CENTRO DE ENSINO SUPERIOR E DESENVOLVIMENTO – CESED

CENTRO UNIVERSITÁRIO – UNIFACISA

CURSO: SISTEMAS DE INFORMAÇÃO

PROFESSOR: JOSÉ ANDERSON RODRIGUES DE SOUZA

COMPONENTE CURRICULAR: CONECTAR BANCO DE DADOS COM P-O-O

LISTA DE EXERCÍCIO 01

(Persistência a Dados, Tipos de Dados, XML, JSON e JDBC)

1. O que você entende por Persistência de Dados?

A persistência de dados refere-se à capacidade de armazenar informações de forma permanente e duradoura, de modo que os dados permaneçam intactos mesmo quando o programa ou sistema que os criou ou manipulou for encerrado ou reiniciado. É uma parte fundamental do desenvolvimento de software e sistemas de informação, pois permite que as informações sejam retidas e acessadas posteriormente, mesmo após o término da execução de um programa ou desligamento de um dispositivo. A persistência de dados pode ser implementada de várias maneiras, dependendo das necessidades e requisitos do sistema.

2. Quais as diferenças entre objetos transientes e objetos persistentes? Explique.

Objetos transientes e objetos persistentes são conceitos comuns na programação e no gerenciamento de dados, especialmente em bancos de dados e sistemas de armazenamento. Eles se referem à forma como os dados são tratados em diferentes contextos. As principais diferenças entre eles são:

Objetos Transientes:

São temporários: Objetos transientes existem apenas temporariamente na memória durante a execução de um programa. Eles não são armazenados permanentemente.

Não são persistidos: Não há nenhum mecanismo automático para salvar objetos transientes em um armazenamento persistente, como um banco de dados. Quando o programa é encerrado, os objetos transientes desaparecem.

Utilização em memória: São utilizados principalmente para manipulação temporária de dados durante a execução de um programa, mas não são projetados para serem armazenados a longo prazo.

Objetos Persistentes:

Armazenamento permanente: Objetos persistentes são projetados para serem armazenados permanentemente em algum tipo de armazenamento durável, como um banco de dados, um arquivo ou outro meio de persistência.

Sobrevivem ao encerramento do programa: Eles não são limitados à vida útil de um programa; em vez disso, podem ser recuperados e utilizados novamente mesmo após o encerramento e reinicialização do programa.

Mapeamento para armazenamento: Geralmente, os objetos persistentes são mapeados para uma estrutura de armazenamento específica, como tabelas em um banco de dados relacional, documentos em um banco de dados NoSQL ou registros em um arquivo.

Exemplo: Registros de clientes em um banco de dados, configurações de um aplicativo salvas em um arquivo, documentos em um sistema de gerenciamento de documentos.

3. Qual a principal diferença entre Banco de Dados Relacional e Banco de Dados Orientada a Objetos?

A principal diferença entre bancos de dados relacionais e bancos de dados orientados a objetos está na forma como eles modelam e organizam os dados.

Em resumo, a principal diferença é que os bancos de dados relacionais são baseados em tabelas e seguem um esquema fixo, enquanto os bancos de dados orientados a objetos modelam os dados como objetos com flexibilidade de esquema e suporte a recursos de programação orientada a objetos, como herança. A escolha entre eles depende dos requisitos específicos do projeto e das preferências de modelagem de dados.

4. Qual o objetivo do mapeamento objeto-relacional (ORM)?

O mapeamento objeto-relacional (ORM) é uma técnica de programação que permite que os objetos de um sistema de software sejam mapeados diretamente para as tabelas de um banco de dados relacional. Isso facilita a interação entre o código orientado a objetos e os sistemas de gerenciamento de banco de dados relacionais, eliminando a necessidade de escrever consultas SQL manualmente.

A principal vantagem do ORM é que ele simplifica o desenvolvimento de aplicativos, tornando-o

mais eficiente e menos propenso a erros. Em vez de lidar com SQL bruto e manipulação manual de dados, os desenvolvedores podem trabalhar com objetos e classes em suas linguagens de programação orientadas a objetos preferidas, como Java, Python, C#, etc.

5. Defina:

a) Dados estruturados;

Exemplos de dados estruturados incluem registros de clientes em um banco de dados, informações de produtos em um catálogo online, registros de vendas em uma loja, informações de estudantes em uma escola e muito mais. Esses tipos de dados são comuns em muitos contextos comerciais e podem ser facilmente manipulados e analisados usando ferramentas e técnicas específicas.

b) Dados semiestrurados;

Dados semiestruturados são um tipo de dados que não se encaixam perfeitamente em um modelo de dados estruturado tradicional, como tabelas em um banco de dados relacional, nem são completamente desorganizados como dados não estruturados. Em vez disso, eles estão em algum lugar entre esses dois extremos.

Os dados semiestruturados geralmente têm uma estrutura básica, mas essa estrutura pode variar de um registro para outro e não é rigorosamente definida. Além disso, os dados semiestruturados são frequentemente armazenados em formatos como XML, JSON, YAML ou mesmo em documentos de texto simples, onde os elementos de dados são identificados por meio de tags, chaves ou outros marcadores, tornando possível interpretar e organizar os dados de maneira significativa.

c) Dados não estruturados.

Dados não estruturados referem-se a informações que não estão organizadas em um formato tabular ou altamente organizado, como uma planilha ou banco de dados relacional. Esses dados podem estar em várias formas e não seguem um padrão predefinido, tornando-os mais desafiadores de serem analisados e processados automaticamente.

O processamento de dados não estruturados é um campo em crescimento que envolve o uso de técnicas avançadas de processamento de linguagem natural, visão computacional, aprendizado de máquina e outras abordagens para extrair informações valiosas desses tipos variados de dados. Essas informações podem ser usadas em uma variedade de aplicações, desde análise de negócios até pesquisa científica e desenvolvimento de produtos.

6. Qual o nome da biblioteca responsável pela extração/captura de dados disponíveis em arquivos HTML ou XML? Explique.

Biblioteca BeautifulSoup. O BeautifulSoup é uma biblioteca poderosa que facilita a análise e a extração de informações de documentos HTML e XML de uma maneira fácil de usar.

7. Os arquivos do tipo XML (eXtensible Markup Language) surgiram como forma de estruturação e troca de dados pela internet. Dentre suas principais característica preencha os seguintes questionamentos:

a) Sintaxe inicial na primeira linha do arquivo.xml

A sintaxe inicial de um arquivo XML deve começar com a declaração XML, que é uma linha especial que informa ao analisador XML sobre a versão do XML sendo usada e a codificação de caracteres.

b) Os dados são organizados em formato hierárquico ou tabular?

A escolha entre o formato hierárquico e o formato tabular depende da natureza dos dados e das necessidades de processamento e análise. Cada formato tem suas vantagens e desvantagens, e a escolha adequada depende do contexto específico em que os dados serão usados.

c) Quais são as formas de representação de um documento XML. Justique.

Um documento XML (Extensible Markup Language) pode ser representado de várias formas, dependendo do contexto e da finalidade. As formas de representação mais comuns incluem:

Texto Puro (Plain Text): Esta é a forma mais básica de representação XML, em que o documento é escrito como texto simples em um arquivo. Os elementos XML são marcados com tags, e os atributos são definidos dentro dessas tags.

Formato de Árvore: XML pode ser representado como uma estrutura de árvore, onde os elementos são organizados hierarquicamente. Cada elemento é representado como um nó na árvore, com os elementos filhos conectados aos elementos pais. Esta é uma representação interna frequentemente usada por programas e bibliotecas que manipulam XML.

JSON (JavaScript Object Notation): XML pode ser convertido em JSON para facilitar a integração com sistemas que usam JSON como formato de dados. Cada elemento XML é representado como um objeto JSON, e os atributos são convertidos em campos de

objeto.

Banco de Dados XML: Em sistemas de gerenciamento de bancos de dados, os documentos XML podem ser armazenados e consultados usando estruturas específicas de banco de dados XML, como XML DB ou coleções XML em bancos de dados NoSQL.

XML Minificado: Em casos onde o tamanho do arquivo é crítico, o XML pode ser minificado, removendo espaços em branco, quebras de linha e outras formatações não essenciais. Isso torna o documento mais compacto, mas menos legível para os humanos.

A forma de representação escolhida depende das necessidades do aplicativo ou sistema que está lidando com o XML. Em muitos casos, uma representação em texto puro é usada para armazenar e trocar dados, enquanto representações em árvore ou objetos em memória são usadas para manipulação e processamento. JSON é frequentemente usado quando a interoperabilidade com sistemas que usam JSON é necessária.

8. Elabore um documento xml sobre produtos disponíveis para venda em empresas do comércio eletrônico/móveis/imóveis/roupas, a partir das seguintes condições: ▪ O produto deve possuir 5 características;
   - Cada produto deve ter um nome de identificação;
   - No documento deverá ter pelo menos dois produtos preenchidos.

```
<Produtos>
  <!-- Produto de Comércio Eletrônico 1 -->
  <Produto>
    <Categoria>Comércio Eletrônico</Categoria>
    <Nome>Smartphone</Nome>
    <Caracteristicas>
      <Caracteristica>Marca: Apple</Caracteristica>
      <Caracteristica>Modelo: iPhone 12</Caracteristica>
      <Caracteristica>Cor: Preto</Caracteristica>
      <Caracteristica>Tamanho da Tela: 6.1 polegadas</Caracteristica>
      <Caracteristica>Armazenamento: 128GB</Caracteristica>
    </Caracteristicas>
  </Produto>

  <!-- Produto de Comércio Eletrônico 2 -->
  <Produto>
    <Categoria>Comércio Eletrônico</Categoria>
    <Nome>Tablet</Nome>
    <Caracteristicas>
      <Caracteristica>Marca: Samsung</Caracteristica>
      <Caracteristica>Modelo: Galaxy Tab S7</Caracteristica>
      <Caracteristica>Cor: Prata</Caracteristica>
      <Caracteristica>Tamanho da Tela: 11 polegadas</Caracteristica>
```

```
      <Caracteristica>Armazenamento: 256GB</Caracteristica>
    </Caracteristicas>
</Produto>

<!-- Produto de Móveis 1 -->
<Produto>
    <Categoria>Móveis</Categoria>
    <Nome>Sofá</Nome>
    <Caracteristicas>
      <Caracteristica>Estilo: Moderno</Caracteristica>
      <Caracteristica>Cor: Cinza</Caracteristica>
      <Caracteristica>Material: Tecido</Caracteristica>
      <Caracteristica>Dimensões: 80 x 32 x 35 polegadas</Caracteristica>
      <Caracteristica>Peso: 100 libras</Caracteristica>
    </Caracteristicas>
</Produto>

<!-- Produto de Móveis 2 -->
<Produto>
    <Categoria>Móveis</Categoria>
    <Nome>Cama King-size</Nome>
    <Caracteristicas>
      <Caracteristica>Estilo: Clássico</Caracteristica>
      <Caracteristica>Cor: Marrom</Caracteristica>
      <Caracteristica>Material: Madeira de Carvalho</Caracteristica>
      <Caracteristica>Dimensões: 76 x 80 polegadas</Caracteristica>
      <Caracteristica>Peso: 150 libras</Caracteristica>
    </Caracteristicas>
</Produto>

<!-- Produto de Imóveis 1 -->
<Produto>
    <Categoria>Imóveis</Categoria>
    <Nome>Apartamento</Nome>
    <Caracteristicas>
      <Caracteristica>Tipo: Apartamento de 2 quartos</Caracteristica>
      <Caracteristica>Localização: Centro da Cidade</Caracteristica>
      <Caracteristica>Área: 1000 pés quadrados</Caracteristica>
      <Caracteristica>Andar: 10º andar</Caracteristica>
      <Caracteristica>Preço: R$ 500.000,00</Caracteristica>
    </Caracteristicas>
</Produto>

<!-- Produto de Imóveis 2 -->
<Produto>
    <Categoria>Imóveis</Categoria>
    <Nome>Casa</Nome>
    <Caracteristicas>
```

```
            <Caracteristica>Tipo: Casa de 3 quartos</Caracteristica>
            <Caracteristica>Localização: Subúrbio</Caracteristica>
            <Caracteristica>Área do Terreno: 5000 pés quadrados</Caracteristica>
            <Caracteristica>Vagas de Garagem: 2</Caracteristica>
            <Caracteristica>Preço: R$ 750.000,00</Caracteristica>
        </Caracteristicas>
    </Produto>

    <!-- Produto de Roupas 1 -->
    <Produto>
        <Categoria>Roupas</Categoria>
        <Nome>Jaqueta de Couro</Nome>
        <Caracteristicas>
            <Caracteristica>Estilo: Casual</Caracteristica>
            <Caracteristica>Cor: Marrom</Caracteristica>
            <Caracteristica>Tamanho: Médio</Caracteristica>
            <Caracteristica>Material: Couro genuíno</Caracteristica>
            <Caracteristica>Marca: FashionCo</Caracteristica>
        </Caracteristicas>
    </Produto>

    <!-- Produto de Roupas 2 -->
    <Produto>
        <Categoria>Roupas</Categoria>
        <Nome>Vestido de Noite</Nome>
        <Caracteristicas>
            <Caracteristica>Estilo: Elegante</Caracteristica>
            <Caracteristica>Cor: Preto</Caracteristica>
            <Caracteristica>Tamanho: Pequeno</Caracteristica>
            <Caracteristica>Material: Seda</Caracteristica>
            <Caracteristica>Marca: GlamourDress</Caracteristica>
        </Caracteristicas>
    </Produto>
</Produtos>
```

9. Defina o que é um documento JSON e quais suas principais características.

Um documento JSON (JavaScript Object Notation) é um formato de dados leve e de fácil leitura que é amplamente utilizado para a troca de informações entre sistemas e para o armazenamento de dados estruturados. JSON é uma representação textual de dados que segue um conjunto de regras simples e é independente de linguagem, o que o torna muito versátil. Suas principais características incluem:

Formato de Texto: JSON é representado como texto legível pelo humano, o que o torna fácil

de criar e analisar manualmente e também facilita a depuração. Ele utiliza uma sintaxe simples baseada em pares chave-valor.

Estrutura Hierárquica: Os dados em um documento JSON são organizados em uma estrutura hierárquica de objetos e arrays. Os objetos contêm pares chave-valor e os arrays contêm uma lista ordenada de valores.

Tipos de Dados: JSON suporta tipos de dados comuns, incluindo strings, números, booleanos, objetos, arrays e null. Isso o torna adequado para representar uma variedade de tipos de dados.

Chave-Valor: Os dados são armazenados como pares chave-valor, em que uma chave é uma string que identifica exclusivamente um valor associado. As chaves são usadas para acessar os valores em um objeto JSON.

Aninhamento: JSON permite a aninhamento de objetos e arrays, o que significa que você pode ter objetos dentro de objetos ou arrays dentro de arrays, criando estruturas de dados complexas.

Universalidade: JSON é independente de linguagem, o que significa que pode ser facilmente utilizado em uma variedade de linguagens de programação. A maioria das linguagens modernas oferece suporte nativo ou bibliotecas para trabalhar com JSON.

Leitura e Escrita: JSON é fácil de ser lido e gerado por máquinas e é amplamente utilizado em serviços web para a troca de dados, como APIs RESTful, devido à sua facilidade de processamento.

Compatibilidade: JSON é amplamente suportado em várias plataformas, tornando-o uma escolha popular para a comunicação entre sistemas heterogêneos.

Portabilidade: Dados em formato JSON podem ser facilmente transportados entre diferentes sistemas e ambientes, sem a necessidade de conversões complexas.

Em resumo, um documento JSON é uma representação de dados estruturados em um formato de texto simples e fácil de ler, com suporte para objetos, arrays, tipos de dados comuns e independente de linguagem, tornando-o uma escolha popular para a troca de dados na programação e comunicação entre sistemas.

10.O que significa o processo de serialização (JSON.stringify) e desserialização (JSON.parse) de documentos do tipo JSON?

A serialização e desserialização de documentos JSON são processos que envolvem a conversão de dados em JavaScript para o formato JSON (JavaScript Object Notation) e vice-versa. Isso é útil quando você deseja transmitir dados entre diferentes sistemas ou armazenar dados em um formato que possa ser facilmente recuperado e manipulado posteriormente.

A serialização é o processo de converter uma estrutura de dados JavaScript em uma sequência de caracteres no formato JSON.

Isso é útil quando você deseja transmitir dados de um aplicativo para outro através da rede, armazenar dados em um arquivo ou em um banco de dados, ou até mesmo quando precisa enviar dados de um cliente da web para um servidor.

A desserialização é o processo inverso, onde você converte uma string JSON de volta para um objeto ou estrutura de dados JavaScript.

Isso é útil quando você recebe dados em formato JSON e deseja trabalhar com eles em seu aplicativo, convertendo-os em objetos JavaScript para processamento posterior.

11. Faça um exemplo de documento JSON a partir de dados sobre serviços de vendas online.
    - Utilize dados do tipo, string, inteiro, array e objetos.

```
{
 "nomeEmpresa": "Loja Online",
 "fundadaEm": 2020,
 "categoria": "Roupas e Acessórios",
 "site": "www.lojaonline.com",
 "endereço": {
  "rua": "Rua da Loja, 123",
  "cidade": "Cidade Online",
  "estado": "Estado Virtual",
  "cep": "12345-678"
 },
 "produtos": [
  {
   "id": 1,
   "nome": "Camisa básica",
   "preço": 29.99,
   "estoque": 150
  },
```

```
    {
      "id": 2,
      "nome": "Calça Jeans",
      "preço": 25.99,
      "estoque": 100
    },
    {
      "id": 3,
      "nome": "Sapatos de Couro",
      "preço": 79.99,
      "estoque": 75
    }
  ],
  "avaliações": [
    {
      "cliente": "ClienteSatisfeito123",
      "classificação": 5,
      "comentário": "Excelente loja, entrega rápida e produtos de alta qualidade!"
    },
    {
      "cliente": "CompradorOnline456",
      "classificação": 4,
      "comentário": "Bons preços, mas o atendimento poderia ser melhor."
    }
  ]
}
```


12. Quais são as principais diferenças entre documentos do tipo JSON e XML.

JSON (JavaScript Object Notation) e XML (Extensible Markup Language) são dois formatos de dados amplamente utilizados para representar informações estruturadas. Ambos têm suas vantagens e desvantagens, e a escolha entre eles depende das necessidades específicas do projeto. Aqui estão algumas das principais diferenças entre JSON e XML:

Sintaxe:

JSON: Usa uma sintaxe mais simples e compacta, representando dados como pares chave-valor. Os valores podem ser números, strings, booleanos, objetos JSON aninhados, arrays e nulos.

XML: Utiliza uma sintaxe baseada em tags e atributos. Os dados são encapsulados em elementos XML, que podem conter atributos e serem aninhados.

Leitura humana:

JSON: Mais fácil de ler e escrever para humanos devido à sua sintaxe simples e concisa.

XML: Pode ser mais verboso e complexo para leitura humana devido à estrutura de tags e atributos.

Tamanho do arquivo:

JSON: Normalmente, arquivos JSON tendem a ser menores e mais eficientes em termos de espaço em disco devido à sua sintaxe mais compacta.

XML: Arquivos XML tendem a ser maiores devido à sintaxe mais verbosa.

Desempenho:

JSON: Geralmente é mais eficiente para análise por máquina, pois a análise de JSON é mais rápida e simples.

XML: A análise de XML pode ser mais demorada e exigir mais recursos devido à complexidade da sintaxe.

Tipagem de dados:

JSON: Oferece tipagem de dados mais simples e direta, com suporte para tipos básicos (números, strings, booleanos) e estruturas aninhadas.

XML: Permite definir tipos de dados personalizados usando Document Type Definitions (DTDs) ou Schemas XML, o que oferece uma flexibilidade maior, mas também pode aumentar a complexidade.

Metadados:

JSON: Geralmente não possui um sistema integrado de metadados. Os metadados precisam ser incorporados no próprio JSON ou ser gerenciados separadamente.

XML: Pode incluir metadados diretamente nos elementos XML usando atributos ou elementos específicos para esse fim.

Uso:

JSON: Amplamente utilizado em aplicativos web modernos, APIs RESTful e sistemas que exigem alta eficiência na transmissão de dados.

XML: Ainda é comum em sistemas legados, padrões de intercâmbio de dados como SOAP e configurações de documentos estruturados, como documentos XML.

Em resumo, a escolha entre JSON e XML depende das necessidades do seu projeto e das preferências da sua equipe de desenvolvimento. JSON é geralmente mais adequado para representar dados simples e eficientes em termos de espaço, enquanto XML oferece uma

flexibilidade maior, mas com uma sintaxe mais complexa.

13. Para que serve utilizar JDBC com Sistemas de Gerenciamento de Banco de Dados.

Aqui estão algumas das principais finalidades e benefícios de utilizar JDBC com SGBDs:

Conexão com o Banco de Dados: JDBC fornece classes e métodos para estabelecer conexões com um SGBD a partir de um aplicativo Java. Isso permite que os aplicativos acessem e recuperem dados armazenados no banco de dados.

Execução de Consultas SQL: Com JDBC, você pode executar consultas SQL no banco de dados a partir do seu aplicativo Java. Isso possibilita a recuperação de dados, a inserção, atualização e exclusão de registros no banco de dados.

Transações: JDBC oferece suporte a transações, permitindo que você crie transações atômicas que garantem a integridade dos dados. Você pode iniciar, confirmar ou reverter transações usando JDBC.

Gerenciamento de Conexão: JDBC ajuda no gerenciamento de conexões com o banco de dados. Ele permite que você crie, reutilize e feche conexões de forma adequada, evitando vazamento de recursos.

Acesso a Diferentes SGBDs: JDBC é uma API genérica e pode ser usada com uma variedade de SGBDs, incluindo Oracle, MySQL, PostgreSQL, SQL Server e outros. Isso significa que você pode desenvolver aplicativos Java que sejam compatíveis com diferentes SGBDs sem grandes modificações no código.

Segurança: JDBC oferece recursos de segurança que permitem a autenticação e a autorização para acessar o banco de dados. Isso ajuda a proteger os dados sensíveis armazenados no banco.

Desempenho: Usar JDBC eficientemente pode melhorar o desempenho do seu aplicativo ao otimizar consultas SQL, minimizar o tráfego de rede e utilizar recursos como o pool de conexões.

Flexibilidade: JDBC oferece flexibilidade para criar aplicativos personalizados que atendam às necessidades específicas do seu projeto. Você não está limitado por estruturas ou paradigmas específicos.

Integração com Java: Como JDBC é uma API Java, ela se integra perfeitamente com outras tecnologias Java, como Servlets, JSP (JavaServer Pages), frameworks de desenvolvimento web, entre outros.

Acesso a Bancos de Dados Legados: JDBC também pode ser usado para acessar sistemas de bancos de dados legados que não oferecem drivers nativos para outras linguagens de programação.

Em resumo, JDBC é uma ferramenta essencial para desenvolvedores Java que desejam interagir com SGBDs de maneira eficiente e flexível. Ele simplifica a tarefa de acessar e manipular dados em bancos de dados relacionais e desempenha um papel fundamental no desenvolvimento de aplicativos empresariais que dependem de armazenamento de dados em banco de dados.

14. Quais são os principais componentes durante a implementação do JDBC? Explique.

Os principais componentes e etapas envolvidos na implementação do JDBC:

Driver JDBC: O driver JDBC é um componente fundamental. Ele fornece a interface para se conectar a um banco de dados específico. Existem quatro tipos de drivers JDBC: Tipo 1 (Bridge), Tipo 2 (Nativo), Tipo 3 (Network Protocol) e Tipo 4 (Thin/Database Protocol). A escolha do driver depende do banco de dados que você está usando.

Conexão com o Banco de Dados: Você precisa criar uma conexão com o banco de dados usando o driver JDBC. Isso envolve a criação de um objeto de conexão usando a classe Connection. A conexão pode ser configurada com informações como URL do banco de dados, nome de usuário e senha.

Statement: O objeto Statement é usado para enviar comandos SQL para o banco de dados. Existem três tipos principais de statements: Statement, PreparedStatement e CallableStatement. O Statement básico é usado para consultas simples, enquanto o PreparedStatement é usado para consultas parametrizadas e o CallableStatement é usado para chamar procedimentos armazenados.

Execução de Consultas: Você pode usar o Statement para executar consultas SQL no banco de dados. As consultas podem ser de leitura (SELECT) ou modificação (INSERT, UPDATE, DELETE).

ResultSet: Quando você executa uma consulta SELECT, o resultado é armazenado em um objeto ResultSet. Você pode iterar sobre o ResultSet para recuperar os dados retornados pela consulta.

Gerenciamento de Transações: O JDBC permite o gerenciamento explícito de transações. Você pode iniciar, confirmar ou reverter transações usando métodos da conexão (commit() e rollback()).

Tratamento de Exceções: É importante lidar com exceções ao usar o JDBC. Erros de conexão, erros de consulta e outros problemas podem ocorrer, e você deve estar preparado para tratá-los adequadamente.

Encerramento de Recursos: Após a conclusão das operações, é importante fechar todos os recursos abertos, como conexões, statements e result sets, para liberar os recursos do sistema e evitar vazamentos de recursos.

Pool de Conexões (Opcional): Em aplicativos do mundo real, é comum usar um pool de conexões para melhorar o desempenho e a eficiência da comunicação com o banco de dados. Isso envolve o uso de bibliotecas de pool de conexões, como o Apache Commons DBCP ou C3P0.

Configuração de Propriedades (Opcional): Você pode configurar várias propriedades do JDBC, como o tamanho do cache do resultado, tempo limite de conexão e outras configurações específicas do driver.

15. Cite restrições sobre a utilização do JDBC para sistemas atuais.

Complexidade da Codificação: O JDBC é uma API de baixo nível, o que significa que requer uma quantidade considerável de código para configurar a conexão com o banco de dados, gerenciar exceções e fechar recursos adequadamente. Isso pode tornar o código complexo e sujeito a erros, especialmente em sistemas grandes e complexos.

Desempenho: O desempenho do JDBC pode ser afetado negativamente se não for usado corretamente. Por exemplo, não fechar conexões de banco de dados adequadamente pode levar a vazamentos de recursos e problemas de desempenho.

Segurança: A segurança é uma preocupação importante ao usar o JDBC. Se não for implementada corretamente, a aplicação pode estar vulnerável a ataques de injeção SQL. É importante usar instruções preparadas ou consultas parametrizadas para proteger contra esse tipo de ataque.

Compatibilidade com Banco de Dados: Embora o JDBC seja uma API padrão, a compatibilidade entre os drivers JDBC e diferentes sistemas de gerenciamento de banco de dados pode variar. Certifique-se de que o driver JDBC que você está usando seja compatível com o banco de dados que deseja acessar.

Gerenciamento de Conexões: O gerenciamento de conexões de banco de dados é uma tarefa crítica ao usar o JDBC. É importante garantir que as conexões sejam abertas apenas quando necessário e fechadas quando não forem mais necessárias. Isso pode ser complicado de implementar corretamente.

Escalabilidade: O JDBC pode ser menos escalável em comparação com outras tecnologias de acesso a dados, especialmente em cenários com grande volume de tráfego. É importante dimensionar adequadamente o pool de conexões para evitar problemas de desempenho.

Limitações da Linguagem SQL: O JDBC é uma API de baixo nível que não oferece recursos avançados de mapeamento objeto-relacional (ORM) ou facilidades para consultas complexas. Isso significa que a escrita de consultas SQL personalizadas pode ser necessária em muitos casos.