ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 6

CQA - COMISSÃO DE QUALIFICAÇÃO E AVALIAÇÃO

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 6

Christiane Mazur Doi

Doutora em Engenharia Metalúrgica e de Materiais, Mestra em Ciências - Tecnologia Nuclear, Especialista em Cálculo e Matemática Aplicada, Especialista em Língua Portuguesa e Literatura, Especialista em Recursos Gramaticais para Revisão Textual, Engenheira Química e Licenciada em Matemática, com Aperfeiçoamento em Estatística e MBA em Engenharia Econômica. Professora titular da Universidade Paulista.

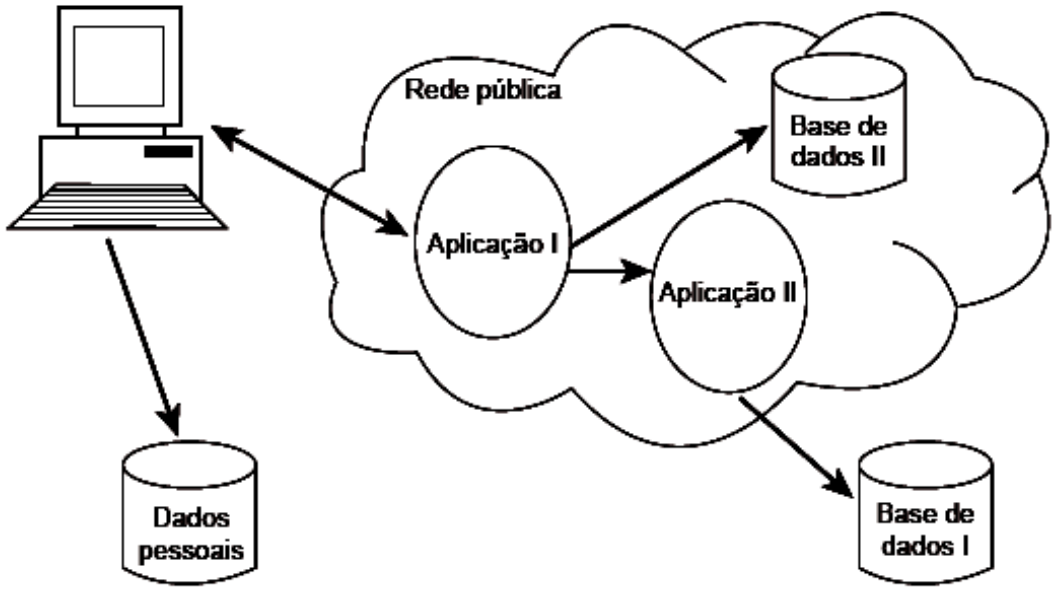
Tiago Guglielmeti Correale

Doutor e Mestre em Engenharia Elétrica e Engenheiro Elétrico - ênfase em Telecomunicações. Professor titular da Universidade Paulista.

Questão 1

Questão 1.[1]

Considere o arranjo computacional apresentado a seguir.



A característica fundamental esperada para tais sistemas de modo a ter o menor impacto sobre a experiência do usuário final é

A. a transparência entre as entidades do sistema.
B. a linguagem de programação orientada a eventos.
C. o hardware com elevada taxa de processamento de dados.
D. a base de dados deve estar localizada no mesmo espaço fixo.
E. a independência quanto à disponibilidade de conexão à rede de comunicação de dados.

1. Introdução teórica

Sistemas distribuídos

Até a década de 1970, computadores eram recursos caros, sofisticados e centralizados. O custo de produção de uma única máquina era tão elevado que, em geral, apenas governos ou grandes corporações tinham condições de adquirir esses equipamentos.

O uso dos computadores era feito por meio de lotes (*batches*), submetidos de uma só vez ao computador, usualmente na forma de um baralho de cartões perfurados. Ao final da execução, o usuário retirava os resultados em algum escaninho, ou seja, nenhum usuário

[1]Questão 15 – Enade 2014.

acessava o computador diretamente: todos os programas eram executados e controlados pelo "operador do sistema", que submetia os lotes ao computador central, controlava a sua execução, trazia os resultados impressos e colocava-os no escaninho público.

Com a evolução do uso dos sistemas em lotes, surgiram os sistemas interativos. Neles, o usuário acessa o computador e controla diretamente a execução do programa e de todos os periféricos ligados à máquina. Nas primeiras versões, os sistemas interativos eram monousuários, ou seja, apenas um único usuário podia utilizar o computador de cada vez.

Posteriormente, devido ao elevado custo dessas máquinas e à crescente demanda por serviços computacionais, surgiu o "compartilhamento temporal", ou, em inglês, o *time-sharing*. Nesse caso, uma única máquina era diretamente acessada por meio de um conjunto de terminais e os recursos eram distribuídos pelos diversos usuários, cada qual em seu terminal. Dessa forma, para cada usuário, parecia que havia um computador "só para ele". No entanto, na realidade, havia apenas um único computador rodando subdivisões independentes para cada usuário. Assim, uma máquina era capaz de atender a mais de um usuário ao mesmo tempo. Esses sistemas são chamados de sistemas multiusuários.

Com o aumento do número de máquinas e de usuários, houve a necessidade de interligar essas "ilhas de processamento" e surgiram as primeiras redes de computadores.

Com o desenvolvimento das redes de computadores e dos sistemas operacionais capazes de executar mais de um processo simultaneamente (sistemas operacionais multitarefa) e, também, com a capacidade de esses sistemas darem suporte a mais de um usuário simultaneamente (sistemas multiusuários), surgiu a possibilidade do compartilhamento de recursos tanto localmente quanto remotamente.

A comunicação entre computadores remotos possibilitou que sistemas fossem desenvolvidos de forma distribuída, tirando-se proveito não apenas dos recursos locais de uma única máquina, mas de várias máquinas remotas, postas em contato por meio da rede.

O resultado desse procedimento chama-se sistema distribuído, o qual é composto por recursos que executam, simultaneamente, em várias máquinas diferentes e se comunicam por meio da troca de mensagens, que são transportadas pela rede.

2. Análise das alternativas

A – Alternativa correta.

JUSTIFICATIVA. Em um sistema distribuído, no qual diferentes aplicações e bases de dados comunicam-se de forma complexa, é importante que existam interfaces bem definidas entre

as diversas entidades do sistema. Além disso, essas interfaces precisam ser transparentes, ou seja, devem permitir que as entidades se comuniquem sem expor a implementação interna. Dessa forma, a realização de alterações internas nessas entidades, como a correção de defeitos ou a execução de melhorias, não afetam a interface de acesso e fazem com que as aplicações que dependam dessas entidades continuem funcionando normalmente.

B – Alternativa incorreta.
JUSTIFICATIVA. Ainda que o uso de uma linguagem orientada a eventos seja útil na construção desse tipo de sistema, sua simples utilização não garante que o usuário final tenha a experiência adequada.

C – Alternativa incorreta.
JUSTIFICATIVA. Uma vez que as aplicações rodem em um sistema distribuído e se comuniquem por meio de uma rede pública, provavelmente há um ambiente bastante heterogêneo com relação ao hardware utilizado. Ainda há a influência da velocidade da rede pública, que está fora do controle da aplicação.

D – Alternativa incorreta.
JUSTIFICATIVA. A base de dados não precisa estar fisicamente próxima do local em que as aplicações são executadas, uma vez que elas podem ser acessadas remotamente pela rede.

E – Alternativa incorreta.
JUSTIFICATIVA. Um sistema distribuído, como o apresentado na figura do enunciado, certamente depende do acesso à rede para o seu funcionamento, uma vez que as aplicações rodam em diferentes máquinas e em diferentes locais, mas acessíveis por rede pública.

3. Indicações bibliográficas

- COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. *Distributed systems: concepts and design.* Harlow: Addison-Wesley, 2012.
- VERÍSSIMO, P.; RODRIGUES, L. *Distributed systems for system architects*. Boston: Kluwer Academic, 2001.

Questão 2

Questão 2.[2]

Para fins estatísticos, uma empresa precisa armazenar os trajetos que seus representantes comerciais percorrem entre pontos de venda. É importante que para cada local visitado sejam armazenadas, além da informação do próprio local, o local de origem do representante (ponto de venda anterior), o local de destino (ponto de venda posterior), as distâncias percorridas e os tempos de viagem. Esse procedimento permite que estes trajetos possam ser analisados, de forma rápida, do local de origem ao local de destino, bem como no sentido inverso, do local de destino (final do trajeto) ao local de origem (início do trajeto).

O analista responsável pelo sistema, que utilizará os dados armazenados e produzirá os relatórios estatísticos, projetou o seguinte esboço de uma classe que representa um ponto de venda:

```
public class Local {
     private String nome_estabelecimento;
     private String endereco;
     private Local origem;
     private Local destino;
     private float distancia_origem;
     private float tempo_origem;
}
```

A respeito do esboço da classe, avalie as afirmativas.

I. O esboço acima representa uma lista duplamente encadeada.

II. A utilização de um nó de uma estrutura de dados do tipo árvore de busca multivias de grau três seria a solução ideal para o problema porque providenciaria a economia de recursos de memória e de disco.

III. A utilização de uma árvore de pesquisa binária para a solução do problema é normal, desde que o atributo de ordenação da árvore seja `distancia_origem`.

É correto o que se afirma em

A. I, apenas.

B. II, apenas.

C. I e III, apenas.

D. II e III, apenas.

E. I, II e III.

[2]Questão 14 – Enade 2014.

1. Introdução teórica

Estruturas de dados: listas ligadas

A construção de um programa é profundamente influenciada pela escolha das estruturas de dados que dão suporte ao seu funcionamento. O livro intitulado *Algoritmos + Estruturas de Dados = Programas* (WIRTH, 1976) mostra como esses dois conceitos (algoritmos e estruturas de dados) estão relacionados e a importância deles para o trabalho do programador.

Com relação às estruturas de dados, podemos observar as listas (estruturas de dados lineares) e as árvores (estruturas de dados não lineares).

No seu formato mais simples, uma lista ligada pode ser vista como um conjunto de registros que apresentam ligações entre si, pelo menos em um sentido (também chamada de lista encadeada simples), como mostrado na figura 1.

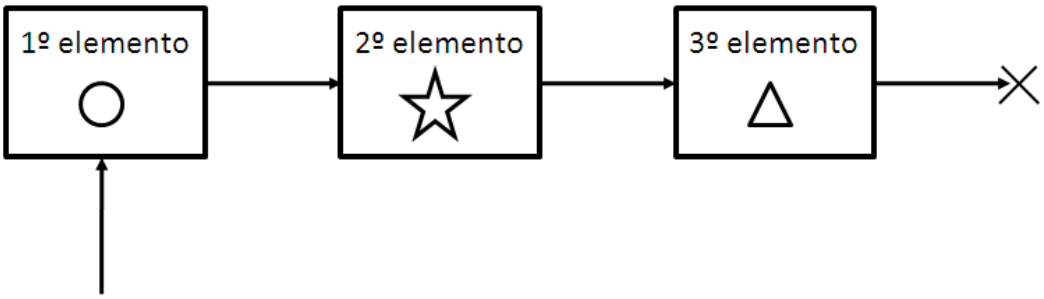


Figura 1. Representação gráfica de uma lista ligada simples com três elementos.

Cada membro da lista deve ter uma referência (ou ponteiro) para, pelo menos, outro elemento. Dessa forma, os elementos da lista estão "ligados" por referências. Isso permite que esses elementos sejam acessados de acordo com uma interface bem definida e de acordo com as referências.

O programador deve estar atento à forma como os elementos devem ser inseridos na lista: no início, no fim ou em algum ponto intermediário. Em alguns casos, pode ser necessário impor um critério de inserção, a fim de garantir que a lista apresente uma ordem particular.

Um exemplo bastante utilizado de lista é o chamado "lista duplamente ligada" (ou "lista duplamente encadeada"). Nesse caso, cada elemento da lista tem uma referência tanto para o elemento seguinte quanto para o elemento anterior, permitindo que o programa percorra a lista em dois sentidos. Caso ocorra a inserção de um elemento novo na

lista, o programador deve atualizar ambas as referências (exceto no caso de inserção nas extremidades).

Além das estruturas de dados lineares, como as listas ligadas, existem também estruturas de dados não lineares, como as árvores. Árvores são utilizadas para representar dados de natureza hierárquica.

Tipicamente, em uma estrutura de dados linear, cada elemento apresenta uma ligação para, no máximo, outros dois elementos (o elemento posterior e o elemento anterior). Dessa forma, ao percorrermos esse tipo de estrutura, obtemos um caminho tipicamente linear. Em estruturas de dados não lineares, podem existir várias opções diferentes de percursos. Uma das estruturas de dados não lineares mais importantes corresponde às árvores.

Existem diversos tipos de árvores, construídos com diferentes finalidades, como as árvores binárias e as árvores B. Em uma árvore binária, cada elemento tem, além de uma referência para o nó pai, até duas referências para os nós filhos.

Formalmente, uma árvore pode ser definida de forma recursiva como um conjunto finito *T* de um ou mais nós com as características a seguir.

- Existe um nó especial chamado de raiz da árvore.
- Os nós restantes (excluindo o nó raiz) são particionados em $m \geq 0$ conjuntos T_1, T_2, ..., T_m e cada um deles também é uma árvore. As árvores T_1, T_2, ..., T_m são chamadas de subárvores da raiz.

2. Análise das afirmativas

I - Afirmativa correta.

JUSTIFICATIVA. A estrutura apresentada tem duas referências para objetos da mesma classe (a classe Local). Logo, trata-se de uma lista duplamente encadeada.

II - Afirmativa incorreta.

JUSTIFICATIVA. Uma árvore de busca multivias (também chamada de árvore B) é uma forma de organização de árvore especialmente projetada para grandes volumes de dados, nos casos em que é impossível armazenar todos esses dados na memória principal do computador. Nesses casos, o computador tem de utilizar alguma forma de armazenamento secundário, que, normalmente, apresenta tempos de acesso muito mais elevados do que os da memória principal.

III - Afirmativa incorreta.

JUSTIFICATIVA. Não é necessária a utilização de uma árvore para a resolução desse problema, uma vez que uma lista duplamente encadeada resolve o problema de forma eficiente e correta.

Alternativa correta: A.

3. Indicação bibliográfica

- CELES, W.; CERQUEIRA, R.; RANGEL, J. R. *Introdução à Estrutura de Dados.* Rio de Janeiro: Campus, 2004.
- KNUTH, D. E. *The art of computer programming.* Upper Saddle River: Addison-Wesley, 2011, v. I.
- WIRTH, N. *Algorithms + Data Structures = Programs*. Englewood Cliffs: Prentice-Hall, 1976.

Questão 3

Questão 3.[3]

Leia o texto a seguir.

Uma função é denominada recursiva quando ela é chamada novamente dentro de seu corpo. Implementações recursivas tendem a ser menos eficientes, porém facilitam a codificação e seu entendimento.

CELES, W.; CERQUEIRA, R.; RANGEL, J. L. *Introdução a estrutura de dados*. Rio de Janeiro, 2004 (com adaptações).

Considere a função recursiva f(), a qual foi escrita em linguagem C:

```
1 int f( int v[], int n){
2     if(n == 0)
3        return 0;
4     else{
5        int s;
6        s = f(v, n-1);
7        if( v[n-1] > 0 ) s = s + v[n-1];
8            return s;
9     }
10 }
```

Suponha que a função f() é acionada com os seguintes parâmetros de entrada:

```
f({2,-4,7,0,-1,4},6);
```

Nesse caso, o valor de retorno da função f() será

A. 8.

B. 10.

C. 13.

D. 15.

E. 18.

1. Introdução teórica

Recursividade

Em um programa computacional, quando temos uma função (ou procedimento) chamando a si mesma, chamamos esse mecanismo de recursividade. À primeira vista, o fato de uma função chamar a si mesma pode parecer estranho e talvez até mesmo errado: se uma função chama a si mesma de forma contínua, quando o processo irá parar?

[3]Questão 16 – Enade 2014.

Ao criar uma função recursiva, o programador deve evitar situações em que o programa nunca termine, com uma função chamando a si mesma sem nunca se estabelecer um critério de parada. Dessa forma, deve existir uma condição na qual ocorra recursividade e outra condição na qual a função retorne algum valor.

Em um programa bem-comportado, sempre deve haver um momento em que o processo de recursividade é interrompido. A função retorna a um valor que vai ser utilizado em cada chamada anterior da função. Normalmente, queremos trabalhar com programas que devem levar um tempo finito para essa execução e, preferencialmente, o menor tempo possível.

Outro cuidado que devemos tomar ao utilizarmos recursividade, mesmo quando não temos uma situação com infinitas chamadas, é que, se o número de chamadas for muito grande, pode ocorrer uma situação chamada de estouro de pilha. Cada chamada para uma função implica a criação de um item a mais em uma região da memória chamada, a pilha de execução.

Se o número de elementos na pilha de execução crescer demasiadamente, a pilha pode estourar, levando ao fim da execução do programa. Esse problema não é exclusivo de programas que utilizam recursividade, mas é mais comum nesses casos, pois um número excessivo de chamadas a uma função pode levar ao estouro da pilha de execução.

Existem algumas técnicas de otimização que podem evitar situações de estouro de pilha. Uma das mais utilizadas é a chamada de recursividade final própria, ou, em inglês, *tail recursion*. Nesse caso, uma chamada pode ser feita sem a necessidade de se adicionar um quadro na pilha de chamadas, o que evita seu crescimento desenfreado.

2. Análise da questão

Para a resolução do problema, podemos construir uma tabela na qual simulamos a execução do programa. Observe que essa tabela será construída na ordem em que a função f(v,n) retorna aos valores, e não na ordem em que ela é chamada. Isso acontece porque essa é uma função recursiva e o primeiro valor a ser retornado é 0, na linha 3, quando n é igual a 0. A função f(v[0], 0) retorna a s=0 na linha 6. Como v[0]=2>0, sabemos que s=s+2=0+2=2. Sendo assim, f(v,1)=2. Esse valor é novamente retornado à linha 6. Com f(v[1],2), mas v[1]=-4<0, portanto, f(v,2)=2. Esse processo é repetido até que se chegue ao valor f(v,6)=13, conforme tabela 1.

Tabela 1. Resultado de f(v,n) para a execução do programa.

N	f(v,n)
0	0
1	2
2	2
3	9
4	9
5	9
6	13

Alternativa correta: C.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J.R. *Introdução à Estrutura de Dados.* Rio de Janeiro: Campus, 2004.
- CORMEN, T.; LEISERSON, C.; RIVEST, R.; STEIN, C. *Introduction to algorithms*. 3. ed. Cambridge: MIT Press, 2009.
- KNUTH, D. E. *The art of computer programming.* Uppers Saddle River: Addison Wesley, 2011, v. I.
- TOSCANI, L. V.; VELOSO, P.A.S. *Complexidade de algoritmos.* 2. ed. Porto Alegre: Bookman, 2008.

Questão 4

Questão 4.[4]

Nos anos de 1970, os sistemas executavam em mainframes com aplicativos escritos em linguagens estruturadas e com todas as funcionalidades em um único módulo, com grande quantidade de linhas de código. Acessos a bancos de dados não relacionais, regras de negócios e tratamento de telas para terminais "burros" ficavam no mesmo programa. Posteriormente, uma importante mudança ocorreu: a substituição dos terminais "burros" por microcomputadores, permitindo que todo o tratamento da interface, e de algumas regras de negócios, passassem a ser feitas nas estações clientes. Surgiam as aplicações cliente-servidor. A partir dos anos 1990 até os dias atuais, as mudanças foram mais radicais, os bancos de dados passaram a ser relacionais e distribuídos. As linguagens passaram a ser orientadas a objetos, cuja modelagem encapsula dados e oferece funcionalidades através de métodos. A interface passou a ser web. Vive-se a era das aplicações em três camadas.

Considerando a evolução da arquitetura de software de sistemas de informação, conforme citado no texto acima, avalie as seguintes afirmações:

I. A separação dos sistemas em três camadas lógicas torna os sistemas mais complexos, requerendo pessoal mais especializado.

II. A separação dos sistemas em três camadas lógicas torna os sistemas mais flexíveis, permitindo que as partes possam ser alteradas de forma independente.

III. A separação dos sistemas em três camadas lógicas aumentou o acoplamento, dificultando a manutenção.

É correto o que se afirma em

A. II, apenas.

B. III, apenas.

C. I e II, apenas.

D. I e III, apenas.

E. I, II e III.

[4]Questão 18 – Enade 2014.

1. Introdução teórica

Arquitetura de software

Uma das maiores dificuldades ao se criar um novo software é garantir uma arquitetura que seja suficientemente flexível para poder crescer e incorporar novas funcionalidades, conforme as necessidades do cliente, mas que também seja robusta, para permitir que esse crescimento possa ser feito com o menor esforço possível.

Essas duas necessidades (flexibilidade e organização), com frequência, caminham em direções aparentemente opostas. Na realidade, o problema é que é muito mais fácil fazer um programa crescer de forma desordenada do que crescer de forma ordenada. A falta de organização pode ser falsamente percebida como flexibilidade. O desafio do arquiteto de software está em encontrar uma arquitetura que englobe tanto flexibilidade quanto organização.

Havia pouco conhecimento sobre o desenvolvimento de programas grandes e complexos e não havia muita preocupação com aspectos arquitetônicos de um programa. Frequentemente, a postura era a de solucionar o problema da forma mais rápida possível, sem nenhuma preocupação com o futuro do programa. Isso levou à geração de uma grande quantidade de códigos "não gerenciáveis": códigos com má qualidade e com má organização, difíceis de manter e de modificar.

O aumento da demanda por sistemas computacionais e o crescente uso de computadores levaram ao surgimento da engenharia de software. Um de seus objetivos é sistematizar a produção do software, incluindo boas práticas no projeto e na codificação de programas. Um dos aspectos fundamentais dessa área é a elaboração de uma arquitetura de software que seja suficientemente flexível para permitir o crescimento futuro, mas também organizada e gerenciável, para permitir que o desenvolvimento seja previsível, especialmente do ponto de vista do esforço necessário para a introdução de novas características, com o menor número possível de defeitos.

Uma das arquiteturas mais difundidas para o desenvolvimento de sistemas computacionais é a chamada arquitetura em camadas. Nessa arquitetura, cada camada do sistema tem responsabilidades específicas e uma interface bem definida. Por responsabilidade, queremos dizer que cada camada apresenta um objetivo claro e bem delimitado. Por interface, queremos dizer que o acesso aos recursos da camada se faz por meio de métodos (funções ou procedimentos, em alguns casos) claramente especificados,

padronizados e estáveis. No entanto, os detalhes técnicos de cada camada devem estar suficientemente isolados, de forma que, para a utilização da interface, não seja necessário conhecer os detalhes internos da sua implementação.

Ao se criar uma arquitetura de software mais sofisticada, surge a necessidade de mão de obra mais qualificada, bem como de um processo de gerenciamento mais adequado. Costuma-se dizer que o preço que se paga para se ter uma complexidade gerenciável é o aumento da complexidade de cada módulo do sistema. Contudo, esse aumento local da complexidade é cuidadoso e planejado e tem como objetivo elevar a qualidade do software desenvolvido, o que torna o desenvolvimento mais previsível e o programa mais flexível.

2. Análise das afirmativas

I - Afirmativa correta.
JUSTIFICATIVA. A divisão de um sistema em camadas certamente implica um aumento da complexidade do código, que deve ter uma arquitetura cuidadosamente projetada para separar responsabilidades adequadas para cada uma das camadas. Contudo, quando devidamente gerenciado, esse aumento de complexidade pode ser compensado, em longo prazo, pela maior facilidade no gerenciamento do desenvolvimento do software e pela maior facilidade de manutenção.

II - Afirmativa correta.
JUSTIFICATIVA. Em um sistema dividido em camadas e com interfaces adequadas, a manutenção e a extensão são muito mais fáceis. Com o correto isolamento conceitual entre cada uma das camadas, alterações podem ser feitas de forma transparente em uma das camadas sem a necessidade de alterações em outras camadas.

III - Afirmativa incorreta.
JUSTIFICATIVA. O objetivo da separação em camadas é diminuir o acoplamento, e não o aumentar. Cada camada deve ser acessível por uma interface bem definida, com o objetivo de desacoplar as camadas e facilitar a extensão e a manutenção do código.

Alternativa correta: C.

3. Indicações bibliográficas

- BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software architecture in practice.* Boston: Addison-Wesley, 2012.
- FAIRBANKS, G. *Just enough software architecture: a risk-driven approach.* Boulder: Marshall & Brainerd, 2010.

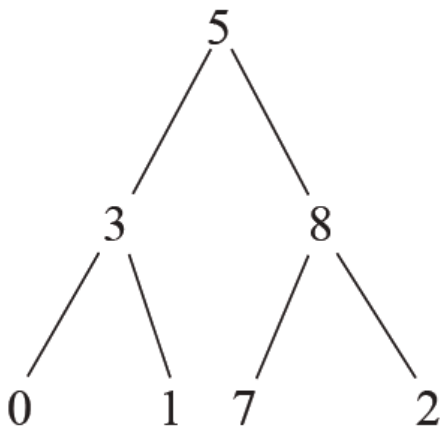
Questão 5

Questão 5.[5]

Existem várias maneiras de se percorrer uma árvore binária. A função a seguir, escrita em pseudocódigo, percorre uma árvore na ordem esquerda-raiz-direita, conhecida por varredura e-r-d recursiva. A função erd() recebe por parâmetro a raiz r de uma árvore e faz uso de seus elementos esq, dir e cont, que representam, respectivamente, ponteiros para uma subárvore à esquerda de r, uma subárvore à direita de r e o conteúdo de r.

```
função erd( árvore r)
{
     se( r != NULO )
     {
          erd( r->esq);
          escreva(r->conteúdo);
          erd(r->dir);
     }
}
```

Considere a árvore binária a seguir.



A sequência correta de exibição do conteúdo da árvore utilizando a função erd() é:

A. 5,3,8,0,1,7,2.

B. 0,1,7,2,3,8,5.

C. 0,3,5,1,7,8,2.

D. 0,3,1,5,7,8,2.

E. 2,7,8,5,0,3,1.

[5]Questão 20 – Enade 2014.

1. Introdução teórica

Árvores binárias

Listas ligadas, pilhas e vetores são estruturas de dados muito úteis para representação de vários tipos de informações em programas computacionais. Contudo, nem sempre conseguimos representar informações utilizando esses tipos de estruturas. Isso ocorre à medida que o relacionamento entre os nós começa a se tornar mais complexo, com mais possibilidades de interligação.

Uma das formas de estrutura de dados mais comuns para a representação de informações hierárquicas é a árvore. Na computação, uma árvore corresponde a uma estrutura que contém um nó raiz (desenhado no topo) e uma série de nós filhos (que correspondem aos ramos).

Existem vários tipos de árvores, cada uma com uma finalidade específica. Um dos tipos mais comuns e úteis é a chamada árvore binária. Nesse caso, cada nó pode ter zero, um ou dois nós filhos. Os últimos elementos da árvore, normalmente desenhados na sua porção inferior, são chamados de nós folhas.

Uma das operações importantes que sempre devemos ser capazes de executar em uma estrutura de dados é chamada de percurso. Por exemplo, frequentemente queremos percorrer uma lista, visitando cada um dos seus elementos. Ou então percorrer um vetor, lendo cada um dos seus elementos ou fazendo outra operação com esses elementos. Também podemos percorrer uma árvore binária visitando cada um dos seus elementos.

É conveniente utilizar funções recursivas para situações em que queremos percorrer estruturas de dados como árvores binárias e listas ligadas. Devido a essas estruturas terem referências (ou ponteiros) para outros elementos da mesma classe (ou tipo), a utilização de funções que seguem essas referências é bastante conveniente.

2. Análise da questão

Para resolvermos a questão, enumeramos três pontos importantes da função "erd", mostrados na figura 1. Devemos observar que um número só é impresso quando a função "escreva" é chamada. Quando r for NULO, a função "erd" retorna sem efetuar nenhuma ação. Caso contrário, a função "erd" é chamada de forma recursiva nos pontos 1 e 3.

```
função erd( árvore r)
{
       se( r != NULO )
       {
           (1) erd( r->esq);
           (2) escreva(r->conteúdo);
           (3) erd(r->dir);
       }
}
```

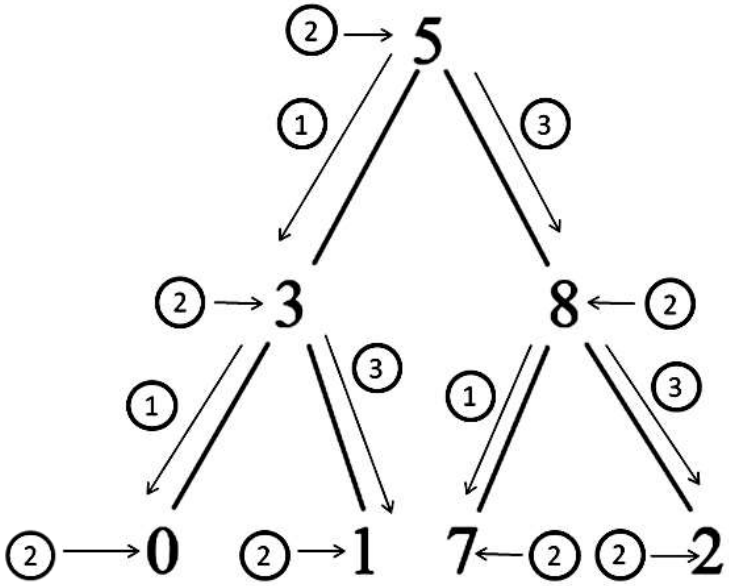


Figura 1. Representação gráfica de uma árvore binária ao lado da função "erd" utilizada para percorrer os elementos de uma árvore.

Para que a função "escreva" seja chamada, a função "erd" chamada no ponto 1 deve retornar, o que ocorre apenas quando r for igual a NULO. Logo, a primeira impressão deve ser a de um dos nós folhas da árvore. Nós folhas são os últimos nós de uma árvore computacional. Observe que, em computação, a árvore é desenhada "de cabeça para baixo", com a raiz no topo do desenho e as folhas na parte de baixo.

Na figura 2, temos o desenho da mesma árvore do enunciado, com a ordem de impressão dos elementos dentro de quadrados ao lado dos nós da árvore. Compare essa figura e a figura 1 e observe os sentidos das setas. O número dentro das circunferências na figura 1 mostra o ponto no código em que o programa toma determinada decisão. Observe que:

- no ponto 1, sempre percorremos o ramo esquerdo da árvore;
- no ponto 2, sempre imprimimos um elemento;
- no ponto 3, sempre percorremos o ramo direito.

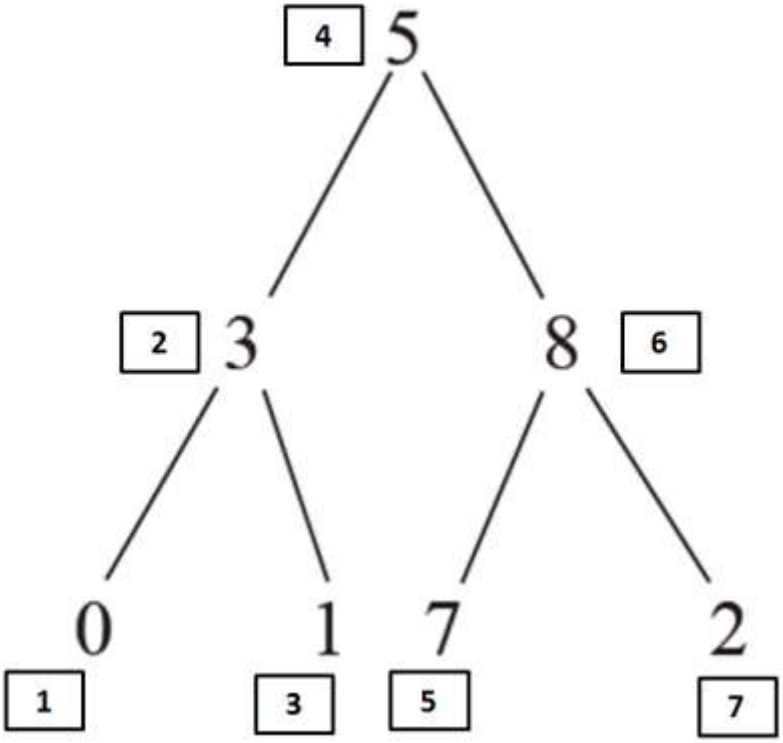


Figura 2. Ordem de impressão dos elementos da árvore binária.

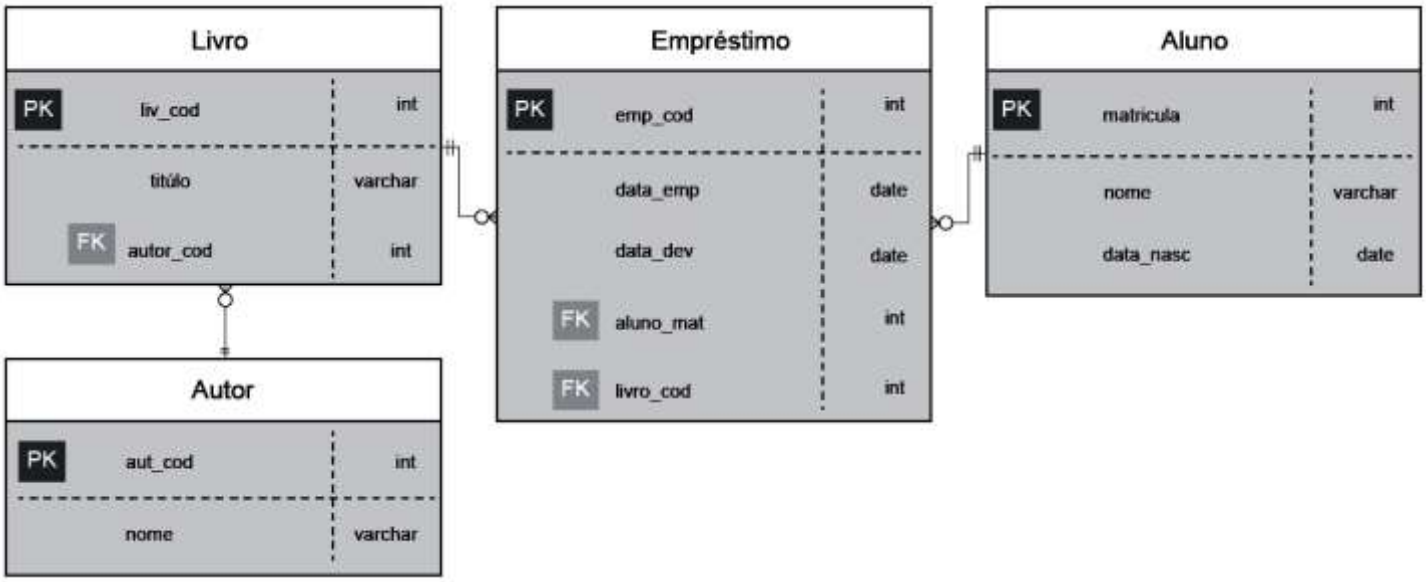
Alternativa correta: D.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J. R. *Introdução à estrutura de dados.* Rio de Janeiro: Campus, 2004.
- TUCKER, A. B.; NOONAN, R. E. *Linguagens de programação: princípios e paradigmas*. 2. ed. São Paulo: McGrawhill, 2008.

Questões 6, 7 e 8

Questão 6.[6]



O modelo de entidade relacionamento apresentado, representa, de forma sucinta, uma solução para persistência de dados de uma biblioteca. Considerando que um livro está emprestado quando possuir um registro vinculado a ele na tabela "Empréstimo", e essa tupla não possuir valor na coluna "data_dev", o comando SQL que deve ser utilizado para listar os títulos dos livros disponíveis para empréstimo é:

A.
```
select titulo from livro
except
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

B.
```
select titulo from livro
union
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

C.
```
select titulo from livro
except
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is not null
```

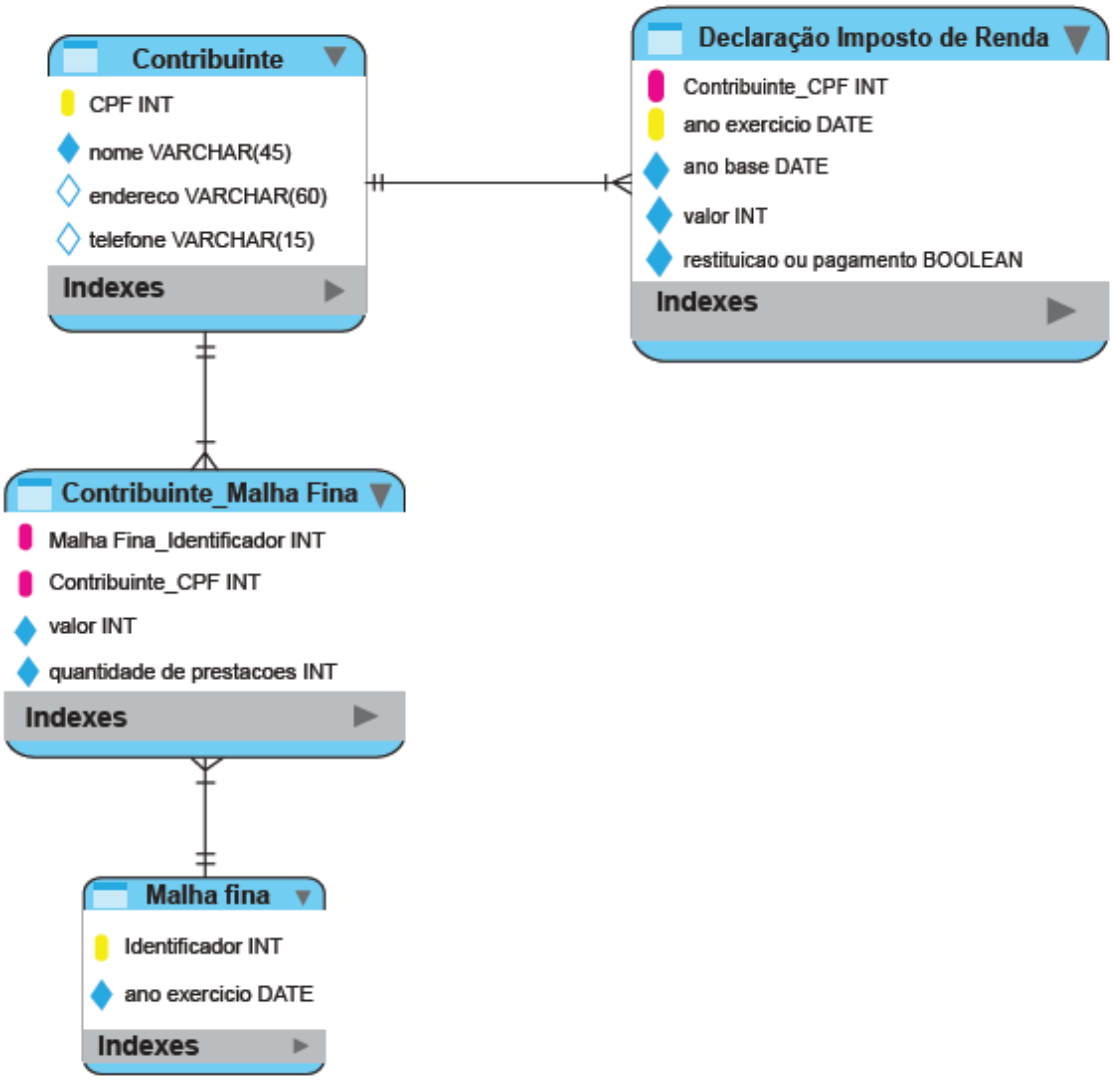
[6]Questão 11 – Enade 2014.

D.
```
select titulo from livro
union select l.titulo from emprestimo e left join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

E.
```
select titulo from livro
except
select l.titulo from emprestimo e right join livro l
on e.livro_cod = l.liv_cod where e.data_dev is not null
```

Questão 7.[7]

O modelo lógico de dados fornece uma visão da maneira como os dados serão armazenados. A figura a seguir representa o modelo lógico de um ambiente observado em um escritório contábil.



[7]Questão 21 – Enade 2014.

Em relação ao modelo, avalie as afirmativas.

I. A entidade Declaração Imposto de Renda é uma entidade fraca.

II. O relacionamento entre Contribuinte e Malha Fina é do tipo N:M (muitos para muitos).

III. O atributo CPF da entidade Contribuinte tem a função de chave estrangeira na entidade Declaração Imposto de Renda e no relacionamento Contribuinte_MalhaFina.

IV. A entidade Malha Fina não possui chave primária somente chave estrangeira.

V. O relacionamento Contribuinte_MalhaFina é um relacionamento ternário.

É correto apenas o que se afirma em:

A. I, II e III.

B. I, II e IV.

C. I, IV e V.

D. II, III e V.

E. III, IV e V.

Questão 8.[8]

Leia o texto a seguir.

O modelo relacional representa o banco de dados como uma coleção de relações (tabelas). Na terminologia formal do modelo relacional, uma linha é chamada de "tupla", o título da coluna é denominado "atributo" e a tabela é chamada de "relação". O tipo de dado que descreve os tipos de valores que podem aparecer em cada coluna é denominado "domínio". Um banco de dados relacional pode impor vários tipos de restrições nos dados armazenados.

ELMASRI, R. NAVATHE, S. B. *Sistema de Banco de Dados: Fundamentos e Aplicações*. Rio de Janeiro: LTC, 2002.

Restrições que permitem controlar situações como, por exemplo, "o salário de um empregado não deve exceder o salário do supervisor do empregado" e utilizam mecanismos chamados *triggers* (gatilhos) na sua implementação, são do tipo

A. restrições de domínio.

B. restrições de unicidade.

C. restrições de integridade referencial.

D. restrições de integridade da entidade.

E. restrições de integridade semântica.

[8]Questão 22 – Enade 2014.

1. Introdução teórica

1.1. Banco de dados relacionais

Frequentemente, problemas na área de computação lidam com grande quantidade de dados, gerados de forma artificial (por um programa), coletados do mundo externo (por exemplo, por meio de cadastros ou de instrumentos conectados ao computador) ou frutos de interação do usuário com o computador. Muitas vezes, queremos que esses dados sejam armazenados de forma permanente ou por um período de tempo potencialmente longo.

Podemos armazenar dados de várias formas. Por exemplo, um simples arquivo de texto gravado em um *pen drive* é uma forma válida de armazenamento de dados. Porém, conforme a quantidade de dados armazenados aumenta, pode surgir uma série de problemas: encontrar dados específicos em meio a uma grande quantidade de dados disponíveis, relacionar os dados entre si e atualizar e remover dados.

Com o aumento da capacidade de armazenamento dos computadores, bem como a sua popularização, várias técnicas de organização e de gerenciamento de dados foram desenvolvidas. Uma das formas que se tornou bastante importante no desenvolvimento de grandes sistemas é a dos bancos de dados relacionais.

Seguindo a abordagem de Ritchie (2002), podemos dizer, de forma simplificada, que os bancos de dados relacionais são uma coleção de tabelas interligadas.

As tabelas são formalmente chamadas de relações e cada uma de suas linhas (que são fisicamente registros da tabela) é chamada de tupla. Cada tabela tem várias colunas, chamadas de atributos. Como exemplo, considere a tabela 1 (ou relação) a seguir.

Tabela 1. Exemplo de uma tabela em um banco de dados.

Compositor	
Nome	**Gênero**
Ludwig van Beethoven	Clássico
Johann Sebastian Bach	Barroco
John Coltrane	Jazz
Miles Davis	Jazz
Antônio Carlos Jobim	MPB

Na tabela 1, "Compositor", os atributos (colunas) são "Nome" e "Gênero". Uma tupla dessa tabela seria: "Miles Davis; Jazz".

O diagrama de entidades e relacionamento apresenta uma notação especial para o projeto de banco de dados. Nesses diagramas, existem três elementos básicos: tipos de entidade, relacionamentos e atributos. O primeiro elemento, o tipo de entidade, é representado por um quadrado na maioria das notações, e é utilizado para representar objetos, coisas, pessoas e também eventos.

Em particular, uma entidade é um membro ou instância de um tipo de entidade. Os atributos são as propriedades dos tipos de entidade, enquanto os relacionamentos são as associações entre os tipos de entidades.

Outro aspecto importante no modelo relacional é que cada linha de uma tabela deve ter uma combinação única de elementos, não podendo haver linhas repetidas. Contudo, isso não quer dizer que não exista a possibilidade de que alguns valores individuais não se repitam, mas sim de que a linha toda seja única. Por exemplo, em uma tabela contendo o nome, o CPF e o salário de cada um dos funcionários de uma empresa, pode haver mais de um funcionário com o mesmo nome e salário (mas nunca com o mesmo CPF, que é único para cada indivíduo). Logo, cada linha dessa tabela é única.

Um conjunto de colunas com uma combinação única é chamado de superchave. Por isso, uma linha completa (com todas as colunas) é sempre uma superchave. Contudo, se pudermos remover elementos da superchave até o ponto em que temos uma combinação mínima de colunas (mas ainda única), dizemos que essa superchave mínima é uma chave candidata. Quando uma chave candidata é especialmente designada para uma tabela, dizemos que ela é uma chave primária.

Para ser possível referenciar tabelas diferentes, pode-se adicionar a chave primária de uma tabela em outra tabela, o que permite a seleção de um registro específico de uma tabela a partir de outra. A isso chama-se chave estrangeira.

Em um diagrama de entidade-relacionamento, existe a possibilidade de que uma entidade dependa de outra. Essa entidade pode precisar que parte da sua chave primária venha de outros tipos de entidades (ou até mesmo toda a chave). Nesse caso, dizemos que ela é uma entidade fraca.

Com o objetivo de garantirmos a integridade das entidades e dos dados armazenados em um banco de dados, introduzimos diversos tipos de restrições, especialmente ligados a cada tipo de integridade.

Assim, se quisermos garantir a integridade referencial, entende-se que a chave estrangeira de uma tabela deve conter os valores vindos da chave primária da tabela pai (ou da chave candidata) ou o valor nulo.

A integridade semântica significa que cada atributo (ou coluna) apresenta um valor consistente com o tipo de dado. A integridade da entidade significa que cada ênupla tem uma chave primária única não nula.

1.2. Consultas e linguagem SQL

Durante muito tempo, para obter as informações contidas nas tabelas dos bancos de dados, era preciso saber fisicamente como essas informações estavam armazenadas e organizadas no banco. O programador precisava conhecer profundamente a forma como o banco funcionava para poder conseguir os dados necessários. Isso levava a um problema de acoplamento: qualquer mudança no banco afetaria a estrutura do código do programa, mesmo que essa mudança não fosse conceitual e fosse apenas armazenamento dos dados. Ao mesmo tempo, mudanças no código também afetavam e costumavam levar a modificações na estrutura do banco de dados.

Para simplificar o desenvolvimento de programas que utilizam bancos de dados, a linguagem SQL foi desenvolvida, partindo-se de um paradigma declarativo, em vez de um paradigma imperativo. Por exemplo, a linguagem C utiliza um paradigma imperativo, enquanto a linguagem Lisp utiliza um paradigma funcional e declarativo.

Dessa forma, o programador deve concentrar-se no resultado que ele busca (os registros desejados do banco), e não na forma como esses registros são encontrados no banco de dados. Além disso, a linguagem SQL foi desenvolvida especificamente a partir da consulta a banco de dados relacionais.

2. Análise das questões

Questão 6

O resultado desejado corresponde a "encontrar todos os livros, exceto aqueles que estão emprestados". Um livro está emprestado se está presente na tabela "Empréstimo" e não ter valor na coluna "data_dev" (ou seja, essa coluna deve conter o valor NULL). Dessa forma, para selecionarmos todos os livros que estão emprestados, fazemos o que segue.

```
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

Lembrando que o comando "inner join" retorna apenas os registros que apresentam o mesmo valor na coluna "livro_cod" da tabela "empréstimo" e da coluna "liv_cod" na tabela "livro". Além disso, observe a condição "e.data_dev is null", ou seja, apenas os registros que não apresentarem valores em "data_dev".
Se quiséssemos selecionar todos os livros, independentemente de estarem emprestados ou não, deveríamos fazer:

```
select titulo from livro
```

Essa consulta vai retornar todos os títulos presentes na tabela livro. Excluiremos os registros que estão emprestados, retornados na consulta anterior. Isso pode ser feito utilizando-se o comando except, que elimina, da primeira consulta, os registros obtidos pela segunda consulta, conforme segue.

```
select titulo from livro
except
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

Alternativa correta: A.

Questão 7

I – Afirmativa correta.
JUSTIFICATIVA. A entidade "Declaração de Imposto de Renda" depende da existência da entidade "Contribuinte" e tem como discriminador o atributo "ano exercício".

II – Afirmativa correta.
JUSTIFICATIVA. A tabela "Contribuinte_Malha Fina" faz o papel de uma tabela de ligação entre as entidades "Contribuinte" e "Malha Fina", justamente porque o relacionamento é do tipo N:M (muitos para muitos).

III – Afirmativa correta.
JUSTIFICATIVA. Tanto na entidade "Contribuinte_Malha Fina" quanto na entidade "Declaração Imposto de Renda", é necessário o atributo CPF, que aponta para a entidade "Contribuinte". Logo, esse atributo é uma chave estrangeira.

IV – Afirmativa incorreta.
JUSTIFICATIVA. O atributo "Identificador" é a chave primária da entidade "Malha Fina".

V – Afirmativa incorreta.
JUSTIFICATIVA. O relacionamento "Contribuinte_Malha Fina" é um relacionamento entre duas entidades, as entidades "Contribuinte" e "Malha Fina" e, portanto, é um relacionamento binário.

Alternativa correta: A.

Questão 8

A – Alternativa incorreta.
JUSTIFICATIVA. A restrição descrita no enunciado não limita o tipo de dados de um atributo e, portanto, não é uma restrição de domínio.

B – Alternativa incorreta.
JUSTIFICATIVA. A restrição descrita no enunciado não garante unicidade.

C – Alternativa incorreta.
JUSTIFICATIVA. A restrição descrita no enunciado não menciona tipo de relacionamento entre entidades e, portanto, não pode ser vista como uma restrição de integridade referencial.

D – Alternativa incorreta.
JUSTIFICATIVA. A restrição descrita no enunciado não garante que mais de um empregado tenha os valores iguais em uma tabela, apenas garante que o empregado ganhe menos do que o seu supervisor. Por exemplo, dois funcionários poderiam ter o mesmo salário, desde que esse salário fosse menor do que o salário do supervisor.

E – Alternativa correta.

JUSTIFICATIVA. A restrição descrita no enunciado é uma regra de negócio. Logo, é uma restrição de integridade semântica.

3. Indicações bibliográficas

- CONRAD, E.; MISENAR, S.; FELDMAN, J. *CISSP Study Guide*. Waltham: Newnes, 2012.
- CORONEL, C.; MORRIS, S. *Database systems: design, implementation and management.* 12. ed. Boston: Cengage Learning, 2016.
- MANNINO, M. V. *Projeto, desenvolvimento de aplicações e administração de banco de dados.* 3. ed. Porto Alegre: AMGH, 2008.
- RITCHIE, C. *Relational database principles.* London: Thomson Learning, 2002.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Database System Concepts.* 6. ed. New York: McGraw-Hill, 2011.

Questão 9

Questão 9.[9]

Leia o texto a seguir.

O serviço DNS (Domain Name System) traduz nomes alfanuméricos de hosts em endereços numéricos, de acordo com o protocolo IP (Internet Protocol). Essa ação é comumente chamada de resolução de endereço.

TANENBAUM, A. S. *Redes de Computadores*. Rio de Janeiro: Campus, 2003 (com adaptações).

Considere um conjunto de computadores conectados em uma rede local, os quais têm à sua disposição um servidor DNS capaz de resolver endereços, sejam eles internos ou externos.

Nesse contexto, avalie as afirmativas a seguir.

I. O servidor DNS também executa funções de cliente DNS quando não é autoritativo para determinado endereço.

II. A adoção do IPv6 (Internet Protocol, versão 6) dispensará serviços de DNS, pois suas funções serão incorporadas pelo próprio protocolo IP.

III. O cache DNS permite que determinada requisição do cliente DNS possa ser resolvida sem que seja necessário recorrer a outro serviço DNS.

IV. O protocolo DNS depende de um banco de dados distribuído.

É correto apenas o que se afirma em

A. I e II.

B. I e III.

C. II e IV.

D. I, III e IV.

E. II, III e IV.

1. Introdução teórica

Domain Name Server (DNS)

As máquinas presentes em uma rede que usa o protocolo TCP/IP têm ao menos um endereço numérico, chamado de endereço IP. O protocolo IPv4 especifica que o endereço IP deve possuir 32 bits, mas devido à grande expansão da Internet após o final da década de 1990, esse tamanho tornou-se muito restritivo. Foi criado, então, o IPv6, de 128 bits, aumentando, também, a quantidade disponível de endereços. Contudo, a implantação do IPv6 vem ocorrendo de forma gradual e ainda não está completa.

[9]Questão 26 – Enade 2014.

A ideia de duas máquinas comunicarem-se utilizando endereços IP é bastante conveniente para os computadores, mas não necessariamente para seres humanos. Por exemplo, para acessar o site da Unip (www.unip.br) pelo endereço IP (v4, de 32 bits), deveríamos digitar o número 200.196.224.129. Obviamente, navegar por sites utilizando números grandes é bastante inconveniente para a maioria dos usuários, sendo muito mais simples utilizar o nome www.unip.br. Contudo, as máquinas continuam utilizando o endereço IP para a comunicação. Logo, é necessário que exista alguma tecnologia para "traduzir" o nome do endereço (no caso, www.unip.br) para o endereço IP 200.196.224.129.

O serviço de DNS faz precisamente essa "tradução". Dessa forma, ao digitarmos a URL de um site no navegador, o servidor de DNS procura qual é o endereço IP correspondente a www.unip.br e retorna essa informação para o computador cliente, que vai agora utilizar o endereço IP para a comunicação. Para o usuário, essa é uma operação transparente: ele apenas deve saber a URL do site (www.unip.br) e o computador, automaticamente, solicita o endereço IP para o servidor de DNS.

Devido ao grande número de sites que existem (um número em constante aumento), o tamanho do banco de dados de DNS é muito grande. Tal informação é importante, pois, por exemplo, a associação do endereço IP de um banco ou de uma instituição financeira à sua URL é alvo de ataques maliciosos em busca de vulnerabilidades. Logo, servidores de DNS devem ser cuidadosamente protegidos.

Além disso, o número de clientes requisitando endereços IP de um servidor de DNS é muito grande, uma vez que o número de máquinas na internet é gigantesco. Assim, é interessante que exista um grande número de servidores de DNS para distribuir a carga da consulta entre diferentes máquinas, preferencialmente as que estejam próximas aos clientes. A fim de facilitar esse processo, existem três tipos de servidores de DNS: os servidores raiz, os servidores autoritativos e os servidores intermediários.

Os servidores autoritários contêm a informação original que associa um endereço IP a uma URL. Quando um servidor de DNS intermediário precisa identificar um endereço IP, ele entra em contato com um dos servidores DNS raiz para identificar qual servidor autoritário contém a informação sobre o endereço IP. Com essa informação, o servidor intermediário pode, então, contatar o servidor autoritativo e identificar e receber a informação do endereço IP, que vai ser passada para o cliente do servidor de DNS.

Com a finalidade de agilizar o processo de consulta, os servidores intermediários de DNS podem manter um conjunto de dados temporários, vindo das consultas anteriores,

chamado de cache local. Dessa forma, se o servidor intermediário já contiver a informação, não é necessário contatar um servidor raiz ou um servidor autoritativo, aliviando a carga nesses servidores (existe um número muito maior de servidores intermediários). No entanto, essa informação é mantida apenas por certo período de tempo, uma vez que pode haver atualizações nas informações (que vão ocorrer inicialmente nos servidores autoritativos).

2. Análise das afirmativas

I – Afirmativa correta.
JUSTIFICATIVA. Quando um servidor não é autoritativo para um endereço, significa que ele não tem, em seu banco de dados, os registros originais que associam determinado domínio a um endereço IP. Ele pode possuir os dados em cache, caso já tenho sido feita uma pesquisa para dado domínio, mas o conteúdo dessa cache é originado em um servidor remoto. Assim, quando o servidor não é autoritativo para um endereço, ele dever buscar o endereço em outros servidores DNS.

II – Afirmativa incorreta.
JUSTIFICATIVA. A adoção do IPv6 vai mudar o tamanho do endereço IP armazenado pelo servidor de DNS, de 32 bits para 128 bits. Contudo, o servidor de DNS, que fica na camada de aplicação, continua existindo.

III – Afirmativa correta.
JUSTIFICATIVA. O cache de um servidor DNS permite que uma informação que tenha sido obtida por uma consulta prévia seja reaproveitada em consultas similares subsequentes. Dessa forma, consultas similares não necessitam gerar novamente tráfego de rede aos servidores autoritativos, além de serem mais rápidas. Consultas em cache podem ficar obsoletas, se a informação armazenada na cache mudar. Por isso, uma das informações armazenadas no banco de dados do servidor DNS é o campo "*Time_to_live*", que registra em quanto tempo (em segundos) o registro deve ser atualizado. Dessa forma, o servidor de DNS pode saber por quanto tempo a informação armazenada no seu cache permanece válida.

IV – Afirmativa correta.

JUSTIFICATIVA. A Internet é uma vasta rede, de alcance mundial e que envolve milhões de máquinas. Se o serviço de DNS fosse centralizado, teríamos uma série de problemas, tanto de desempenho (milhares de computadores no mundo todo acessando uma única central de informações) e de segurança (a estrutura central se tornaria alvo de vários tipos de ataques), quanto de confiabilidade (se o servidor estivesse fora do ar, a rede mundial não funcionaria). Para resolver esses problemas, o serviço de DNS depende de um banco de dados distribuído, ou seja, várias máquinas com autoridade local que dividem a responsabilidade por zonas virtuais; a falha de um servidor de DNS pode afetar no máximo uma pequena fração das máquinas na Internet.

Alternativa correta: D.

3. Indicações bibliográficas

- BOURKE, T. *Server load balancing.* Sebastopol: O'Reilly Media, 2001.
- TANENBAUM A. S.; WETHERALL D. J. *Computer networks.* Upper Saddle River: Prentice Hall, 2011.

ÍNDICE REMISSIVO

Questão 1	Sistemas distribuídos. Arquitetura de software. Sistemas operacionais. Comunicação entre computadores. Compartilhamento.
Questão 2	Estruturas de dados. Listas ligadas. Árvores. Introdução à programação.
Questão 3	Funções recursivas. Linguagem C. Estruturas de dados. Introdução à programação. Pilhas.
Questão 4	Arquitetura de software. Engenharia de software. Arquitetura em camadas.
Questão 5	Estruturas de dados. Árvores binárias. Nós. Introdução à programação.
Questão 6	Banco de dados relacionais. Modelo entidade relacionamento. Linguagem SQL.
Questão 7	Banco de dados relacionais. Modelo lógico. Tipos de entidades.
Questão 8	Banco de dados relacionais. *Triggers* (gatilhos). Restrições.
Questão 9	Redes de computadores. Protocolo IP. *Domain Name System* (DNS).