais do que a média do seu departamento.

C) Incorreta. Embora seja possível usar uma subconsulta no FROM para criar uma tabela temporária, essa abordagem seria mais complexa e menos eficiente do que usar a subconsulta diretamente no WHERE, que faz a comparação necessária.

D) Incorreta. Uma subconsulta no JOIN não seria apropriada para comparar os salários com a média salarial. O JOIN é mais adequado para combinar dados de diferentes tabelas, não para realizar comparações baseadas em agregações.

E) Incorreta. O GROUP BY é usado para agrupar dados com base em uma ou mais colunas e aplicar funções de agregação, mas não seria adequado para realizar a comparação entre os salários individuais e a média salarial dentro do mesmo departamento.

Para saber mais sobre esse conteúdo, acesse:

Módulo 2

SUBCONSULTAS ANINHADAS

“Uma subconsulta aninhada ocorre quando é necessário obter dados que dependem do resultado de uma - ou mais - consulta(s) mais interna(s). Para isso, cria-se uma condição na cláusula WHERE de forma a envolver o resultado da subconsulta em algum tipo de teste, como comparar o salário de cada funcionário com a média salarial de seu departamento.”

Desafio 3

Você é responsável por desenvolver um sistema de gerenciamento de clientes para uma empresa. O sistema armazena informações sobre clientes em diferentes tabelas, como CLIENTES_ATIVOS e CLIENTES_INATIVOS. Para gerar uma lista de todos os clientes que estiveram ativos em algum momento, mas que atualmente estão inativos, qual operador de conjunto você deve utilizar em sua consulta SQL?

A

UNION

B

INTERSECT

C

EXCEPT

D

FULL OUTER JOIN

E

LEFT JOIN

A alternativa C está correta.

A) Incorreta. O operador UNION combina todos os registros de ambas as tabelas, incluindo tanto clientes ativos quanto inativos. Este operador não é adequado para subtrair os clientes que ainda estão ativos.

B) Incorreta. INTERSECT é usado para encontrar registros comuns entre duas tabelas, ou seja, clientes que são tanto ativos quanto inativos simultaneamente, o que não faz sentido neste contexto.

C) Correta. EXCEPT é o operador de conjunto ideal para esta situação. Ele permite que você subtraia os registros da tabela CLIENTES_ATIVOS dos registros da tabela CLIENTES_INATIVOS, resultando em uma lista de clientes que estiveram ativos em algum momento, mas que atualmente estão inativos.

D) Incorreta. FULL OUTER JOIN combina todos os registros de ambas as tabelas, independentemente de correspondência, o que resultaria em uma lista que inclui tanto clientes ativos quanto inativos, sem diferenciá-los.

E) Incorreta. LEFT JOIN é usado para incluir todos os registros da tabela à esquerda e os registros correspondentes da tabela à direita. Isso não ajuda a identificar exclusivamente os clientes que estão inativos.

Para saber mais sobre esse conteúdo, acesse:

Módulo 3

CONSULTAS COM O OPERADOR EXCEPT

“O operador EXCEPT implementa a operação de subtração da Teoria dos Conjuntos e serve para exibir linhas que aparecem em uma consulta e não aparecem na outra. Este operador é ideal para identificar registros exclusivos de uma tabela, subtraindo os registros que também aparecem em outra tabela.”

Desafio 4

Você está desenvolvendo uma aplicação de relatórios para uma grande empresa de e-commerce que possui uma tabela de CLIENTES com milhões de registros e uma tabela de PEDIDOS com dezenas de milhões de registros. O gerente pediu para você gerar uma lista que combine todos os clientes com todos os pedidos. Por que o uso da operação CROSS JOIN pode ser ineficiente neste cenário?

A

Porque o CROSS JOIN pode retornar resultados duplicados, aumentando o tempo de processamento.

B

Porque o CROSS JOIN pode gerar um número excessivamente grande de combinações, sobrecarregando o servidor.

C

Porque o CROSS JOIN não é compatível com tabelas que possuem um grande número de colunas.

D

Porque o CROSS JOIN não permite a filtragem de registros, retornando todos os dados de ambas as tabelas.

E

Porque o CROSS JOIN é mais lento do que outros tipos de junção, como o INNER JOIN ou LEFT JOIN.

A alternativa B está correta.

A) Incorreta. O CROSS JOIN não necessariamente retorna resultados duplicados, mas combina todos os registros de ambas as tabelas, o que pode resultar em um grande número de combinações.

B) Correta. O CROSS JOIN gera o produto cartesiano de duas tabelas, criando todas as combinações possíveis entre os registros das tabelas envolvidas. Em um cenário onde uma tabela possui milhões de registros e outra possui dezenas de milhões, isso pode resultar em trilhões de combinações, o que sobrecarrega o servidor e pode levar a um tempo de processamento excessivamente longo, além de retornar um conjunto de dados massivo e, na maioria das vezes, inútil.

C) Incorreta. A ineficiência do CROSS JOIN neste contexto não está relacionada ao número de colunas nas tabelas, mas sim ao número de combinações de registros que ele gera.

D) Incorreta. Embora o CROSS JOIN retorne todas as combinações possíveis, ele não impede a filtragem de registros; a filtragem pode ser aplicada após o CROSS JOIN, mas isso não resolve o problema da explosão combinatória.

E) Incorreta. O CROSS JOIN pode ser mais lento em certos contextos devido ao grande volume de dados gerado, mas a principal questão aqui é a quantidade de combinações geradas, não a velocidade de execução em si.

Para saber mais sobre esse conteúdo, acesse:

Módulo 1

OPERAÇÃO DE PRODUTO CARTESIANO

“Cabe aqui um alerta sobre o cuidado que se deve ter ao usar esse tipo de consulta, que resulta em produto cartesiano. Suponha uma tabela CLIENTES com 1.000.000 de linhas e uma tabela PEDIDOS com 10.000.000 de linhas. O comando SELECT * FROM CLIENTES, PEDIDOS retornaria 10.000.000.000.000 de linhas e, provavelmente, ocuparia longas horas de processamento no servidor, além de retornar um resultado inútil!”

Considerações finais

Continue explorando, praticando e desafiando-se. Cada exercício é uma oportunidade de crescimento e cada erro, uma lição valiosa. Que sua jornada de aprendizado seja repleta de descobertas e realizações. Bons estudos e sucesso na sua carreira!

Compartilhe conosco como foi sua experiência com este conteúdo. Por favor, responda a este formulário de avaliação e nos ajude a aprimorar ainda mais a sua experiência de aprendizado!