

**TEMA:**

- Consistência de Dados

TURMA:

- Técnico em Desenvolvimento de Sistemas – ENT
- Professor Juliano Silva

INTEGRANTES:

- Filipe Juliano
- Gabriel Nunes
- Isabella Godinho
- Lucas Vinicius de Oliveira
- Ryan Yago
- Sarah Cristina
- Vitor Pinheiro

Sumário

Introdução.....	4
Importância da Consistência de Dados em Sistemas de Software	4
Objetivos da Apresentação	4
Fundamentos da Consistência de Dados.....	5
O que é Consistência de Dados?	5
Exemplos de problemas de inconsistência de dados:	5
Transações Parcialmente Completas:	6
Dados Obsoletos:.....	6
Validação de Dados	6
Métodos Comuns de Validação de Dados:.....	6
Tipos de Consistência.....	8
Consistência Forte	8
Consistência Eventual.....	8
Tipos de Validações de Dados	9
1. Validação de Formato	9
3. Validação de Tipo de Dado	9
4. Validação de Unicidade.....	9
5. Validação de Consistência de Referência	9
6. Validação de Negócio	9
7. Validação de Presença	10
8. Validação de Tamanho.....	10
Técnicas para Garantir Consistência de Dados	11
Definir Prazo:	11
Verifique a Fonte de Dados:.....	11
Definir as Regras de Dados:	11
Executar a Transformação de Dados:	11
Comparar e Reconciliar Dados:.....	12
Testar e Verificar Dados:.....	12
Controle de Alterações:.....	12
Backup e Recuperação:	13
FERRAMENTAS E PRÁTICAS	14
1. Sistemas de Gerenciamento de Bancos de Dados (SGBD)	14
2. Ferramentas de Administração de Bancos de Dados	15

3. Soluções de Backup e Restauração.....	16
4. Ferramentas de Modelagem de Dados.....	17
5. Ferramentas de ETL (Extract, Transform, Load)	17
6. Ferramentas de Monitoramento e Desempenho.....	18
7. Plataformas de Business Intelligence (BI)	19
ESTUDO DE CASO E EXEMPLO PRÁTICO.....	21
Declaração de variável que exibirá o menu:	21
Inicialização das demais variáveis:.....	22
Estruturação do código e opção um; depósito:	22
Opção dois; saque:	23
Opção 3; extrato:.....	23
Opção zero, sair:.....	24
Validação de Dados: Importância da Estrutura de Condicional e Tratamento de Exceções:.....	24
Exemplos SQL:	25
Principais Desafios na Garantia da Consistência de Dados.....	27
Conclusão.....	29
Importância da Consistência de Dados:	29
Principais Técnicas e Ferramentas Discutidas:	29
Recapitulação	30
Recursos Adicionais.....	31
Links para documentação de utilizadas:	31
Artigos e livros recomendados sobre consistência de dados e validação de dados: 31	
Tutoriais e cursos online para aprofundamento:	31

Introdução

Vamos explorar um tema crucial para qualquer sistema de software: a **importância da consistência de dados**. Nos dias atuais, com a crescente quantidade de dados sendo gerados e processados, garantir que esses dados sejam consistentes é essencial para a integridade e confiabilidade dos sistemas.

Importância da Consistência de Dados em Sistemas de Software

A consistência de dados se refere à garantia de que os dados em um sistema sejam precisos, completos e válidos em todos os momentos. Isso é fundamental porque:

1. **Tomada de Decisão:** Decisões baseadas em dados inconsistentes podem levar a conclusões erradas e ações equivocadas.
2. **Confiabilidade:** Sistemas que não garantem a consistência de dados perdem a confiança dos usuários e stakeholders.
3. **Segurança:** Dados inconsistentes podem ser um indicativo de falhas de segurança ou ataques maliciosos.
4. **Eficiência Operacional:** Processos de negócios dependem de dados consistentes para funcionar corretamente e evitar retrabalho.

Objetivos da Apresentação

1. **Entender o que é Consistência de Dados:** Definir e explicar o conceito de consistência de dados em sistemas de software.
2. **Explorar Métodos de Garantia de Consistência:** Discutir as técnicas e práticas comuns para garantir a consistência dos dados.
3. **Importância da Validação de Dados:** Entender como a validação de dados contribui para a consistência e quais são as melhores práticas.
4. **Estudos de Caso:** Analisar exemplos reais onde a consistência de dados fez a diferença.

Fundamentos da Consistência de Dados

O que é Consistência de Dados?

Definição de Consistência de Dados: Consistência de dados refere-se à garantia de que os dados em um banco de dados estão corretos, completos e confiáveis. Em um sistema de banco de dados, consistência significa que, após qualquer operação (inserção, atualização, exclusão), os dados permanecem em um estado válido e as regras de integridade e restrições definidas no banco de dados são mantidas. Isso implica que todas as transações devem levar o banco de dados de um estado consistente para outro estado consistente, sem deixar dados corrompidos ou incompletos.

Exemplos de problemas de inconsistência de dados:

1. Registros Duplicados:

- *Exemplo:* Um sistema de CRM onde o mesmo cliente é registrado várias vezes com pequenas variações no nome ou endereço. Isso pode resultar em múltiplas entradas para o mesmo cliente, dificultando o acompanhamento preciso de suas interações e histórico de compras.

2. Referências Perdidas:

- *Exemplo:* Em um sistema de gerenciamento de inventário, um produto pode ser referenciado em uma tabela de pedidos, mas se a entrada correspondente na tabela de produtos for excluída sem atualização na tabela de pedidos, isso resultaria em uma referência perdida. Pedidos podem referenciar produtos que não existem mais.

3. Dados Contraditórios:

- *Exemplo:* Em um sistema bancário, se uma transferência de dinheiro é registrada como completa na conta do remetente mas não aparece na conta do destinatário, isso causa uma inconsistência. Um exemplo mais simples seria um sistema onde

um saldo bancário mostra valores diferentes em diferentes telas ou relatórios.

Transações Parcialmente Completas:

- *Exemplo:* Durante uma transação de reserva de passagem aérea, o pagamento pode ser processado mas o assento não ser reservado devido a uma falha de sistema. Isso deixaria os dados em um estado inconsistente, onde o cliente foi cobrado mas não tem uma passagem reservada.
-

Dados Obsoletos:

- *Exemplo:* Em um sistema de e-commerce, a quantidade em estoque de um produto pode não ser atualizada corretamente após uma venda devido a uma falha na comunicação entre os módulos de venda e inventário, resultando em informações de estoque desatualizadas e possivelmente permitindo vendas de produtos que não estão mais disponíveis.

Validação de Dados

Definição de Validação de Dados: Validação de dados é o processo de garantir que os dados inseridos em um sistema estejam corretos, precisos e em conformidade com as regras e restrições definidas. Isso é feito antes que os dados sejam armazenados no banco de dados ou processados, para prevenir problemas de inconsistência e garantir a integridade dos dados.

Métodos Comuns de Validação de Dados:

1. Validação de Tipo:

- Verifica se os dados inseridos são do tipo correto (por exemplo, números, strings, datas).

2. Validação de Formato:

- Assegura que os dados estejam no formato correto (por exemplo, um endereço de e-mail válido, um número de telefone com o formato correto).

3. Validação de Intervalo:

- Garante que os dados estejam dentro de um intervalo aceitável (por exemplo, uma nota entre 0 e 100).

4. Validação de Consistência:

- Verifica se os dados são consistentes com outras partes do sistema (por exemplo, a data de término não pode ser anterior à data de início).

5. Validação de Unicidade:

- Garante que não haja duplicação de registros onde não deveria haver (por exemplo, IDs únicos, números de CPF).

6. Validação de Integridade Referencial:

- Assegura que as chaves estrangeiras estejam corretamente associadas às chaves primárias correspondentes (por exemplo, um pedido deve referenciar um cliente válido). A validação de dados é uma etapa crítica para manter a consistência e a integridade do banco de dados, prevenindo a entrada de dados inválidos ou incorretos e evitando problemas futuros.



Tipos de Consistência

Consistência Forte

A consistência forte, também conhecida como consistência imediata, garante que todas as réplicas de um dado vejam a mesma versão desse dado ao mesmo tempo. Isso significa que assim que uma atualização é feita, todas as leituras subsequentes retornarão a versão mais recente do dado. É o modelo de consistência mais intuitivo, mas pode ser difícil de alcançar em sistemas distribuídos devido à latência de rede e à possibilidade de falhas.

- **Vantagens:**
 - Simplifica o desenvolvimento de aplicações, pois os desenvolvedores não precisam se preocupar com versões conflitantes dos dados.
 - Garante a precisão dos dados em tempo real.
- **Desvantagens:**
 - Pode afetar o desempenho, especialmente em sistemas distribuídos geograficamente dispersos.
 - Maior latência nas operações de escrita e leitura.

Consistência Eventual

A consistência eventual é um modelo mais relaxado onde o sistema garante que, em algum momento, todas as réplicas de um dado estarão consistentes, mas não necessariamente imediatamente após uma atualização. Esse tipo de consistência é comum em sistemas distribuídos de grande escala, como redes de servidores em data centers diferentes.

- **Vantagens:**
 - Melhora o desempenho e a disponibilidade do sistema.
 - Reduz a latência nas operações de leitura e escrita.
- **Desvantagens:**
 - Pode ser mais complexo para os desenvolvedores gerenciarem conflitos de dados e garantirem a precisão dos dados em tempo real.

- Introduz uma janela de tempo em que leituras podem retornar dados desatualizados.

Tipos de Validações de Dados

1. Validação de Formato

Garante que os dados inseridos seguem um formato específico. Por exemplo, um campo de email deve conter um "@" e um domínio válido.

2. Validação de Range (Intervalo)

Assegura que os dados estejam dentro de um intervalo específico. Por exemplo, a idade de uma pessoa deve estar entre 0 e 120 anos.

3. Validação de Tipo de Dado

Confirma que os dados são do tipo esperado, como inteiros, strings, datas, etc.

4. Validação de Unicidade

Garante que os dados são únicos no contexto do sistema. Por exemplo, um número de identificação único para cada usuário.

5. Validação de Consistência de Referência

Verifica que as referências entre dados estão corretas, como garantir que uma chave estrangeira em um banco de dados relacional existe na tabela de referência.

6. Validação de Negócio

Aplica regras específicas de negócio aos dados. Por exemplo, o saldo de uma conta bancária não pode ser negativo.

7. Validação de Presença

Assegura que certos campos não podem estar vazios. Por exemplo, um formulário de cadastro deve ter o campo de nome preenchido.

8. Validação de Tamanho

Verifica que o tamanho dos dados está dentro de um limite aceitável, como o número máximo de caracteres em uma string.

Implementar essas validações em diferentes camadas do sistema, como na interface do usuário, na lógica de aplicação e no banco de dados, ajuda a garantir a integridade e a consistência dos dados ao longo de todo o ciclo de vida do sistema.

Técnicas para Garantir Consistência de Dados

Definir Prazo:

- Não ser tímido ao dar a opinião sobre o código que está desenvolvendo e o andamento do projeto, se esforçar e não ceder à pressão e arriscar a criar códigos confusos que trarão problemas futuros. Prestar atenção nos detalhes. Um tratamento de erro reduzido é apenas uma das maneiras pela qual os programadores deixam de notar os detalhes.

Verifique a Fonte de Dados:

- Verificar a fonte de dados e entender como os dados foram coletados, armazenados e processados. Isso pode ajudá-lo a identificar quaisquer fontes potenciais de inconsistência, como valores ausentes, valores atípicos, duplicatas, problemas de formatação ou erros humanos. Você também pode verificar a fonte de dados com o proprietário ou provedor de dados e solicitar qualquer documentação ou metadados que possam ajudá-lo a entender melhor os dados.

Definir as Regras de Dados:

- Definir as regras de dados que especificam o formato, o intervalo, o tipo e a qualidade esperados dos valores de dados. As regras de dados podem ajudá-lo a definir os padrões e critérios de consistência de dados e detectar quaisquer desvios ou violações deles. Você pode usar regras de dados para executar a validação de dados, onde você verifica se seus valores de dados estão em conformidade com as regras de dados e sinaliza ou relata quaisquer erros ou inconsistências.

Executar a Transformação de Dados:

- Executar a transformação de dados, onde você modifica ou manipula seus valores de dados para torná-los consistentes com



suas regras de dados e metas de análise. A transformação de dados pode envolver várias operações, como converter tipos de dados, padronizar unidades, formatar datas, substituir valores ausentes, remover valores atípicos, mesclar ou dividir colunas ou aplicar funções matemáticas. Você pode usar várias ferramentas e linguagens, como Python, para executar a transformação de dados.

Comparar e Reconciliar Dados:

- Comparar e reconciliar dados, onde você verifica se seus valores de dados são consistentes em diferentes fontes, conjuntos de dados ou versões. A comparação e a reconciliação de dados podem ajudá-lo a identificar e resolver quaisquer discrepâncias ou conflitos em seus dados, como registros incompatíveis, rótulos inconsistentes ou valores contraditórios. Você pode usar vários métodos e ferramentas, como inspeção visual, estatísticas de resumo, tabelas dinâmicas ou algoritmos de correspondência de dados, para comparar e reconciliar dados.

Testar e Verificar Dados:

- Testar e verificar dados, onde você avalia e confirma os resultados e resultados do seu processo de limpeza de dados. O teste e a verificação de dados podem ajudá-lo a garantir a consistência dos dados, validando a precisão, a confiabilidade e a qualidade dos dados limpos. Você pode usar várias técnicas e ferramentas, como criação de perfil de dados, visualização de dados, estatísticas descritivas ou teste de hipóteses, para testar e verificar dados.

Controle de Alterações:

- São mecanismos que controlam o acesso concorrente aos dados. Os bloqueios garantem que apenas uma operação possa acessar e modificar os dados em um determinado momento, evitando assim conflitos e inconsistências.

- Garanta que as atualizações de software do sistema sejam feitas para cumprir os regulamentos em constante mudança, especialmente na implementação de novos recursos. Colabore com os fornecedores para se manter informado sobre as mudanças e atualizar seus sistemas de acordo.
- Selecione sistemas que sejam fáceis de atualizar após a adição de um novo hardware ou outras entradas do sistema.

Backup e Recuperação:

- Implementar sistemas robustos de backup e recuperação para proteger os dados contra perdas e corrupções. A replicação de dados também pode ser usada para garantir a consistência. A replicação envolve a criação de cópias dos dados em vários locais, de modo que, se um local falhar, os dados ainda estejam disponíveis em outros locais. No entanto, a replicação de dados também apresenta desafios, como a sincronização e a resolução de conflitos entre as cópias dos dados.

FERRAMENTAS E PRÁTICAS

Ferramentas de bancos de dados são essenciais para a gestão eficiente de dados em diferentes contextos, desde pequenos projetos até grandes corporações. Elas ajudam na criação, manipulação, manutenção e otimização dos bancos de dados. Aqui estão algumas das principais categorias e exemplos de ferramentas de bancos de dados:

1. Sistemas de Gerenciamento de Bancos de Dados (SGBD)

MySQL

- Recursos: Suporta múltiplas tabelas, índices, transações, integridade referencial e triggers.
- Casos de Uso: Ampla utilização em aplicações web, e-commerce e sistemas de gerenciamento de conteúdo (CMS) como o WordPress.

PostgreSQL

- Recursos: Oferece suporte a tipos de dados avançados (JSON, XML), índices GIN e GIST, funções definidas pelo usuário e replicação.
- Casos de Uso: Indicado para aplicações que demandam conformidade com os padrões SQL e robustez, como sistemas financeiros, aplicações GIS e plataformas de análise de dados.

SQLite

- Recursos: Totalmente embutido (biblioteca em C), sem necessidade de configuração, com o banco de dados armazenado em um único arquivo.
- Casos de Uso: Aplicativos móveis, sistemas embarcados, prototipagem e pequenos projetos.

Microsoft SQL Server

- Recursos: Conta com segurança avançada, integração com outras ferramentas Microsoft e suporte a transações distribuídas.



- Casos de Uso: Indicado para aplicações corporativas, ERP, CRM e sistemas de data warehousing.

Banco de Dados Oracle

- Recursos: Suporte a grandes volumes de dados, recursos de alta disponibilidade, particionamento, e segurança avançada.
- Casos de Uso: Grandes corporações, bancos, telecomunicações, e aplicações que requerem alta disponibilidade e escalabilidade.

2. Ferramentas de Administração de Bancos de Dados

phpMyAdmin

- Recursos: Interface web para administração do MySQL, execução de queries, backup e restauração de bancos de dados.
- Aplicações: Gerenciamento de bancos de dados MySQL/MariaDB, sobretudo para aplicações web.

Toad for Oracle

- Recursos: Desenvolvimento e otimização de SQL, administração do Oracle, criação de relatórios.
- Aplicações: Administradores de bancos de dados Oracle, desenvolvedores que necessitam de ferramentas avançadas de SQL.

SQLyog

- Recursos: Administração e desenvolvimento para MySQL, análise de desempenho, sincronização de schemas.
- Aplicações: Desenvolvedores e DBAs que trabalham com MySQL em ambientes de produção. Cenários de Utilização: Gestão e desenvolvimento de bases de dados em ambientes diversos.

DBeaver

- Recursos: Suporte a vários SGBDs, interface gráfica para execução de queries, design de schemas.

3. Soluções de Backup e Restauração

Bacula

- Características: Realização de backups centralizados, recuperação de desastres, suporte a múltiplos tipos de mídia.
- Cenários de Utilização: Implementações extensas de backup, ambientes corporativos com diversos servidores.

Amanda

- Características: Automatização de backups, recuperação de dados, compatibilidade com várias plataformas.
- Cenários de Utilização: Empresas que demandam uma solução de backup escalável e de código aberto.

Veeam

- Características: Backup e recuperação para ambientes virtuais, físicos e na nuvem, replicação de dados.
- Cenários de Utilização: Empresas que utilizam virtualização, necessitam de alta disponibilidade e recuperação rápida de dados.

Oracle RMAN

- Características: Backup e recuperação específicos para Oracle, suporte a backups incrementais e completos.
- Cenários de Utilização: Administração de grandes bases de dados Oracle, situações de recuperação de desastres.

4. Ferramentas de Modelagem de Dados

ER/Studio

- Recursos: Modelagem conceitual, lógica e física, geração de scripts SQL, engenharia reversa.
- Aplicações: Desenvolvimento de novos bancos de dados, integração de sistemas e modernização de aplicações legadas.

IBM InfoSphere Data Architect

- Recursos: Modelagem de dados empresariais, integração com outras ferramentas IBM, suporte a diversos Sistemas de Gerenciamento de Banco de Dados (SGBDs).
- Aplicações: Empresas que precisam de uma solução integrada para modelagem de dados e arquitetura de informação.

MySQL Workbench

- Recursos: Design visual de esquemas de banco de dados, administração de servidores MySQL, consultas SQL.
- Aplicações: Desenvolvedores e administradores de bancos de dados que trabalham com MySQL.

pgAdmin

- Recursos: Interface visual para administração de PostgreSQL, execução de consultas, design de esquemas.
- Utilização: Administração e manutenção de servidores PostgreSQL, tanto para desenvolvedores quanto para DBAs.

5. Ferramentas de ETL (Extract, Transform, Load)

Apache Nifi

- Recursos: Automação de fluxo de dados, suporte a diversas fontes de dados, transformações em tempo real.



- Utilização: Integração de dados em tempo real, ingestão de dados para data lakes, pipelines de dados complexos.

Talend

- Recursos: Interface visual para design de processos ETL, suporte a big data, integração com várias fontes de dados.
- Utilização: Projetos de integração de dados, preparação de dados para análise, migração de dados.

Microsoft SSIS

- Recursos: Pacotes de integração de dados, transformação de dados, tarefas de automação.
- Utilização: Empresas que utilizam a stack Microsoft, integração de dados entre diferentes sistemas.

Informática PowerCenter

- Recursos: Transformação e limpeza de dados, suporte a big data, integração com ferramentas de BI.
- Aplicações: Implementações extensas de ETL, cenários complexos de integração de dados, soluções empresariais.

6. Ferramentas de Monitoramento e Desempenho

Nagios

- Recursos: Monitorização de servidores, redes e aplicações, alertas em tempo real.
- Utilizações: Monitorização da infraestrutura de TI, deteção precoce de problemas de desempenho.

Zabbix

- Recursos: Monitorização de redes e servidores, recolha de métricas, alertas configuráveis.

- Utilizações: Ambientes corporativos, monitorização de extensas redes e sistemas distribuídos.

New Relic

- Recursos: Monitorização do desempenho de aplicações, análise de transações, alertas.
- Utilizações: Monitorização de aplicações web, otimização de desempenho, análise da experiência do usuário.

SolarWinds Database Performance Analyzer

- Recursos: Monitorização do desempenho de bases de dados, análise de queries, alertas.
- Utilizações: Administração de bases de dados, otimização de desempenho, deteção de gargalos.

7. Plataformas de Business Intelligence (BI)

Tableau

- Recursos: Desenvolvimento de dashboards interativos, análise visual de dados, integração com múltiplas fontes de dados.
- Aplicações: Análise de informações empresariais, elaboração de relatórios e dashboards, visualização de dados complexos.

Power BI

- Recursos: Conexão com diversas fontes de dados, criação de relatórios e dashboards, análises preditivas.
- Aplicações: Empresas que utilizam a linha de produtos Microsoft, necessidade de análises de dados acessíveis e interativas.

QlikView

- Recursos: Análise de dados associativos, desenvolvimento de dashboards, integração com várias fontes de dados.

- Aplicações: Empresas que requerem análises de dados ágeis e intuitivas, elaboração de relatórios personalizados.

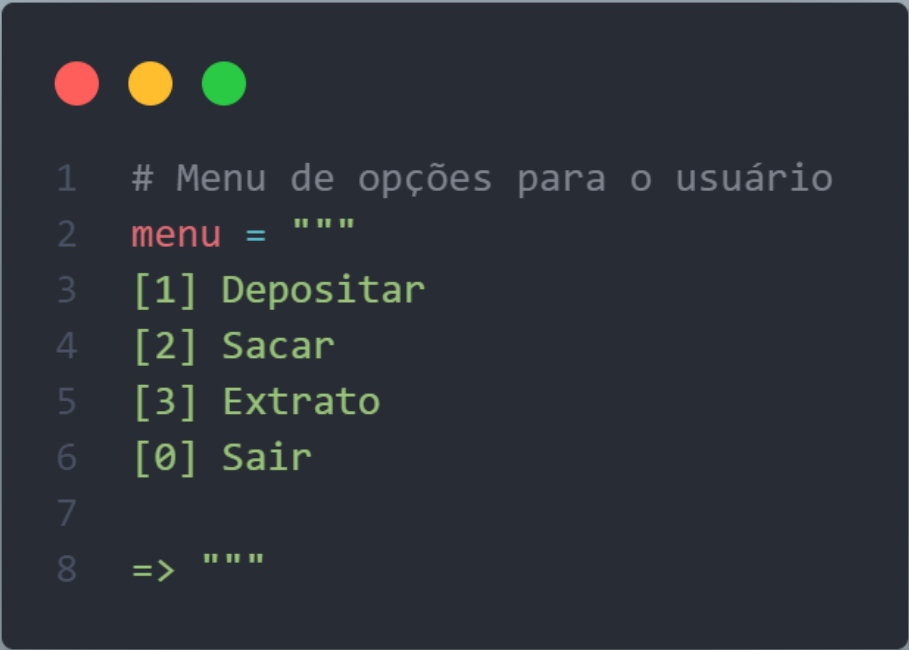
A escolha das ferramentas de bancos de dados deve ser baseada nas necessidades específicas do projeto e da organização. As ferramentas mencionadas acima cobrem uma ampla gama de funcionalidades e casos de uso, desde a administração básica até a análise avançada de dados e integração complexa. A seleção cuidadosa das ferramentas corretas pode resultar em um gerenciamento de dados mais eficiente, maior desempenho e melhores insights para a tomada de decisões.

ESTUDO DE CASO E EXEMPLO PRÁTICO

Para estudo de caso, criamos um código na linguagem Python. Mostraremos também a sintaxe básica da criação de tabelas de um banco de dados e de consulta, a fim de verificar a integridade dos dados. No exemplo, utilizamos SQL.

Trata-se de um código simples, simulando transações bancárias em que oferece na tela do usuário quatro opções de ações, sendo elas: 1 – Depositar, 2 – Sacar, 3 – Consultar o Extrato, 0 – Sair.

Declaração de variável que exibirá o menu:



```
1  # Menu de opções para o usuário
2  menu = """
3  [1] Depositar
4  [2] Sacar
5  [3] Extrato
6  [0] Sair
7
8  => """
```

Veja que na inicialização da variável menu, já guardamos nela os valores do menu que será visualizado na tela do usuário ao chamarmos a função print(). Este é um exemplo de código limpo, pois facilita a manutenção do código em eventuais alterações futuras.

Inicialização das demais variáveis:

```
1 # Inicialização o restante das variáveis das variáveis
2 saldo = 0 # Saldo inicial da conta
3 limite = 500 # Limite máximo para saques
4 extrato = "" # Armazena as transações realizadas
5 numero_saques = 0 # Contador de saques realizados
6 LIMITE_SAQUES = 3 # Número máximo de saques permitidos
```

Aqui iniciamos o restante das variáveis. Note que todos os nomes das variáveis são claros, com o intuito de ser legível quanto ao tipo de dado que esta irá armazenar. Veja que nessas variáveis já deixamos claros alguns dados relativos à regra do negócio: O limite para saques, que é de 500, e o limite de números de saques diários que o usuário pode fazer, que é 3.

Estruturação do código e opção um; depósito:

```
1 # Loop infinito para exibir o menu até que o usuário escolha sair
2 while True:
3     try:
4         # Solicita que o usuário selecione uma opção do menu
5         opcao = int(input(menu))
6
7         # Opção 1: Depositar
8         if opcao == 1:
9             valor = float(input("Informe o valor do depósito: ")) # Solicita o valor do depósito
10
11             # Verifica se o valor do depósito é positivo
12             if valor > 0:
13                 saldo += valor # Adiciona o valor ao saldo
14                 extrato += f"Depósito: R$ {valor:.2f}\n" # Adiciona a transação ao extrato
15             else:
16                 print("Operação falhou! O valor informado é inválido.") # Mensagem de erro para valor inválido
```

Em seguida, iniciamos o nosso loop (estrutura de repetição), solicitamos a opção que o usuário deseja e já começamos a trabalhar com nossas condicionais. A condicional em questão trata a opção 1 de operação, de depósito de valores. O código verifica se o valor inserido pelo usuário é positivo e adiciona esse valor à variável saldo, assim como cria um registro dessa transação, que ficará disponível na variável extrato.

Opção dois; saque:

```
1 # Opção 2: Sacar
2 elif opcao == 2:
3     valor = float(input("Informe o valor do saque: ")) # Solicita o valor do saque
4
5     # Verificações de condições para saque
6     excedeu_saldo = valor > saldo # Verifica se o valor do saque excede o saldo
7     excedeu_limite = valor > limite # Verifica se o valor do saque excede o limite permitido
8     excedeu_saques = numero_saques >= LIMITE_SAQUES # Verifica se o número de saques já atingiu o limite
9
10    # Verifica cada condição e exibe mensagens de erro correspondentes
11    if excedeu_saldo:
12        print("Operação falhou! Você não tem saldo suficiente.")
13    elif excedeu_limite:
14        print("Operação falhou! O valor do saque excede o limite.")
15    elif excedeu_saques:
16        print("Operação falhou! Número máximo de saques excedido.")
17    elif valor > 0:
18        saldo -= valor # Deduz o valor do saque do saldo
19        extrato += f"Saque: R$ {valor:.2f}\n" # Adiciona a transação ao extrato
20        numero_saques += 1 # Incrementa o contador de saques
21    else:
22        print("Operação falhou! O valor informado é inválido.") # Mensagem de erro para valor inválido
```

Na opção 2, o usuário realiza a operação de saque. Nela, temos que tratar alguns dados fundamentais: verificar se o valor do saque excede o saldo, se o valor do saque excede o limite permitido e se o usuário ainda tem saques diários permitidos.

Opção 3; extrato:

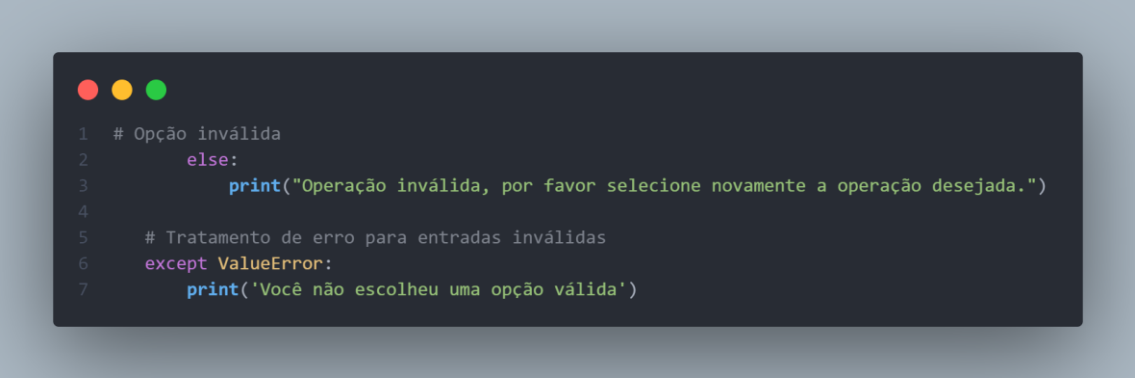
```
1 # Opção 3: Extrato
2 elif opcao == 3:
3     # Exibe o extrato e o saldo atual
4     print("\n===== EXTRATO =====")
5     print("Não foram realizadas movimentações." if not extrato else extrato)
6     print(f"\nSaldo: R$ {saldo:.2f}")
7     print("=====")
```

É uma opção de suma importância, onde o usuário irá conferir todas as suas transações realizadas. Uma inconsistência de dados nessa operação pode comprometer todo o sistema, pois deixaria em dúvida a confiabilidade de todos os outros dados.

Opção zero, sair:

O usuário realizou suas transações e deseja encerrar o programa.

Validação de Dados: Importância da Estrutura de Condicional e Tratamento de Exceções:



```
1 # Opção inválida
2     else:
3         print("Operação inválida, por favor selecione novamente a operação desejada.")
4
5 # Tratamento de erro para entradas inválidas
6 except ValueError:
7     print('Você não escolheu uma opção válida')
```

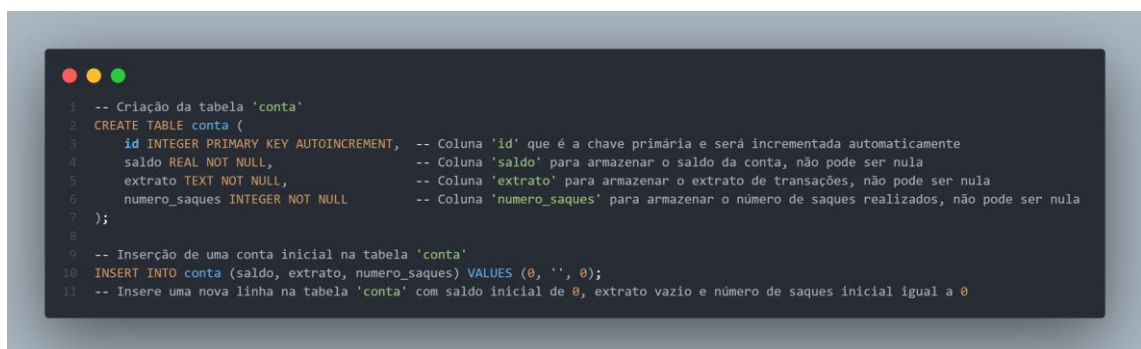
No final das opções de transações do nosso código, é crucial incluir mecanismos robustos de validação de dados. A parte do bloco que contém a estrutura `else` para opções inválidas e o uso de `try-except` são fundamentais para garantir a robustez e a confiabilidade do sistema.

- **Estrutura `else` para opções inválidas:** Esta condicional é essencial para lidar com casos em que o usuário escolhe uma opção que não está prevista no menu. Ao fornecer uma resposta apropriada, como uma mensagem de erro ou um aviso para escolher uma opção válida, isso melhora significativamente a experiência do usuário e evita comportamentos inesperados que poderiam comprometer a integridade do sistema.
- **`try-except` para tratamento de exceções:** O uso de `try-except` é crucial para lidar com entradas de dados que não estão no formato esperado. Por exemplo, se o usuário digitar um valor não numérico onde é esperado um número (como na opção de depósito ou saque), o bloco `try-except` pode capturar essa exceção e fornecer uma resposta adequada, como pedir ao usuário para inserir um valor válido. Isso previne falhas no programa e mantém a estabilidade durante a interação do usuário com o sistema.

Implementar essas práticas não apenas melhora a usabilidade e confiabilidade do código, mas também fortalece a segurança e evita potenciais falhas que poderiam surgir de entradas inesperadas ou incorretas por parte do usuário.

Exemplos SQL:

Agora, vamos ver o exemplo da sintaxe da criação de tabelas de um banco de dados relacional para a persistência desses dados:

A screenshot of a code editor with a dark background and light-colored text. The code is a SQL script for creating a table named 'conta' and inserting an initial record. The script includes comments in Portuguese explaining the purpose of each column and the insertion. The code is as follows:

```
1 -- Criação da tabela 'conta'
2 CREATE TABLE conta (
3     id INTEGER PRIMARY KEY AUTOINCREMENT, -- Coluna 'id' que é a chave primária e será incrementada automaticamente
4     saldo REAL NOT NULL,                  -- Coluna 'saldo' para armazenar o saldo da conta, não pode ser nula
5     extrato TEXT NOT NULL,                 -- Coluna 'extrato' para armazenar o extrato de transações, não pode ser nula
6     numero_saques INTEGER NOT NULL        -- Coluna 'numero_saques' para armazenar o número de saques realizados, não pode ser nula
7 );
8
9 -- Inserção de uma conta inicial na tabela 'conta'
10 INSERT INTO conta (saldo, extrato, numero_saques) VALUES (0, '', 0);
11 -- Insere uma nova linha na tabela 'conta' com saldo inicial de 0, extrato vazio e número de saques inicial igual a 0
```

No código acima, criamos a tabela conta. Nela, definimos um id como a chave primária da tabela, com incremento automático, garantindo que cada conta tenha um identificador único. Fizemos isso como boa prática para garantir a consistência dos dados de cada usuário. Em seguida, usamos em nossas colunas o NOT NULL e especificamos o tipo de dado que será inserido (DECIMAL, TEXT, INT). Trata-se também de uma técnica de consistência de dados, garantindo que a coluna não receberá um valor nulo e que o tipo de dado estará correto.

Podemos verificar manualmente se as operações foram bem-sucedidas executando a seguinte consulta SQL:



```
1  SELECT * FROM conta WHERE id = 1;  
2
```

Isso retornará os valores atuais do saldo, extrato e número de saques da conta, permitindo confirmar que as operações foram registradas corretamente no banco de dados.

Principais Desafios na Garantia da Consistência de Dados

A consistência de dados é crucial para sistemas que manipulam informações. No entanto, garantir essa consistência em ambientes distribuídos e de alta concorrência apresenta desafios significativos. Alguns dos principais desafios incluem:

1. **Latência e Sincronização:** Em sistemas distribuídos, os dados podem estar distribuídos geograficamente em diferentes nós. Manter a consistência entre esses nós requer sincronização eficiente, considerando que a latência na comunicação entre os nós pode afetar a consistência dos dados.
2. **Modelos de Consistência:** Existem diversos modelos como "strong consistency", "eventual consistency" e "causal consistency". A escolha do modelo adequado é crucial, pois cada um possui trade-offs em termos de desempenho e garantias de consistência.
3. **Conflitos e Resolução:** Concorrentemente, múltiplas operações podem tentar modificar os mesmos dados, levando a conflitos. A resolução desses conflitos (por exemplo, usando vetores de relógio ou timestamps) é um desafio crítico.

O gerenciamento de dados em sistemas distribuídos envolve estratégias como:

1. **Particionamento:** Dividir os dados em partições distribuídas para escalabilidade e balanceamento de carga.
2. **Replicação:** Replicar dados em vários nós para redundância e disponibilidade. No entanto, isso aumenta a complexidade na manutenção da consistência entre as réplicas.
3. **Distribuição Transparente:** Os aplicativos devem acessar dados independentemente da localização física. Isso requer uma camada de abstração para gerenciar a distribuição de forma eficiente.

Em cenários de alta concorrência, como em sistemas de banco de dados ou serviços web, manter a consistência é crítico. Estratégias como bloqueio adequado, uso de transações e técnicas assíncronas são fundamentais:

1. **Bloqueio e Escalonamento:** Estratégias de bloqueio como locks são essenciais para evitar conflitos, mas um escalonamento adequado é necessário para evitar impactos negativos no desempenho.
2. **Transações:** Agrupar operações relacionadas em transações ajuda a manter a consistência. Definir níveis de isolamento apropriados garante que as operações sejam atomicamente consistentes.
3. **Técnicas Assíncronas:** Em vez de bloqueio síncrono, técnicas assíncronas como filas de mensagens podem minimizar a latência e melhorar o desempenho em cenários distribuídos.

Alguns exemplos de sistemas distribuídos incluem redes de computadores, sistemas de armazenamento em nuvem, redes peer-to-peer (P2P) e sistemas distribuídos de gerenciamento de banco de dados.

Para garantir a consistência de dados, é crucial adotar melhores práticas como:

1. **Escolha do Modelo de Consistência:** Selecionar o modelo adequado (strong, eventual, causal) considerando os requisitos do sistema.
2. **Transações e Isolamento:** Usar transações para garantir atomicidade e definir níveis de isolamento apropriados para evitar conflitos.
3. **Padrões de Acesso aos Dados:** Definir padrões claros para acessar e modificar dados, evitando atualizações concorrentes que possam causar inconsistências.

No design de banco de dados e na arquitetura de sistemas, considerar a escolha do banco de dados, estratégias de particionamento, replicação e gerenciamento de cache são fundamentais para suportar a consistência de dados.

Monitorar e auditar dados regularmente, além de usar ferramentas de teste e validação para simular cenários de concorrência, são práticas adicionais para assegurar a consistência em sistemas distribuídos complexos.

Conclusão

A consistência de dados é um pilar fundamental para a integridade e confiabilidade dos sistemas que manipulam informações. Ao longo deste estudo, exploramos os desafios significativos enfrentados na garantia da consistência, especialmente em ambientes distribuídos e de alta concorrência.

Importância da Consistência de Dados:

Garantir a consistência de dados é essencial para que os sistemas forneçam resultados confiáveis e precisos. Em um mundo onde a informação é uma moeda vital, inconsistências podem levar a decisões errôneas, falhas de segurança e perda de confiança dos usuários. A integridade dos dados é, portanto, crucial para sustentar operações eficientes e seguras.

Principais Técnicas e Ferramentas Discutidas:

Durante a exploração dos desafios da consistência de dados, discutimos diversas técnicas e ferramentas que ajudam a mitigar esses desafios:

- **Modelos de Consistência:** Escolher entre modelos como "strong consistency", "eventual consistency" e "causal consistency", dependendo dos requisitos do sistema.
- **Transações e Isolamento:** Utilizar transações para agrupar operações relacionadas e definir níveis adequados de isolamento para prevenir conflitos.
- **Técnicas Assíncronas:** Implementar técnicas assíncronas, como filas de mensagens, para minimizar a latência e melhorar o desempenho em sistemas distribuídos.
- **Particionamento e Replicação:** Dividir dados em partições distribuídas e replicar esses dados em múltiplos nós para escalabilidade e disponibilidade.
- **Monitoramento e Auditoria:** Implementar logs, trilhas de auditoria e métricas para detectar e corrigir inconsistências de dados de forma proativa.

- **Ferramentas de Testes:** Utilizar ferramentas de teste e validação para simular cenários de carga e verificar a robustez das estratégias de consistência adotadas.

Recapitulação

Em resumo, manter a consistência de dados é um desafio contínuo e multifacetado, especialmente em ambientes distribuídos e altamente concorrentes. A escolha do modelo de consistência adequado, a implementação de transações robustas e o uso eficiente de técnicas de sincronização são fundamentais para garantir que os sistemas operem de maneira confiável e eficiente.

Ao adotar as melhores práticas discutidas, os desenvolvedores e arquitetos de sistemas podem enfrentar os desafios da consistência de dados com maior confiança, assegurando que suas aplicações ofereçam resultados precisos e consistentes, independentemente das condições adversas ou da complexidade do ambiente distribuído.

Recursos Adicionais

Links para documentação de utilizadas:

- Documentação do SQL: <https://docs.microsoft.com/pt-br/sql/>
- Documentação do MySQL: <https://dev.mysql.com/doc/>
- Documentação do Python: <https://docs.python.org/3/>

Artigos e livros recomendados sobre consistência de dados e validação de dados:

- Martin, R. C. (2009). *Código Limpo: Habilidades Práticas do Agile Software*. Alta Books.
- Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson.

Tutoriais e cursos online para aprofundamento:

- Curso: "Banco de Dados SQL" da Fundação Bradesco, disponível em: <https://www.ev.org.br/cursos/banco-de-dados-sql>
- Curso: "SQL para Análise de Dados" da Data Science Academy, disponível em: <https://www.datascienceacademy.com.br/course/sql-para-analise-de-dados>
- Curso: "MySQL: Modelagem de Dados e Projeto de Banco" da Udemy, disponível em: <https://www.udemy.com/course/mysql-modelagem-de-dados-e-projeto-de-banco/>
- Curso: "Introdução à Programação com Python" do Curso em Vídeo, disponível em: <https://www.cursoemvideo.com/curso/python-3-mundo-1/>
- Tutorial: "Aprenda SQL" da Alura, disponível em: <https://www.alura.com.br/curso-online-sql>