Estratégia Concursos

Aula 00

TCM-SP (Auditor de Controle Externo - Tecnologia da Informação) Banco de dados

Autor:
Thiago Rodrigues Cavalcanti

07 de Setembro de 2023

Índice

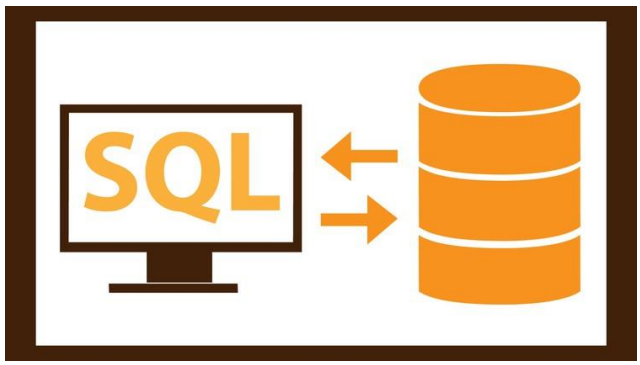
INTRODUÇÃO A SQL

A linguagem SQL foi criada por um grupo de pesquisadores do laboratório da IBM em San Jose, Califórnia, mais especificamente pelos pesquisadores **Donald D. Chamberlin e Reymond Boyce**. Eles publicaram um artigo denominado "**SEQUEL: *A Structured English Query Language***", no qual apresentavam detalhes da linguagem, como sua **sintaxe e operações**. A IBM chegou a implementar, na IBM Research, o SQL como a interface para um sistema de banco de dados relacional experimental, chamado **SYSTEM R**.


Donald D. Chamberlin


Raymond F. Boyce

Alguns anos depois da publicação do artigo, devido a um conflito de patentes com um projeto secreto de uma empresa inglesa de aviação, SEQUEL mudou de nome para SQL.



A *Structured Query Language* – SQL – pode ser considerada **uma das principais razões para o sucesso dos bancos de dados relacionais**. Ela se provou bastante útil e foi adotada como padrão pela ANSI (*American National Standards Institute*). São várias versões da norma **ANSI/ISO/IEC 9075** publicadas ao longo do tempo: 1986, 1989, 1992, 1999, 2003, 2006, 2008, 2011 e 2016. Nove ao todo! Nosso foco de estudo se baseia na parte da norma que apresenta os fundamentos da linguagem SQL: **ISO/IEC 9075-2 (SQL/Foundation).** Uma norma geralmente possui várias partes, uma numeração separa os conteúdos específicos em diversos documentos.

Por exemplo, a norma produzida pelo comitê com a numeração ISO/IEC 9075-**1** descreve a **estrutura conceitual** usada para especificar a **gramática** de SQL e o resultado das instruções de processamento

nessa linguagem. A norma também define os termos e a notação utilizados nos demais documentos da série.

O termo **tabela** é definida na norma ISO/IEC 9075-**1.** "Uma **tabela** possui uma **coleção** ordenada de uma ou mais **colunas** e uma coleção não ordenada de zero ou mais **linhas**. Cada coluna tem um nome e um tipo de dados. Cada linha tem, para cada coluna, exatamente um valor no tipo de dados dessa coluna." (tradução da versão em inglês). Observe a tabela APROVADOS abaixo e preencha a linha em branco com os seus dados 😊

APROVADOS

Identificador	Nome	Data de Nascimento	Concurso Aprovado
1	Thiago	12/02/1983	Banco Central

OK! Agora vamos transformar em realidade a tabela acima. SQL é uma linguagem de programação. Pense que programar é semelhante a aprender uma nova língua. Você precisa estudar os elementos da linguagem para conseguir escrever bons textos. Para manter banco de dados relacionais você vai construir tabelas e relacionamentos entre elas. Em seguida, você poderá inserir, consultar, atualizar e remover dados das tabelas. **A linguagem SQL vai viabilizar essas ações**. Antes de colocar a mão na massa para criar e inserir dados nas tabelas, vamos definir alguns conceitos.

CONCEITOS BÁSICOS

Primeiramente, SQL é uma linguagem de programação reconhecida internacionalmente e usada para definição e manutenção de bancos de dados relacionais. A **principal característica da linguagem** é ser **declarativa**[1], ou seja, os detalhes de implementação dos comandos são deixados para os SGBDs relacionais. Não esqueça disso! Já caiu várias vezes em provas anteriores. No SQL você **declara o que você quer e o SGBD vai achar os dados para você no banco, caso existam é claro.**

Vamos tirar do contexto de programação e tentar entender a ideia por trás de linguagem declarativa e da sua "rival" a linhagem procedural.

[1] Linguagens declarativas se contrapõem com as linguagens procedurais. Nestas você precisa descrever o passo a passo para execução de uma determinada tarefa.

Imagine que eu te faça a seguinte pergunta: Estou ao lado do supermercado Pão de Açúcar, como eu faço para chegar na sua casa?

A resposta **procedural** seria: Pegue a direita, faça o retorno na rotatória e volte no sentido do bosque do Sudoeste, em seguida, vire à direita e siga até a primeira avenida, então saia na rotatória na terceira saída. Siga até a entrada da quadra 101/102 e procure pelo bloco C da quadra 101. Veja que eu mostrei o passo a passo da rota até a minha casa.

E resposta **declarativa**!? Como seria? Vejamos: Meu endereço é na Quadra 101 bloco C, Sudoeste Brasília, Apto 304[2].

Percebe a diferença? Linguagem declarativa descrevem **O QUE** você quer, já a linguagem procedural descreve **COMO** fazer o que você quer.

Outro aspecto importante é que SQL **não** existe fora do contexto relacional devido ao fato de ser fundamentada no neste modelo. Perceba que, por ser considerado um padrão, teoricamente, deve ser possível portar um banco de dados de um SGBD para outro com certa facilidade. A linguagem utiliza o cálculo e a álgebra relacional como base teórica para implementar as operações dentro dos SGBDs.

Ficou assuntado com esses termos matemáticos? Não se preocupe! A **sintaxe SQL é mais fácil de ser utilizada do que qualquer uma das duas linguagens formais. Quer um exemplo?** Vamos voltar para a nossa tabela inicial. Favor preencher novamente com seus dados.

APROVADOS

Identificador	Nome	DataDeNascimento	Concurso Aprovado
1	Thiago	12/02/1983	Banco Central
3	Diego	15/03/1984	STN
4	Herbert	05/02/1987	SEFAZ

Vamos fazer uma consulta sobre a tabela aprovados para extrair apenas as colunas Nome e Data de Nascimento.

Colunas da tabela

[2] Esse não é meu endereço. Se você estiver pensando em me mandar um presente entre em contato.

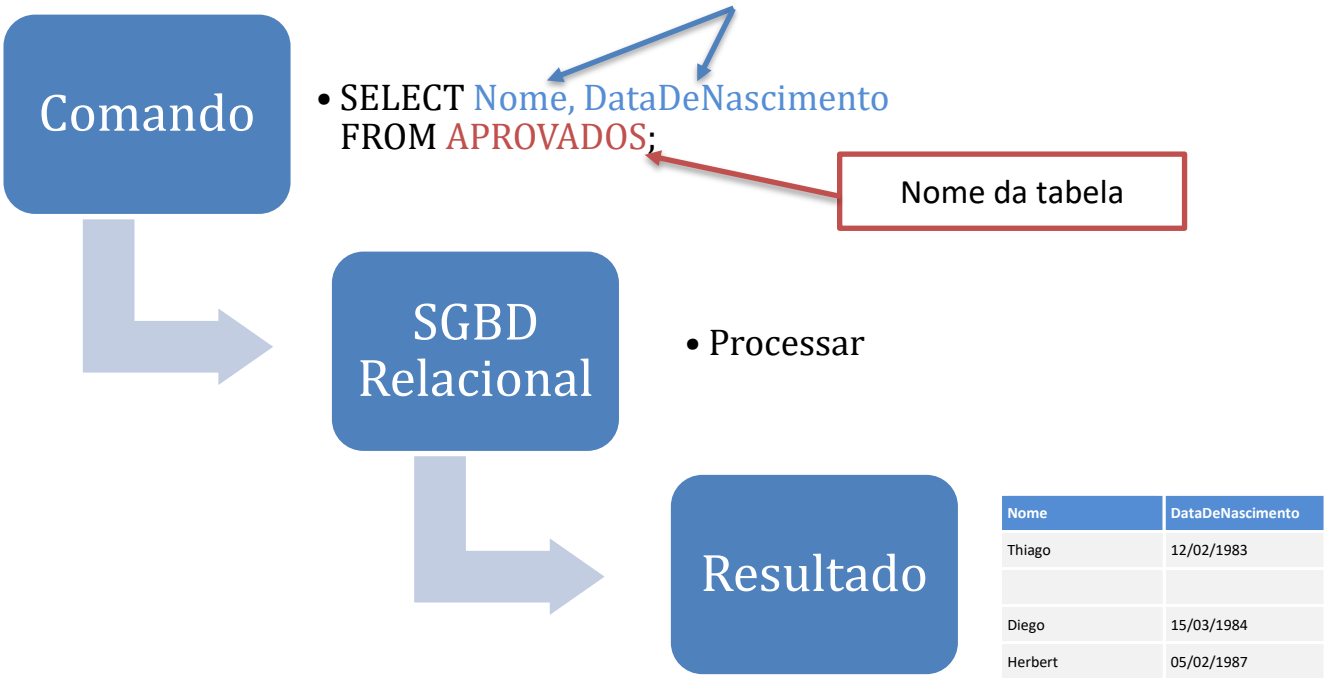


Figura 1 - Um exemplo de consulta

Seu nome e data de nascimento, devem aparecer no resultado da mesma forma estão escritos na tabela anterior. Percebe que é simples? Dizemos para o SGBD quais as colunas que vamos retornar da cláusula SELECT e a tabela onde estão os dados na cláusula FROM ... e voilà! Eis a nossa resposta em forma de tabela exatamente com os dados que solicitamos.

Mas o SELECT não é o único comando presente na linguagem. O SELECT faz parte de um grupo de comandos de **manipulação da base de dados**. Ele vai procurar os dados em tabelas usando critérios de busca definidos pelo usuário. Vamos conhecer um pouco mais sobre os grupos de comandos da linguagem.

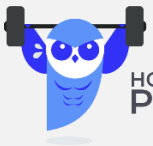
SUBLINGUAGENS DE SQL

Quando começamos a estudar a linguagem, precisamos entender as subdivisões que existem dentro dos comandos possíveis. SQL é uma linguagem de banco de dados abrangente, tem instruções para **definição** de dados (DDL), consultas e atualizações (DML). Possui ainda facilidades para definir **visões** sobre o banco de dados, para especificar **segurança e autorização**, para definir **restrições de integridade** e para especificar **controles de transação**.

Alguns autores chegam a dividir a linguagem em cinco categorias. As categorias são baseadas nas funcionalidades que cada comando executa sobre o banco de dados. Vejam a lista abaixo:

- **DDL** – Data Definition Language – A linguagem de definição de dados contém comandos que criam, modificam e excluem objetos de banco de dados. São exemplos de comando: CREATE, ALTER, DROP e TRUNCATE.
- **DML** – Data Manipulation Language – A linguagem de manipulação de dados fornece instruções para trabalhar com os dados armazenados como SELECT, INSERT, UPDATE, DELETE e MERGE.
    - **DQL** – Data Query Language – A linguagem de consulta de dados é um subconjunto da DML que possui apenas a instrução de SELECT.
- **DTL** – Data Transaction Language – Linguagem de transação de dados inclui comandos de COMMIT, ROLLBACK e SAVEPOINT
- **DCL** – Data Control Language – A linguagem de controle de dados contém os comandos relacionados com as permissões de controle de acesso. Garante os privilégios aos usuários para acessar os objetos do banco. Os mais conhecidos comandos são o GRANT e o REVOKE.

Agora que já entendemos a diferença entre cada uma das subdivisões da linguagem, vamos fazer duas questões para fixação.



(Ano: 2016 Órgão: TRE-SP Cargo: Analista Judiciário de TI – Q. 56.)

Em uma situação hipotética, ao ser designada para atender aos requisitos de negócio de um usuário, uma Analista de Sistemas do TRE-SP escreveu expressões e comandos para serem executados em um Banco de Dados Relacional que visavam (1) criar uma tabela que contivesse dados de processos partidários, (2) controlar a segurança e o acesso a ela e (3) manipular dados nela. Desta forma ela, se valeu, correta e respectivamente, por exemplo, de alguns elementos de expressões tais como:

(A) INSERT, REVOKE e SELECT.
(B) CREATE, REVOKE e INSERT.
(C) CREATE, GRANT e ALTER.
(D) DROP, ALTER e UPDATE.
(E) INSERT, INDEX e CREATE.

Comentário: Ok! Questão introdutória do assunto de banco de dados, relaciona os comandos da linguagem SQL a uma taxonomia específica. Na questão são apresentados o conjunto de comados de criação de objetos ou *data definition language* (CREATE, DROP, ALTER), comandos relacionados à segurança dos dados ou *data control language* (GRANT e REVOKE) e os comandos utilizados para a manipulação de dados ou *data manipulation language* (DELETE, INSERT, SELECT e UPDATE). Os comandos entre parênteses são apenas alguns exemplos dos representantes de cada subgrupo da linguagem SQL.

De posse dessas informações, podemos encontrar nossa resposta na alternativa B, o que está de acordo com o gabarito apresentado pela banca.

Gabarito: B.

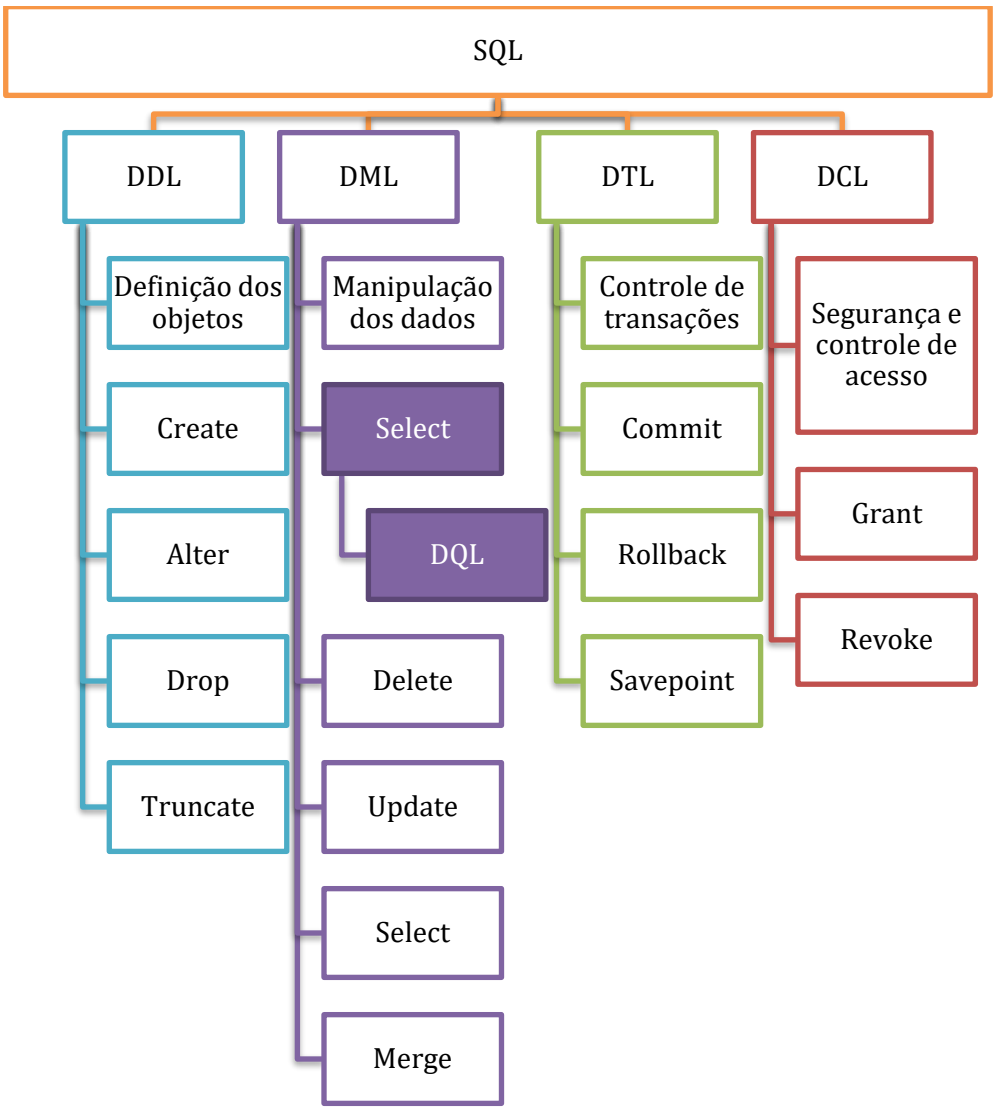


Figura 2 - Sublinguagem de SQL

UMA EXTENSÃO DE SQL: LINGUAGEM PROCEDURAL

Pessoal, com o passar do tempo, alguns fornecedores introduziram alguns comandos procedurais, entre outras extensões dentro do SQL e os lançaram com nomes parecidos, tais como: **PL/SQL ou Transact-SQL** – no entanto eles são conhecidos apenas como dialetos. Nosso intuito aqui é estudar o SQL tradicional (de raiz! Padrão!), definido pelo American National Standards Institute (ANSI), mas gostaria de fazer um rápido comentário sobre a existência destes dialetos.

Grandes empresas, no intuito de melhorar suas ferramentas de banco de dados quiseram incrementar as funcionalidades dos seus SGBDs e procuraram expandir a linguagem aceita por eles. Como assim? A Oracle e a Microsoft resolveram que seria legal que, além dos comandos presentes na linguagem SQL padrão, o SGBD **entendesse mais algumas informações ou comandos.**

Estes dialetos comumente evoluem mais rapidamente do que o padrão SQL ANSI, pois, um determinado banco de dados de uma comunidade de usuários ou de um fornecedor **requer**

funcionalidades para a utilização do banco de dados antes do comitê da ANSI/ISO criar uma versão do padrão que aplique tal funcionalidade.

> Ocasionalmente as comunidades acadêmicas e de pesquisa introduzem novas características em resposta a pressões de tecnologias concorrentes. Por exemplo, muitos fornecedores de banco de dados estão aumentando suas ofertas às funções programáticas tanto com Java (DB2, Oracle e Sybase) quanto com VBScript (Microsoft). Hoje em dia, programadores e desenvolvedores usam estas linguagens de programação em conjunto com SQL para construir programas. Nos últimos anos, com o frenesi da análise de dados, linguagens como R e Python também estão integradas a grande parte dos SGBD, como por exemplo SQL Server.

Assim, existem dois motivos para agregar uma linguagem procedural ao SQL. Uma é **permitir o desenvolvimento de definições mais complexas no banco de dados**. O outro é o **desempenho**.

Muitos destes dialetos incluem habilidades de **processamento condicional** (controladas por instruções do tipo IF ...THEN), funções de **controle de fluxo** (como laços WHILE e FOR), **variáveis** e habilidades de manipulação de erros. A ANSI não desenvolveu ou evoluiu o padrão para que estas características importantes estivessem disponíveis quando os usuários começaram a exigir; sendo assim, desenvolvedores e fornecedores de SGBDR criaram seus próprios comandos e sintaxe.

Alguns destes dialetos **introduziram comandos procedurais** para suportar a funcionalidade de uma linguagem de programação mais completa. Por exemplo, estas implementações procedurais contêm comandos para **lidar com erros**, **linguagem de controle de fluxo**, **comandos condicionais**, **comandos para manipular variáveis**, **suporte para *arrays*** e **muitas outras extensões**.

Embora estas sejam implementações procedurais tecnicamente divergentes, elas são chamadas, aqui, de **dialetos**. Outro ponto interessante é que a ANSI/ISO lançaram uma extensão para agregar a linguagem procedural ao SQL. O pacote, denominado SQL/PSM (*Persitent Stored Module*) tenta padronizar a sintaxe e a semântica para o fluxo de controle, tratamento de exceções, variáveis locais, atribuição de expressões a variáveis e parâmetros, e uso (procedural) de cursores.

Vejam abaixo uma lista com os principais dialetos disponíveis no mercado.

PL/SQL - Encontrado em SGBDs Oracle. PL/SQL, que é a sigla para Procedural Language/SQL, contém muitas semelhanças com a linguagem de programação geral.

Transact-SQL - Usado pelo Microsoft SQL Server e Sybase Adaptive Server. Como Microsoft e Sybase se afastaram, a plataforma comum que compartilhavam no início na década de 1990 foi dividida, suas implementações de agora também divergem, produzindo dois dialetos distintos de Transact-SQL.

SQL PL - extensão processual do IBM DB2 para SQL, introduzido na versão 7.0 do SGBD, fornece construções necessárias para a execução de controle de fluxo em torno de consultas SQL tradicionais e operações.

PL/pgSQL - O nome do dialeto SQL e das extensões implementadas no PostgreSQL. A sigla significa Procedural Language/PostgreSQL.

MySQL – O MySQL introduziu uma linguagem procedural em seu banco de dados na versão 5, mas não há nenhum nome oficial para ela. Ela segue o padrão definido pela ANSI (SQL/PSM).

Uma lista completa dos dialetos associados a linguagem SQL agrupada por fornecedores de SGBDs relacionais pode ser visualizada na tabela abaixo:

Tabela 1 - Dialetos de SQL e respectivos fornecedores

Fonte/SGBDR	Nome comum	Nome completo
ANSI/ISO	SQL/PSM	SQL/Persistent Stored Modules
Interbase/Firebird	PSQL	Procedural SQL
IBM DB2	SQL PL	SQL Procedural Language
MySQL	SQL/PSM	SQL/Persistent Stored Modules
Oracle	PL/SQL	Procedural Language SQL
PostgreSQL	PL/pgSQL	Procedural Language PostgreSQL
SQL Server/Sybase	T-SQL	Transact-SQL

Aprender SQL vai proporcionar as habilidades que você precisa para recuperar informações a partir de qualquer banco de dados relacional. Também ajuda a compreender os mecanismos por trás das interfaces gráficas de consulta encontradas em muitos produtos de SGBDR. O entendimento de SQL vai ajudar você a criar consultas complexas e a fornecer o conhecimento necessário para corrigir consultas quando ocorrem problemas. Mas, você deve estar se perguntando: quais as diferenças entre linguagem procedurais e SQL? Vejamos uma comparação na tabela abaixo:

Tabela 2 - Comparação dentre PL/SQL e SQL

PL/SQL	SQL
Os blocos de códigos podem ser usados para escrever programas inteiros.	Os comandos são usados para executar operações de DML e DDL.
Por ser **procedural**, devemos descrever **COMO** as coisas devem ser feitas.	Por ser **declarativa**, devemos definir apenas **O QUE** deve ser feito.
São executados em blocos inteiros	São executados em declarações únicas. (*statements*).
Usados para criar aplicações.	Usados para manutenção dos dados.

Uma extensão de SQL, é permitido o uso de código SQL dentro do código PL/SQL	Não pode conter código PL/SQL.

Certo, você já sabe as diferenças! Para finalizar este tópico da nossa aula queria que você conhecesse um exemplo de código em PL/SQL, a extensão procedural da Oracle. Quero mostrar para você os principais componentes presentes no código. Basicamente, um bloco PL/SQL é composto por uma área de **declaração de variáveis** (*declare*), uma **área de escopo** para inserção de comandos e outros sub-blocos (*begin-end*) e uma área de **tratamento de erros** (*exception*).

```
1  SQL> declare
2    2    soma number;
3    3  begin
4    4    soma := 45+55;
5    5    dbms_output.put_line('Soma :'||soma);
6    6  exception
7    7    when others then
8    8     raise_application_error(-20001,'Erro ao
somar valores!');
9    9  end;
10  10
11
12 SQL>
```


Figura 3 - Exemplo de código PL/SQL

Vamos, a seguir, tratar detalhadamente dos comandos de definição de dados (DDL). É importante perceber que, para manipular os dados em objetos como tabelas, os objetos precisam existir na base da dados. E, para criar esses objetos precisamos entender quais comandos são usados para criação e alteração das estruturas, em especial das tabelas.

LINGUAGEM DE DEFINIÇÃO DE DADOS (DDL)

Quando estamos construindo um modelo de dados, precisamos ter em mente que ele, em algum momento será implementado em um banco de dados físico. Para controlar esse banco usamos as interfaces fornecidas pelo SGBD. Cada objeto de dados, geralmente, está associado a um esquema que faz parte de uma instância de banco de dados associada a um SGBD específico. Essa **instância** deve conter **uma coleção de esquemas** que recebe o nome de **catálogo**. Dentro destes esquemas temos um esquema especial, conhecido como **INFORMATION SCHEMA.** Ele proporciona informações sobre todos os esquemas do catálogo e todos os descritores de seus elementos.

Lembra do dicionário de dados que apresentamos na nossa aula introdutória de banco de dados? O **information schema** faz o papel de dicionário de dados! É nele que os metadados são armazenados!

A figura a seguir tenta organizar os conceitos acima de forma hierárquica. No topo da hierarquia temos uma instância do banco de dados denominada **Estratégia**. Abaixo temos diversos esquemas que fazem parte da coleção de esquemas. Veja que, para cada área de atuação do Estratégia, temos um esquema específico. Abaixo de cada um desses esquemas teremos **os objetos de esquema**, em especial as tabelas. Em um esquema a parte, temos o INFORMATION_SCHEMA, que descreve todos os objetos presentes na base de dados.

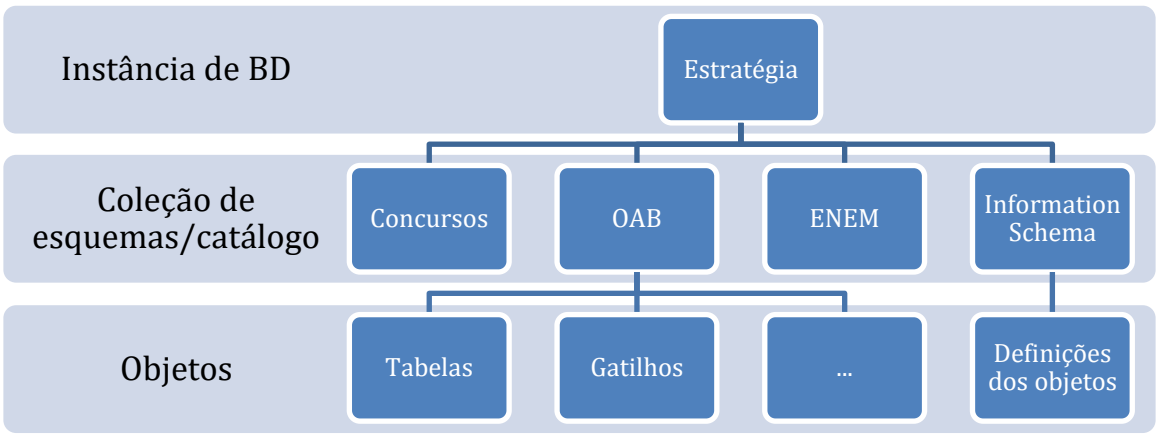


Figura 1- Hierarquia entre instâncias, esquemas e objetos de banco de dados.

O INFORMATION SCHEMA esquema contém definições para uma série de objetos de esquema, a maioria das informações podem ser acessadas por meio de visões. Uma **visão** é uma tabela virtual e derivada que permite visualizar os dados coletados a partir de tabelas reais. Ao usar essas visões, você pode exibir **as definições de objetos do catálogo como se fossem dados em SQL**.

E qual a importância de apresentar os dados por meio de visões**?? Você não pode alterar qualquer um dos dados nestas visões** - se você o fizesse, estaria alterando as definições de objetos de si -, mas você pode exibir informações simplesmente consultando a visão apropriada.

Veja que esse esquema serve como um provedor de informação sobre os objetos. Quer saber, por exemplo, quais as colunas da tabela aprovados? Escreva uma consulta usando o comando SELECT sobre a visão que lista as colunas do banco de dados e você receberá a informação sobre os nomes das colunas e respectivos tipos de dados.

Os demais esquemas estão relacionados aos seus **modelos de dados** e podem ser vistos como um **modo de agrupar** as tabelas e outros construtores que pertencem à mesma aplicação ou sistema. Cada esquema é identificado por um nome e inclui uma identificação de autorização, que indica o usuário ou a conta a qual o esquema pertence. Um esquema possui diversos elementos, tais como: tabelas, restrições, visões, domínios e outros construtores (como concessão de autoridade).

Já conhecemos a hierarquia, agora vamos conhecer os comandos para criação dos elementos presentes na hierarquia: banco de dados (database), esquemas (schema) e tabela.

A instrução **CREATE DATABASE** é usada para criar um banco de dados, embora o padrão ANSI não contenha essa instrução, quase todos os SGBDs comerciais aceitam esse comando. Ele cria uma instância do banco de dados. Por exemplo, se quisermos criar um banco de dados denominado ESTRATEGIA, usamos o seguinte comando:

```
CREATE DATABSE ESTRATEGIA;
```

Um servidor de banco de dados pode ter vários bancos de dados criados a partir de um comando semelhante ao descrito acima. Para exibir a lista de bancos de dados usando o comando **SHOW DATABASES.** É possível ainda remover um banco de dados usando o seguinte comando:

```
DROP DATABSE nome_do_banco_de_dados; -- Eu jamais iria apagar o banco de dados do ESTRATEGIA 😊
```

Para criar um SCHEMA é preciso que o privilégio para criação de esquemas seja explicitamente concedido. Geralmente apenas usuários relevantes, administradores de sistemas ou DBAs podem criar esquemas. Vamos criar um esquema usando o seguinte comando:

```
CREATE SCHEMA CONCURSOS;
```

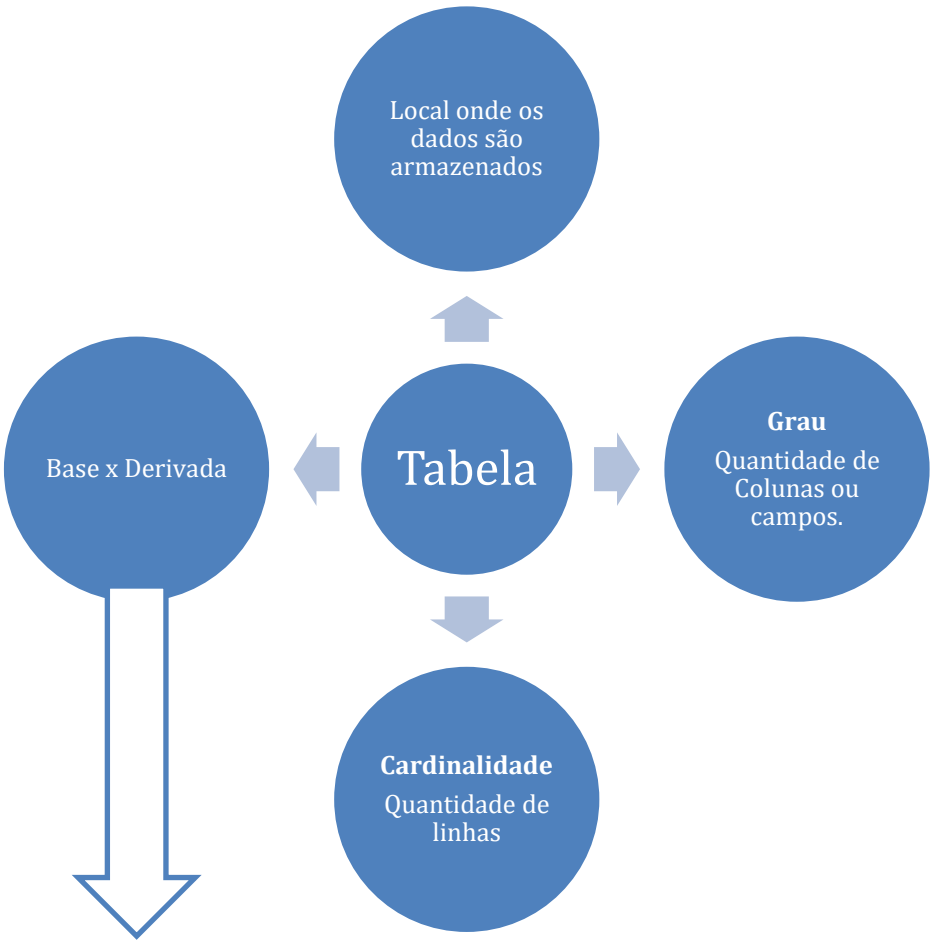
Depois de criar um esquema é possível removê-lo. Deve-se utilizar o comando **DROP SCHEMA**, seguido pelo nome do esquema a ser excluído. Podendo, ainda, utilizar as opções **RESTRICT** e **CASCADE** após o nome do esquema. Se a opção **CASCADE** for especificada, todos os objetos de esquema e de dados SQL dentro desses objetos são excluídos do sistema. Se a opção **RESTRICT** é usada, o esquema será excluído somente se não existir objetos de esquema. Segue um exemplo:

```
DROP SCHEMA CONCURSOS;
```

TRABALHANDO COM TABELAS

Vamos, agora, conversar um pouco sobre tabelas, são várias as questões que cobram o conteúdo deste assunto. A tabela é o **local onde todos os dados são armazenados**. Segundo a documentação do SQL/ANSI, as tabelas são coleções de linhas que possuem uma ou mais linhas. Uma linha é uma instância do tipo descrito pela tabela. Todas as linhas de uma tabela possuem o mesmo tipo.

O **grau** de uma tabela é o número de colunas existentes nessa tabela. Já o número de linhas é denominado **cardinalidade**. Uma tabela com cardinalidade zero é conhecida como vazia (*empty table*).



Outro conceito importante está relacionado ao fato de a tabela ser de base ou derivada (*base table ou derived table*). Uma **tabela base** pode ser uma tabela persistente (*presistence base*), temporária global *(global temporaty)*, temporária local criada (*created local temporary*) ou temporária global declarada *(declared local temporary)*. Todas essas tabelas **têm seus registros ou linhas armazenados na base de dados.**

Já uma **tabela derivada** está direta ou indiretamente associada a uma ou mais tabela por meio da avaliação de uma consulta. Uma tabela de visão (viewed table) é uma tabela derivada construída utilizando o comando de definição de visões **CREATE VIEW**. As tabelas de visão são comumente chamadas de visões. Uma visão cuja definição é feita sobre apenas uma tabela é considerada atualizável (*updatable*) se a chave primária fizer parte dos atributos da visão, bem como todos os atributos não nulos que não tiverem valores padrão (*default*) definidos.

Visões definidas sobre múltiplas tabelas usando operações de junção ou ainda definidas sobre apenas uma tabela, mas que possuam agrupamentos ou funções de agregação **não** são atualizáveis. Em outras palavras, não podem sofrer modificações nos seus valores bases.

Para criar uma tabela, basta seguir a sintaxe do **CREATE TABLE** apresentada a seguir. Neste exemplo, estamos criando uma tabela empregando vários atributos e descrevendo algumas restrições de integridade. Cada atributo possui um tipo de dado associado. Vejamos:

SINTAXE DO COMANDO

```
CREATE TABLE NOME_DA_TABELA (
       NOME_COLUNA1        TIPO_DE_DADO        RESTRIÇÕES   ,
       NOME_COLUNA2        TIPO_DE_DADO        RESTRIÇÕES   ,
       NOME_COLUNA3        TIPO_DE_DADO        RESTRIÇÕES   ,
       ...
       );
```

EXEMPLO DO COMANDO

```
CREATE TABLE ALUNO (
       NOME                VARCHAR(20)         NOT NULL     ,
       CPF                 INT                 PRIMARY KEY  ,
       SEXO                CHAR(1)             NOT NULL     ,
       DATA_NASCIMENTO     DATE                NOT NULL     ,
       CIDADE              VARCHAR(50)         NOT NULL     ,
       VALOR_PAGO          INT                 NOT NULL
       );
```

RESULTADO DO COMANDO

ALUNO					
NOME	CPF	SEXO	DATA_NASCIMENTO	CIDADE	VALOR_PAGO

Observem que você deve descrever as colunas que vão compor sua tabela, juntamente com os **tipos** e as **restrições de integridade** associados. Vejamos mais um exemplo, desta vez usando um SGBD específico. Para o nosso teste sugiro que você acesse o site http://sqlfiddle.com/ e tente escrever o comando de criação de tabela. Se você não tem tempo ou paciência, você pode copiar e colar apenas.

Lembra da tabela que criamos no início da nossa aula. Vamos então escrever o comando para criar a mesma:

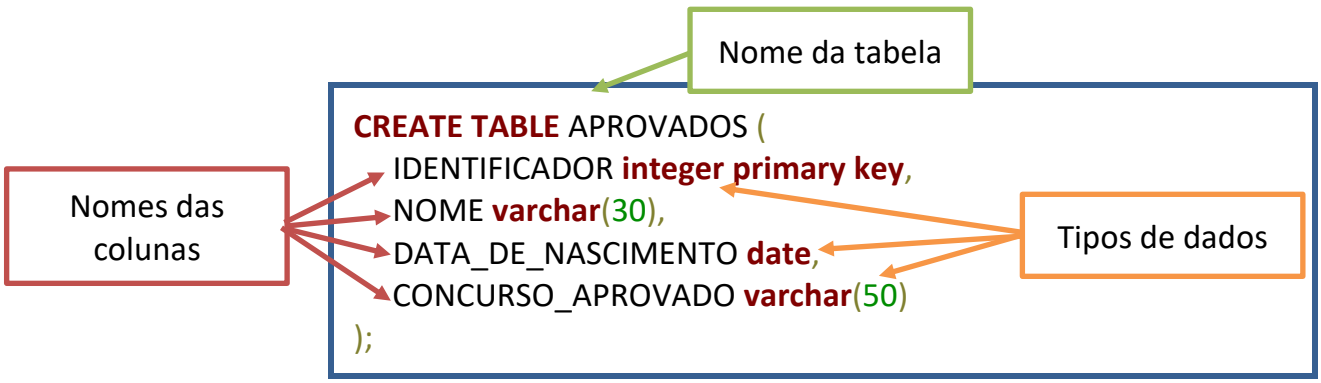


Figura 2 - Criando a tabela aprovados.

Sobre o comando **CREATE TABLE**, ele é usado para especificar uma nova relação, dando-lhe um **nome** e especificando seus **atributos** e **restrições iniciais**. Vejam que os **atributos** são definidos

primeiro e a cada atributo é dado **um nome**, **um tipo** para especificar o **domínio** de seus valores e alguma **restrição de atributo**. As restrições de **chave**, de **integridade,** de entidade e de integridade referencial **podem** ser específicas no mesmo comando após os atributos serem declarados ou poderão ser adicionadas depois, usando o comando **ALTER TABLE.**

HORA DE PRATICAR!

(Ministério da Economia – Especialista em Ciência de Dados - 2020) Julgue os itens a seguir, a respeito de conceitos de SQL.

O comando CREATE DATABASE TAB é utilizado para criar uma tabela em um banco de dados.

Comentários: A instrução CREATE DATABASE é usada para criar um banco de dados SQL. Para criar tabelas usamos o comando CREATE TABLE.

Gabarito: ERRADO.

(Ministério da Economia – Desenvolvimento de Sistemas - 2020)

Tendo como referência o diagrama de entidade relacionamento precedente, julgue próximo item, a respeito de linguagem de definição de dados e SQL.
As expressões DDL a seguir permitem a criação das tabelas presentes no diagrama apresentado.

```
create table aluno (
id integer primary key,
nome varchar(40) );

create table disciplina (
id integer primary key,
descricao varchar(60) );

create table matricula (
aluno integer,
disciplina integer,
ano integer,
nota numeric,
constraint pk_matricula primary key (aluno, disciplina, ano),
constraint fk_matricula_aluno foreign key (aluno) references aluno,
```

constraint fk_matricula_disciplina foreign key (disciplina) references disciplina);

_____

Comentários: O comando CREATE TABLE criará uma tabela inicialmente vazia no banco de dados atual. O nome ou identificador exclusivo da tabela segue a instrução CREATE TABLE. Em seguida, entre colchetes, vem a lista que define cada coluna da tabela e seu respectivo tipo. Por fim, podemos adicionar as restrições de integridade na própria coluna ou ao final da definição da tabela por meio da palavra-chave constraint. **Perceba que os comandos funcionam perfeitamente e estão de acordo com o padrão SQL ANSI.**

Gabarito Certo.

Outro comando importante é o comando **DROP TABLE** que remove todas as informações de uma relação do banco de dados. Em outras palavras, o comando exclui o objeto e os dados armazenados. É possível usar o modificador **CASCADE** para deletar também tabelas que sejam filhas da tabela removida ou pelo menos ajustar a integridade referencial. Vejamos a sintaxe do comando e um exemplo.

SINTAXE DO COMANDO

```
DROP TABLE NOME_DA_TABELA;
```

EXEMPLO DO COMANDO

```
DROP TABLE ALUNO;
```

RESULTADO DO COMANDO

Na alteração de tabelas feita pelo comando **ALTER TABLE,** podemos adicionar e excluir atributos de uma relação. É possível definir um valor default para os novos atributos ou eles vão receber valores nulos para as tuplas já existentes. Vejam, abaixo, um resumo da sintaxe do ALTER TABLE.

SINTAXE DO COMANDO

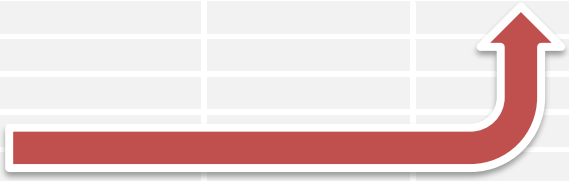
```
ALTER TABLE NOME_DA_TABELA ADD [COLUMN] NOME_COLUNA1 TIPO_DE_DADO;
ALTER TABLE NOME_DA_TABELA DROP [COLUMN] NOME_COLUNA1 {CASCADE|RESTRICT};
ALTER TABLE NOME_DA_TABELA ALTER COLUMN NOME_COLUNA1 TIPO_DE_DADO {SET DEFAULT VALOR_PADRAO|DROP
DEFAULT};
```

EXEMPLO DO COMANDO

```
ALTER TABLE ALUNO ADD COLUMN EMAIL VARCHAR(40);
ALTER TABLE ALUNO DROP COLUMN SEXO;
ALTER TABLE ALUNO ALTER COLUMN VALOR_PAGO FLOAT;
```

RESULTADO DO COMANDO

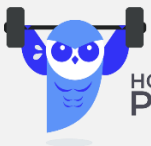
NOME	CPF	EMAIL	DATA_NASCIMENTO	CIDADE	VALOR_PAGO



Resumindo os principais comandos usadas para definição de objetos em SQL, conhecidos como DDL (Data Definition Language) são:

COMANDO	DESCRIÇÃO
CREATE	Comando utilizado para criar bancos de dados, tabelas, índices, entre outros.
DROP	Comando utilizado para deletar uma tabela do banco de dados.
TRUNCATE	Comando utilizado para apagar os dados de uma tabela do banco de dados.
ALTER	Comando utilizado para adicionar, deletar ou modificar colunas do banco de dados.
RENAME[1]	Comando utilizado para renomear uma tabela do banco de dados.

Para entendermos melhor como escolher os tipos de dados das nossas colunas, passaremos, a seguir, a analisar os principais tipos de dados de SQL e suas peculiaridades. Antes, porém, gostaria de apresentar uma questão sobre o assunto.



Ano: 2017 Órgão: UFRN Prova: Engenheiro - Neuroengenharia

Para criar e remover uma tabela em um banco de dados, os comandos SQL são, respectivamente,

a) CREATE TABLE e DROP TABLE.
b) NEW TABLE e DELETE TABLE.
c) CREATE SCHEMA e DROP SCHEMA.
d) NEW SCHEMA e DELETE SCHEMA.

Comentário: Como eu falei no início da aula, não estamos focados em uma banca específica, mas sim no seu aprendizado. A questão acima apresenta os dois principais comandos que acabamos de tratar, sem questionar detalhes sobre a sintaxe do comando. Lembre-se de que o comando para criação de tabelas é o CREATE TABLE e para a sua remoção temos o comando DROP TABLE. Sendo assim, podemos marcar nossa resposta na alternativa A. O outro comando que falamos altera a estrutura da tabela, e tem como sintaxe inicial os termos ALTER TABLE.

Gabarito: A.

Antes de seguir em frente com o nosso conteúdo, gostaria de apresentar um esquema com os principais comandos DDL e sua respectiva sintaxe.

[1] O comando RENAME é uma extensão do padrão SQL ANSI implementada por alguns SGBDs como o SQL Server da Microsoft, o DB2 da IBM e MySQL.

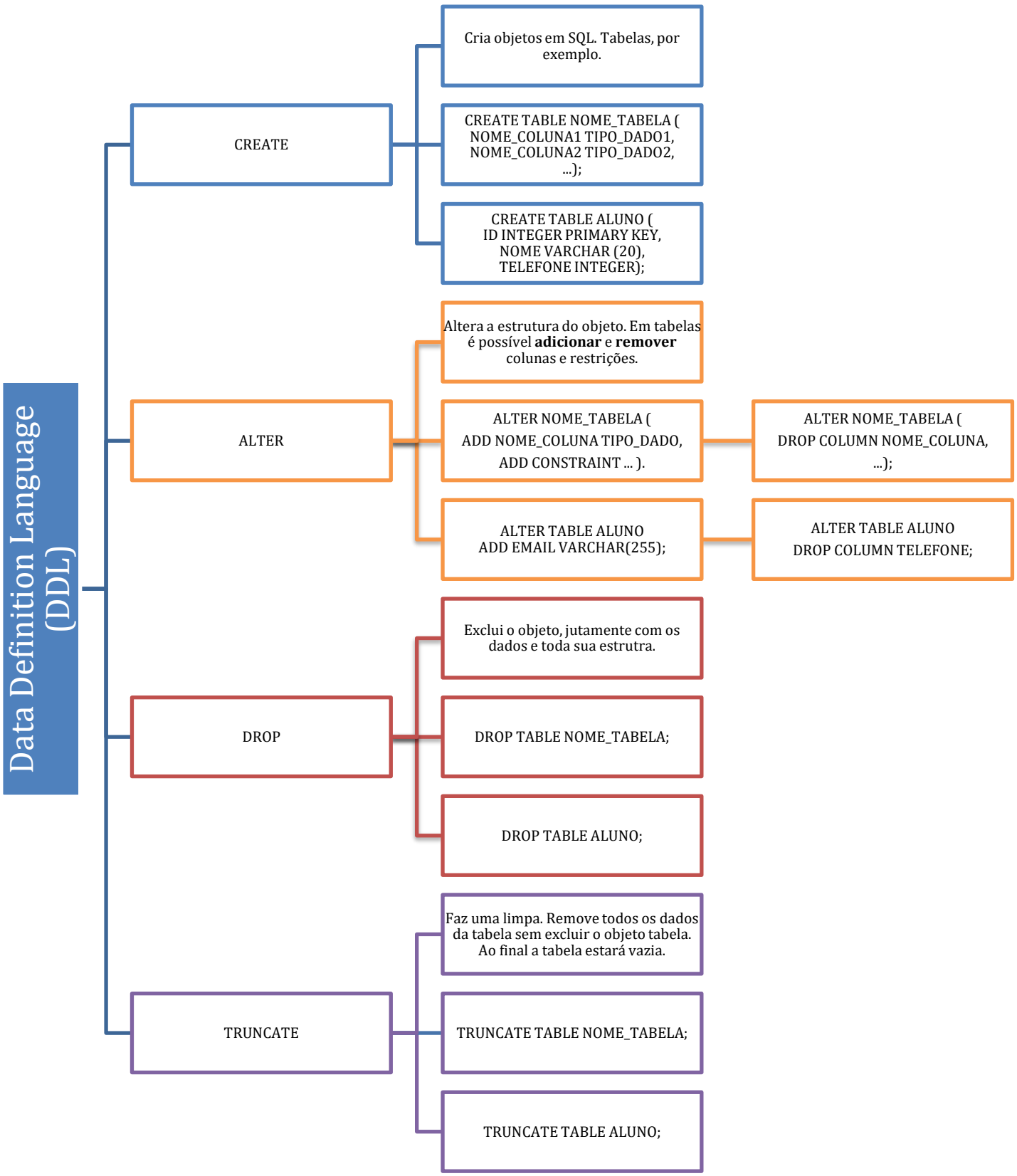


Figura 3 - Resumo dos comandos DDL.

TIPOS DE DADOS

Pessoal, vocês devem se lembrar que características de uma entidade são representadas no modelo relacional por meio de colunas de uma tabela no banco de dados. **Para que você crie uma coluna, você deve especificar qual tipo de dados será armazenado nela**.

SQL inclui **diversos** tipos de tipos de dados **pré-definidos**: string (conjunto de caracteres), numéricos, binário, datetime, interval, boolean e XML. Os tipos de dados são usados na instrução CREATE TABLE como parte das definições da coluna:

```
CREATE TABLE <tablename>(
<column_name> <data_type> ... ,
<column_name> <data_type> ... ,
... );
```

Primeiramente, vamos falar dos tipos de dados formados por cadeias de caracteres.

TEXTO (CHARACTER)

No SQL padrão, esses tipos são divididos em tamanho fixo e variável. Todas as cadeias de caracteres em SQL podem ser de **comprimento fixo ou variável**. Você tem três tipos principais de cadeias de caracteres:

1. Cadeias de caracteres de tamanho fixo (**CHARACTER** ou **CHAR**)

2. Cadeias de caracteres de tamanho variável (**CHARACTER VARYING** ou **VARCHAR**)

3. Cadeia de caracteres para armazenar grandes objetos (**CHARACTER LARGE OBJECT** ou **CLOB**).

Como você pode imaginar, essa flexibilidade tem um preço de desempenho, pois o SGBD deve executar a tarefa adicional de alocação dinâmica quando temos conjuntos de caracteres de tamanho variável.

Um **CHAR ou CHARACTER** especifica **o número exato de caracteres** que serão armazenados para cada valor. Por exemplo, se você definir o comprimento de 10 caracteres, mas o valor contém apenas seis caracteres, os quatro caracteres restantes serão completados com **espaços em branco**. Um exemplo da sintaxe do comando seria:

```
CONCURSO_NOME CHAR (100).
```

O outro tipo de cadeia de caracteres é o **CHARACTER VARYING ou VARYING CHAR ou VARCHAR**. Ele especifica **o número máximo de caracteres** que pode ser incluído em uma variável. O número de caracteres armazenados é exatamente o mesmo número do valor introduzido, de modo que nenhum espaço será adicionado ao valor. Um exemplo da definição de uma coluna deste tipo:

```
CONCURSO_NOME VARCHAR (60).
```

Observem que o comprimento da cadeia é passado entre parênteses.

O tipo de dados CHARACTER LARGE OBJECT (CLOB) foi introduzido juntamente com o SQL:1999. Como o próprio nome sugere, ele é usado para armazenar **grandes cadeias de caracteres** que não conseguem ser alocadas no tipo CHARACTER. CLOBs se comportam como cadeias de caracteres comuns, mas há uma série de restrições sobre o que você pode fazer com eles.

Por exemplo, um CLOB **não** pode ser usado em uma **chave primária**, **chave estrangeira** ou com predicado **UNIQUE**.

Vários idiomas têm alguns caracteres que diferem de quaisquer caracteres em outro idioma. Por exemplo, o alemão tem alguns caracteres especiais não presentes no conjunto de caracteres do idioma inglês. Você pode especificar o idioma inglês definindo-o como o padrão para o seu sistema, mas você pode usar outros conjuntos de caracteres alternativos; para isso existe o NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, e NATIONAL CHARACTER LARGE OBJECT. Eles são tipos de dados com a mesma função que o CHARACTER, CHARACTER VARYING e CHARACTER LARGE OBJECT - a única diferença é que **o conjunto de caracteres que você está especificando é diferente do conjunto de caracteres padrão.**

Você pode especificar o conjunto de caracteres quando define uma coluna de uma tabela. Se você quiser, cada coluna pode usar um conjunto de caracteres diferente. O exemplo a seguir mostra a criação de uma tabela e usa vários conjuntos de caracteres:

```
CREATE TABLE IDIOMAS (
  IDIOMA_1 CHARACTER (40),
  IDIOMA_2 CHARACTER VARYING (40) CHARACTER SET GREEK,
  IDIOMA_3 NATIONAL CHARACTER (40),
  IDIOMA_4 CHARACTER (40) CHARACTER SET KANJI
  ) ;
```

Aqui a coluna IDIOMA_1 contém caracteres em conjunto de caracteres padrão do sistema. A coluna IDIOMA_3 contém caracteres em conjunto de caracteres nacional do sistema. A coluna IDIOMA_2 contém caracteres gregos. E a coluna IDIOMA_4 contém caracteres Kanji. Depois de uma longa ausência, conjuntos de caracteres asiáticos, como Kanji, estão, agora, disponíveis em muitos produtos.

Para finalizar vamos observar um comando de criação de tabela com os principais tipos de dados de caracteres associados às colunas. Vejamos:

```
CREATE TABLE teste (
  id   DECIMAL PRIMARY KEY,        -- esse não é texto, é numérico.
  col1  CHAR(8),                   -- exatamente 8 caracteres
  col2  VARCHAR(100),                  -- até 100 caracteres
  col3  CLOB                       -- strings muito longas
);
```

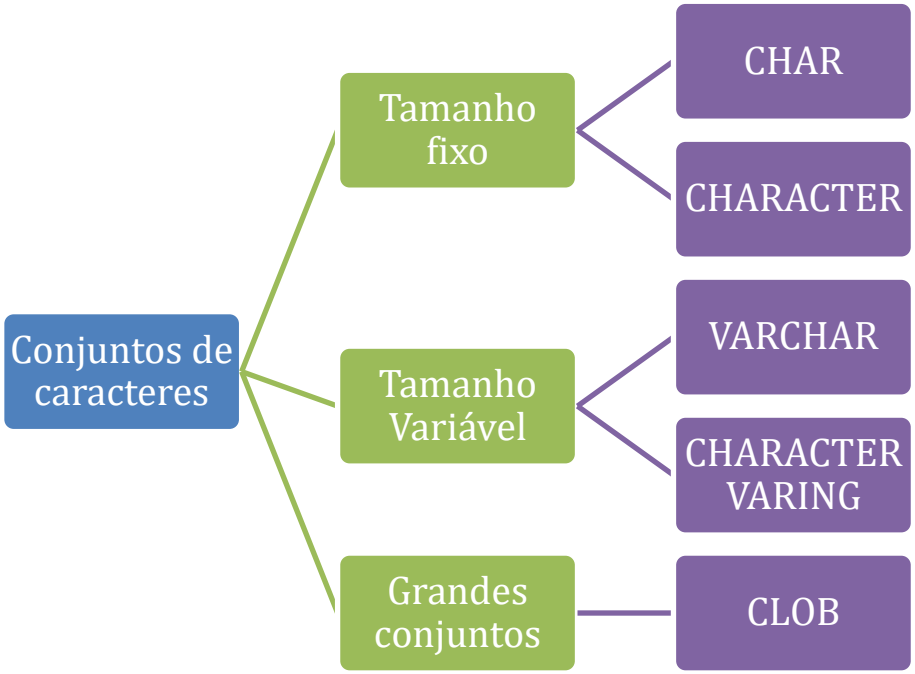


Figura 1 - Tipos de dados textuais

BINARY STRING

Ainda dentro das strings temos as **strings binárias**. **Uma string binária é uma sequência de bytes da mesma forma que uma string de caracteres é uma sequência de caracteres**. Ao contrário das strings de caracteres que geralmente contêm informações na forma de texto, ela é usada para armazenar dados não tradicionais (não estruturados), tais como imagens, áudio e arquivos de vídeo, executáveis de programa, e assim por diante.

Até o SQL 2003, os tipos de dados binários eram representados pelas palavras chaves bit e bit varing. Sendo que o bit representar os conjuntos binários de tamanho fixo, já o bit varing representava o conjunto de bits de tamanho variável. SGBDs como o PostgreSQL, mantém esses tipos de dados até hoje.

Os tipos de dados de strings binárias foram introduzidos no SQL:2008. Considerando que os dados binários têm sido fundamentais para computadores digitais desde o Atanasoff-Berry Computer (primeiro computador eletrônico digital) da década de 1930, esse reconhecimento da importância dos dados binários parece vir um pouco tarde para SQL. Antes tarde do que nunca! Existem três tipos diferentes BINARY, BINARY VARYING e BINARY LARGE OBJECT. BINARY LARGE OBJECT é conhecido como BLOB. O BLOB armazena grandes grupos de bytes, até o valor especificado.

Se você definir o tipo de dados de uma coluna como **BINARY**, poderá especificar o número de **bytes (octetos)** que a coluna contém usando a sintaxe BINARY(n), onde n é o número de bytes. Se você especificar o tipo de dados de uma coluna como, por exemplo, uma string binária deve ter 16 bytes de comprimento, os dados devem ser inseridos como bytes. Vejamos o exemplo:

```
...
nomedaColuna BINARY(16),
...
```

Outra opção é usar o tipo BINARY VARYING ou VARBINARY quando o comprimento de uma string binária **for variável**. Para especificar este tipo de dados, use VARBINARY (x) onde x é o número **máximo de bytes permitidos**. Perceba que o tamanho mínimo da string é zero e o tamanho máximo é *x*. Vejamos um exemplo:

```
...,
nomeDaColuna01 VARBINARY (20),
...
```

O tipo de dados BINARY LARGE OBJECT(BLOB) é usado com cadeias binárias enormes que são muito grandes para os tipos BINARY. Imagens gráficas e arquivos de música são exemplos de enormes cadeias binárias. BLOBs se comportam de maneira muito semelhante às strings binárias comuns, mas **o SQL impõe algumas restrições sobre o que você pode fazer com eles**.

BLOBs seguem a mesma lógica dos CLOBs, você **não** pode usar uma coluna BLOB em uma CHAVE PRIMÁRIA ou ESTRANGEIRA. Também não é possível usar a constraint UNIQUE em uma coluna cujo tipo é um BLOB. Além disso, não são permitidos BLOB em comparações além das de igualdade ou desigualdade.

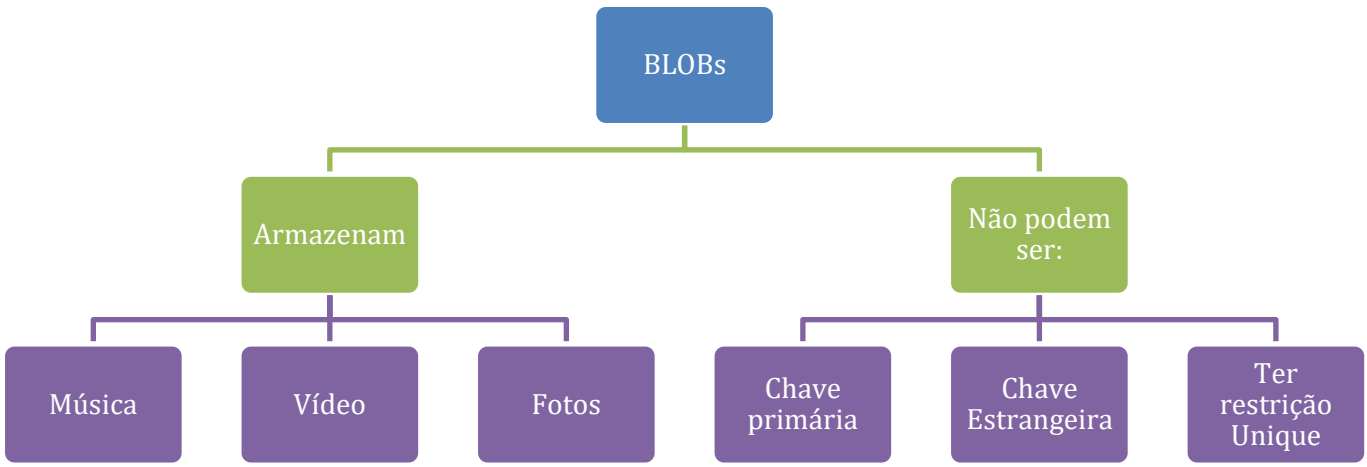


Figura 2 - Esquematizando os BLOBs

TIPOS NUMÉRICOS

Vamos, agora, tratar dos tipos de dados numéricos. Eles são divididos em duas grandes categorias: **números exatos (*exact numbers*)** e **números aproximados (*approximate*)**.

Os **números exatos** podem ser números **inteiros** (lápis, pessoas ou planetas) ou ter **casas decimais** (preços, pesos ou percentuais). Os números podem ser **positivos ou negativos**. Eles também podem ter **precisão e escala**. O que seria isso? Vejamos ...

A **precisão (p)** determina o número total máximo de dígitos decimais que podem ser armazenados (tanto à esquerda quanto à direita do ponto decimal). E **escala (e)** especifica o número máximo de casas decimais permitidas. SQL permite criação de colunas com os tipos NUMERIC e DECIMAL. Quando usamos, numeric(p,e), informamos, respectivamente a precisão e a escala. Veja o exemplo abaixo em que o número 235,89:

Tabela 1 - A tabela abaixo representa exemplos de precisão e escala para o número 235,89.

Especificação	Valor armazenado
NUMERIC(5)	236
NUMERIC(5,0)	236
NUMERIC(5,1)	235,9
NUMERIC(5,2)	235,89
NUMERIC(4,0)	236
NUMERIC(4,1)	235,9
NUMERIC(4,2)	Excede a precisão
NUMERIC(2,0)	Excede a precisão

Os números de pontos flutuantes ou aproximados são números que não podem ser representados com precisão absoluta. O tipo de dado *float*, representado por *float(p)*, é um valor numérico aproximado **com precisão binária**, ou seja, o número de dígitos binários necessários para o armazenamento do valor aproximado. O valor máximo da <precisão> é definido pela implementação e não deve ser maior do que o valor máximo definido na implementação do SGBD.

Vamos imaginar um exemplo. Suponha que exista uma coluna col_float na nossa base de dados e que essa coluna foi definida com o tipo de dado float(20). O valor 20 nos informa que é possível armazenar seu número em até 20 bits. Vamos então armazenar o número 235,89 nesta coluna. São utilizados oito dígitos binários para armazenar a parte inteira (235 (decimal) = 1110 1011 (binário)) e sete para armazenar a parte fracionária (89 (decimal) = 101 1001 (binário). Assim, temos 8+7=15 bits. Esse valor será inteiramente gravado na coluna do banco.

Agora vamos pensar em um outro número, imagine o saldo na conta do prof. Valley Carvalhus, R$ 116.456,23. Para armazenar a parte inteira precisamos de 17 bits, 116456 (decimal) = 1 1100 0110 1110 1000 (binário). Sobrando, portanto, 3 bits para armazenar o valor fracionário. Como 23 em decimal é equivalente a 10111 em binário, não temos espaço para armazenar esse valor. Assim o

SGBD vai arredondar o número 0,23 para 0,2, truncando a segunda casa decimal. Neste caso o número 2, que em binário é 10, pode ser armazenado em 3 bits. Logo, o valor gravado no banco de dados será 116.456,2. Lógico que esses 3 centavos não farão diferença para o professor, mas farão você entender o funcionamento do parâmetro passado na criação da variável float.

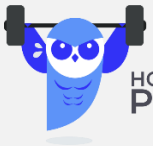
Deu para entender? 😊

Esses tipos de dados numéricos estão disponíveis em todos os SGBDs relacionais, por exemplo, no SGBD Oracle. Vejam, abaixo, a relação entre os tipos SQL padrão, os do ORACLE e do MYSQL, bem como alguns comentários importantes sobre os aspectos numéricos do Oracle.

ANSI SQL DATA TYPE	ORACLE DATA TYPE	MYSQL DATA TYPE
NUMERIC [(p,s)] DECIMAL [(p,s)] DECFLOAT (novo do SQL 2016)	NUMBER [(p,s)]	DECIMAL(SIZE,D) DEC(SIZE,D)
INTEGER INT SMALLINT BIGINT	NUMBER(p,0) NUMBER(p)	TINYINT SMALLINT MEDIUMINT INT INTEGER BIGINT
FLOAT DOUBLE PRECISION REAL	FLOAT (p) NUMBER[(p,s)]	FLOAT(size,d) FLOAT(p) DOUBLE (size,d))

Os tipos de dados NUMERIC e DECIMAL do SQL ANSI são representados pelo tipo NUMBER no ORACLE. Eles só podem ser definidos como números de ponto fixo. E, para esses tipos de dados, o valor padrão para a escala é zero (0). O tipo de dado FLOAT é um tipo de ponto flutuante com precisão binária. O valor default para a precisão neste tipo de dados é de 126 casa binárias ou 38 casas decimais.

Perceba que utilizamos o tipo FLOAT do ORACLE para representar os tipos DOUBLE PRECISION e REAL do SQL ANSI. O tipo de dados DOUBLE PRECISON é também um número de ponto flutuante como precisão binária de 126. E, por fim, o tipo REAL é um ponto flutuante com precisão de 63 casas binárias ou 18 casas decimais. Esses valores são informados como parâmetros, como observamos na tabela acima.



ANO: 2013 PROVA: TRT - 12ª REGIÃO (SC) - TÉCNICO JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO

Disciplina: Banco de Dados (TI) - Assuntos:

NÃO é um dos tipos de dados nativos do Oracle:

a) LONG.

b) BINARY_DOUBLE

c) NUMERIC.

d) BINARY_FLOAT.

e) ROWID.

Comentário. Essa questão faz uma remissão ao assunto que acabamos de estudar. Percebam a importância de conhecer os tipos de dados e as diferenças em relação ao padrão SQL ANSI. Neste caso, temos um tipo de dados, o NUMERIC, que não é um tipo de dados padrão do Oracle, mas é do SQL ANSI. Sendo assim, nossa resposta está presente na alternativa C.

Gabarito: C.

Vamos observar agora um exemplo de comando com a descrição de vários atributos do tipo numérico.

```
CREATE TABLE test (
      id    DECIMAL PRIMARY KEY,
      col1  DECIMAL(5,2),          -- 3 dígitos antes e 2 dígitos depois da
                                      vírgula   decimal
      col2  SMALLINT,             -- Apenas valores inteiros.
      col3  INTEGER,              -- Apenas valores inteiros.
      col4  BIGINT,               -- Apenas valores inteiros.
      col5  FLOAT(24),            -- pelo SQL/ANSI você vai utilizar 24 bits
para representar o número, começando pelos dígitos a esquerda da casa decimal.
      col6  REAL,
      col7  DOUBLE PRECISION );
```

Para concluir o nosso estudo sobre tipos de dados numéricos, apresentamos uma tabela que relaciona os tipos com o espaço de armazenamento necessário para cada valor associado, e o range, que seria os valores possíveis ou o domínio de um determinado tipo numérico.

Tipo de dado	Tamanho em Bytes	Range

SMALLINT	2	-32768 até 32768
INTERGER	4	-2.147.483.648 até +2.147.483.648
BIGINT	8	-9.223.372.036.775.808 até +9.223.372.036.775.808
REAL	4	- 3,40E + 38 a -1,18E - 38, 0 e 1,18E - 38 a 3,40E + 38
FLOAT	4 até 8 bytes	- 1,79E+308 a -2,23E-308, 0 e 2,23E-308 a 1,79E+308
DOUBLE	8	±1.79769313486231570E+308

DATETIME E INTERVAL

Que horas são? Qual a data que você está lendo esta aula? As horas e a data são informações importantes e que precisam ser armazenadas com frequência nos bancos de dados. Primeiro trataremos do tipo DATE.

DATE é uma estrutura que consiste em três elementos: ano, mês e dia. O ano é um número de quatro dígitos que permite que os valores de 0000 a 9999. O mês é um elemento de dois dígitos com valores de 01 a 12, e dia que tem dois dígitos com um intervalo de 01 a 31. SQL/ANSI define a semântica de data usando a estrutura descrita acima, mas os SGBDs não são obrigados a usar essa abordagem, desde que a implementação produza os mesmos resultados.

O tipo TIME consiste das partes hora, minuto e segundo. A hora é um número de 00 a 23. O minuto é um número de dois algarismos, de 00 a 59. O segundo é um inteiro entre 00-59 ou um número decimal com uma precisão mínima de cinco e escala mínima de três que pode conter valores de 00.000 para 59.999.

Temos ainda os tipos:

1. DATETIME que combina data e hora em um único tipo, com intervalo de datas.

2. TIMESTAMP que engloba os campos DATE e TIME, mais seis posições para a fração decimal de segundos e uma qualificação opcional WITH TIME ZONE (que especifica o fuso horário) e

3. INTERVAL que serve para calcular o intervalo entre dois objetos que representam tempos. Vejamos um exemplo de criação de uma tabela cujas colunas são dos tipos temporais.

```
CREATE TABLE tempo (
      id   DECIMAL PRIMARY KEY,
      col1 DATE,                    -- armazena ano, mês e dia
      col2 TIME,                    -- armazena data, hora e segundos
      col3 TIMESTAMP(9),            -- armazena um selo de tempo.
```

```
    col4 TIMESTAMP WITH TIME ZONE    -- exemplo de timestamp com timezone
);
```

BOOLEAN E XML

O tipo de dados BOOLEAN consiste na verdade dos valores distintos, verdadeiro e falso, assim como o valor desconhecido. O valor verdadeiro ou falso booleano pode ser comparado com um valor NULL.

XML é um acrônimo para eXtensible Markup Language, que define um conjunto de regras para a adição de marcação aos dados. As estruturas de marcação permitem aos dados transmitir o que eles significam. XML permite o compartilhamento de dados entre diferentes plataformas.

O tipo de dados XML tem uma estrutura de árvore, assim, um nó raiz pode ter nós filho, o que pode, por sua vez, ter seus próprios filhos. Introduzido pela primeira vez no SQL:2003, o tipo XML foi concretizado na versão SQL/XML: 2005, e ainda mais melhorado no SQL:2008. Podem armazenar até 2GB. Vejamos um exemplo de como criar uma tabela com um coluno do tipo XML.

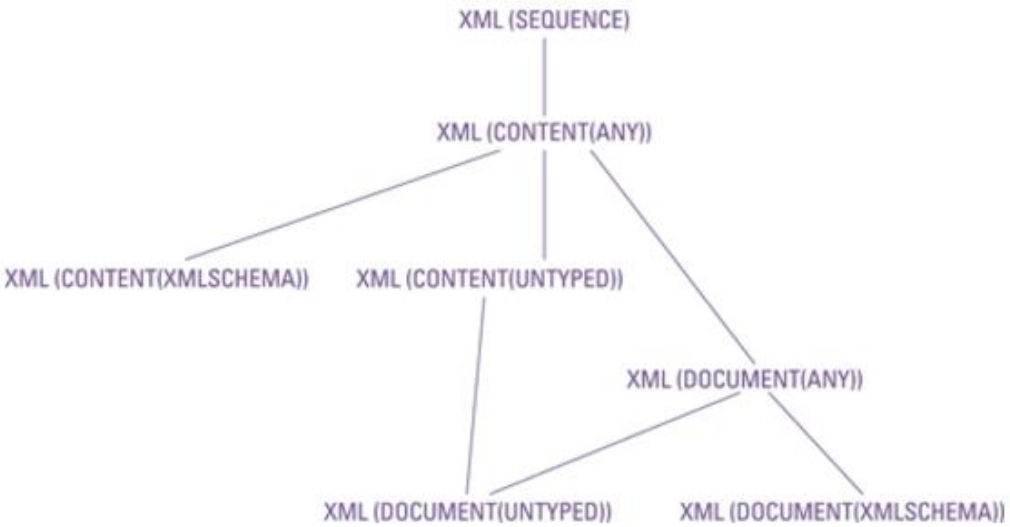
```
CREATE TABLE T1(
Col1 int primary key,
Col2 xml);
```

Um modificador de tipo para XML associa um conjunto de um ou mais esquemas XML ao tipo de dados XML. Este modificador de tipo impõe que todos os documentos XML armazenados em uma coluna XML sejam validados de acordo com um dos esquemas XML especificados no modificador de tipo.
Os modificadores primários do tipo XML são SEQUENCE, CONTENT e DOCUMENT. Os modificadores secundários são UNTYPED, ANY e XMLSCHEMA. Vejamos um exemplo de uma tabela criada com um tipo XML:

```
CREATE TABLE ESTRATEGIAXMLEXEMPLO(
 ID INT NOT NULL,
 CONTENT XML (XMLSCHEMA URI 'http://www.example.com/Estrategia'
 LOCATION 'http://www.example.com/ Estrategia.xsd'));
```

Veja que o modificador primário CONTENT descreve uma coluna do tipo XML que deve ser validada por um XMLSCHEMA.
A figura mostra a estrutura de árvore que ilustra as relações hierárquicas entre os subtipos. Uma sequência é uma coleção ordenada de zero ou mais itens. Um item é um valor atômico ou um nó. Uma sequência não pode ser membro de outra sequência. Perceba que ela está na raiz da árvore abaixo. Dentro da sequência é possível ter vários conteúdos em formato XML. Estes, por sua vez, podem ser

validados por um XMLSCHEMA, não ter um tipo definido (UNTYPED) ou serem um documento XML.



Por fim, é importante lembrar que **o valor nulo é membro de todos os tipos de domínios**. Se você declarar o domínio de um atributo como NOT NULL, o SGBD vai proibir a inserção de valores nulos para essa coluna.

Tipos ROW, Collections e UDT

Além dos tipos predefinidos, embutidos, SQL suporta tipos de coleção, tipos construídos e tipos definidos pelo usuário (UDT).

Tipos ROW

O tipo de dados de linha foi introduzido com o SQL:1999. Uma coisa notável sobre o tipo de dados ROW é que ela viola as regras de normalização declaradas por Codd nos primeiros dias da teoria de banco de dados relacional. Uma das características que definem a primeira forma normal é que um atributo de uma linha da tabela não pode ter um valor múltiplo. Um campo pode conter um e apenas um valor, ou seja, é atômico. No entanto, o tipo de dados ROW permite que você declare uma linha inteira de dados contida dentro de um único campo de uma linha da tabela, em outras palavras, uma linha dentro de outra.

As formas normais, definidas por Codd, são características que definem a estrutura para bancos de dados relacionais. A inclusão do tipo ROW no padrão SQL foi a primeira tentativa de ampliar SQL além do modelo relacional puro. Considere a seguinte instrução SQL, que define um tipo ROW para informações sobre o endereço de uma pessoa:

```
CREATE ROW TYPE tipo_endereco (
        Rua                 CHARACTER VARYING (25),
        Cidade              CHARACTER VARYING (20),
        Estado      CHARACTER (2),
        CEP     CHARACTER VARYING (9)
```

```
) ;
```

Depois que ele é definido, um novo tipo de linha pode ser usado em uma definição de tabela:

```
CREATE TABLE CLIENTE (
        ClienteID           INTEGER     PRIMARY KEY,
        PrimeiroNome        CHARACTER VARYING (25),
        ÚltimoNome          CHARACTER VARYING (20),
        Endereco            addr_typ,
        Telefone            CHARACTER VARYING (15)
) ;
```

A vantagem aqui é que se você está mantendo informações de endereço para diferentes entidades - como clientes, fornecedores, colaboradores e acionistas - você tem que definir os detalhes da especificação de endereço apenas uma vez: na definição do tipo ROW.

Tipos de coleção

Após SQL sair da fronteira do modelo relacional com o SQL:1999, os tipos de dados que violam a primeira forma normal tornaram-se possíveis. Um campo pode conter uma coleção de objetos, em vez de apenas um. O tipo ARRAY foi introduzido no SQL:1999, e o tipo MULTISET foi introduzido no SQL:2003 contribuem com essa capacidade.

Duas coleções podem ser comparadas entre si somente se ambas são do mesmo tipo, ou ARRAY ou MULTISET, e se os seus tipos de elementos são compatíveis. Como ARRAYs têm uma ordem para os elementos definidos, elementos correspondentes podem ser comparados. MULTISETS não tem nenhuma ordem definida para os elementos, mas você pode compará-los, se (a) existe uma enumeração sobre cada um deles e (b) as enumerações podem ser emparelhadas.

Tipos definidos pelo usuário (UDT – User defined types)

Tipos definidos pelo usuário (UDTs) representam outro exemplo de recursos que chegaram no SQL:1999 que vêm do mundo de programação orientada a objetos. Como um programador de SQL, você não está mais restrito aos tipos de dados definidos na especificação SQL. Você pode definir seus próprios tipos de dados, usando os princípios de tipos de dados abstratos (ADTs) encontrados em tais linguagens de programação orientada a objetos como C++.

Um dos benefícios mais importantes dos UDTs é o fato de que você pode usá-los para eliminar a diferença de impedância entre o SQL e a linguagem de host que está "envolvendo" o SQL. Um problema de longa data com o SQL tem sido o fato de tipos de dados predefinidos do SQL não correspondem aos tipos de dados das linguagens host no qual as instruções SQL são incorporadas. Agora, com UDTs, um programador de banco de dados pode criar tipos de dados dentro do SQL que correspondem aos tipos de dados da linguagem de host.

UDT tem atributos e métodos, que são encapsulados. O mundo exterior pode ver as definições de atributos e os resultados obtidos pelos métodos - mas as implementações específicas dos métodos estão escondidas. O acesso aos atributos e aos métodos de uma UDT pode ser ainda mais restrito, podemos especificá-los como públicos, privados ou protegidos. Vejamos um exemplo de um UDT:

```
CREATE TYPE livro_udt AS                    -- o nome da UDT será livro_udt
titulo CHAR(40),
precodecompra DECIMAL(9,2),
ISBN varchar(40);
```

ESQUEMATIZANDO

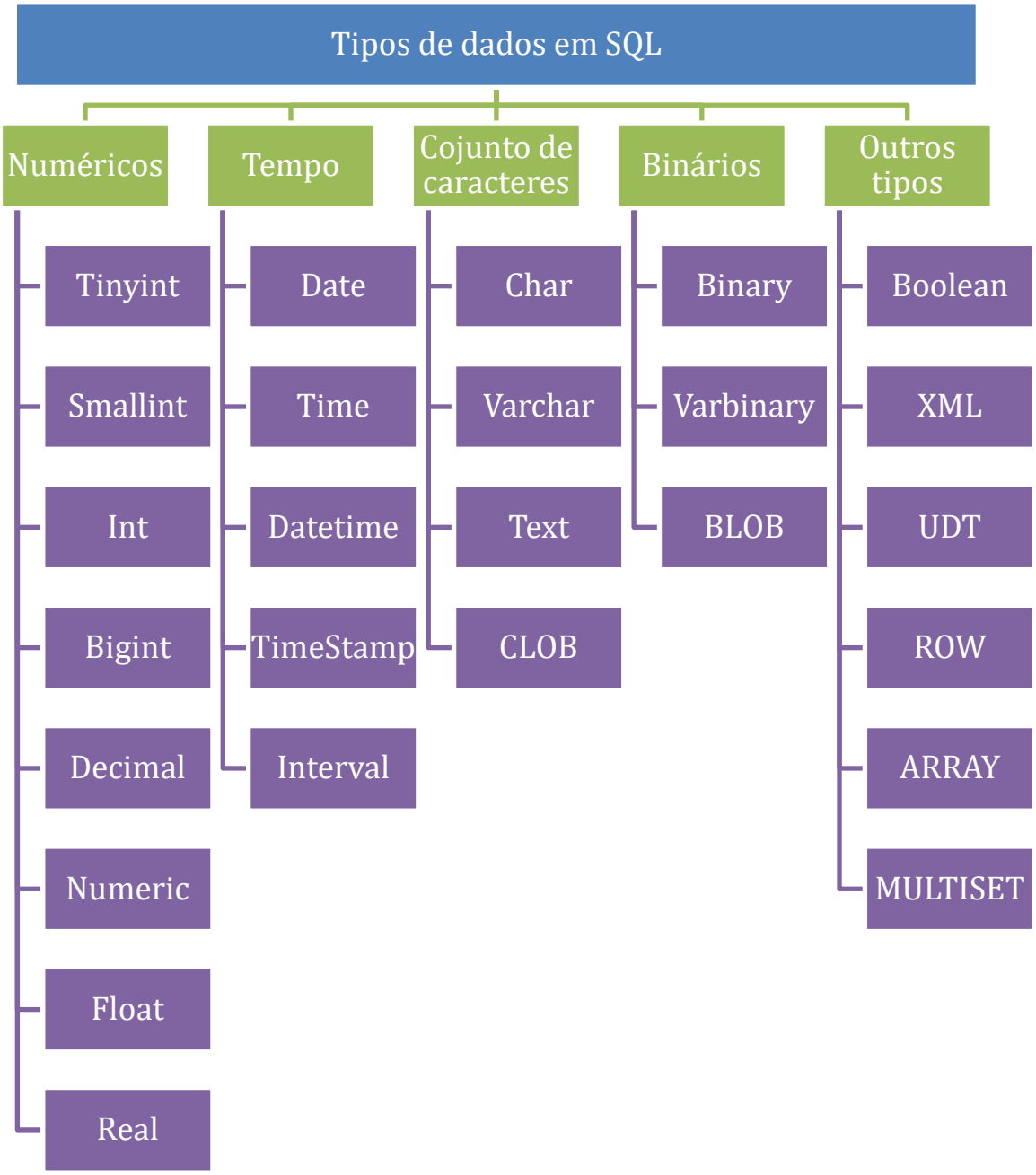


Figura 3 - Resumo dos tipos de dados de SQL

Agora que vimos os tipos de dados, vamos passar rapidamente pelas possíveis **restrições de integridade** que compõem o padrão SQL e vão fazer parte dos comandos de criação de tabelas.

RESTRIÇÕES DE INTEGRIDADE

Restrições de integridade SQL, que são geralmente referidas como restrições, podem ser divididos em três categorias:

- **Restrições em tabelas** – Definida dentro de uma definição de tabela. A restrição pode ser definida como parte da definição de coluna ou como um elemento ao final da definição. Restrições definidas no nível de tabela podem ser aplicadas a uma ou mais colunas.
- **Assertions (Afirmações)** - Um tipo de restrição que é definida dentro de uma definição afirmação (separado da definição da tabela). Uma afirmação pode ser relacionada com **uma ou mais tabelas**.
- **Restrições de domínio** - Um tipo de restrição que é definida dentro de uma definição de domínio (separado da definição da tabela). A restrição de domínio **está associada com qualquer coluna** que é definida baseada no domínio específico. Aqui você pode criar um domínio para atribuí-lo a uma coluna de uma tabela.

Para tentar clarear um pouco mais, observe a figura abaixo que apresenta o comando necessário para "ativar" cada uma das restrições e a distribuição, bem como o uso destes comandos de acordo com a classificação acima.

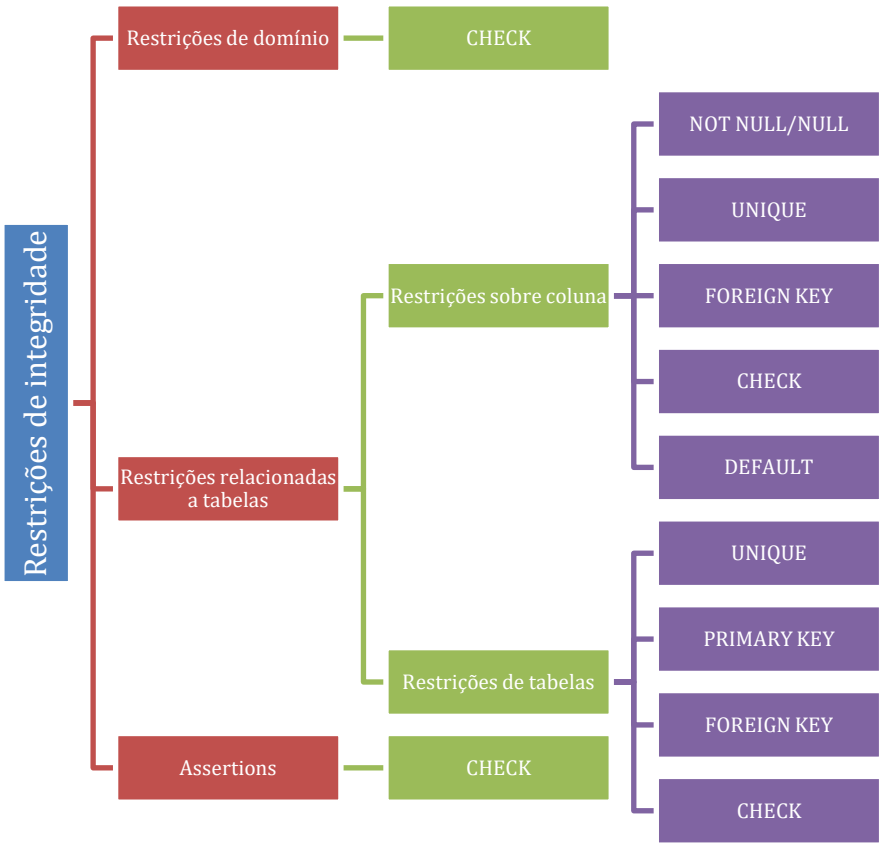


Figura 1 – Tipos de restrições de integridade

Falaremos um pouco sobre cada uma delas. A primeira seria a restrição de chave primária que pode ser simples, neste caso, basta colocar as palavras PRIMARY KEY ao lado da definição da coluna na construção do CREATE TABLE, ou composta adicionando ao final das definições das colunas PRIMARY KEY (col1, col2, ..., coln). Vejam que, entre parênteses, aparece o nome das colunas que fazem parte da chave, separadas por vírgulas.

A próxima restrição seria a restrição de integridade referencial. Esta restrição trata do relacionamento entre tabelas. Quando temos um atributo da tabela A sendo referenciado como atributo de uma tabela B, temos que garantir que os valores presentes em B sejam válidos. Esses valores têm, portanto, que estar presentes ou armazenados dentro da coluna referenciada ou serem nulos.

Ao criarmos uma integridade referencial, podemos incluir nela uma ação referencial engatilhada ou REFERENCIAL TRIGGERED ACTION. Ela basicamente vai optar por executar uma ação (CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION) quando for executada uma operação de DELETE ou UPDATE em uma tabela. Veja os exemplos abaixo.


```
CONSTRAINT EMPSUPERFK
      FOREIGN KEY (SUPERSSN) REFERENCES EMPREGADO (SSN)
      ON DELETE SET NULL
      ON UPDATE CASCADE,

CONSTRAINT EMPDEPTFK
       FOREIGN KEY (DNO) REFERENCES DEPARTAMENTO (DNUMERO)
       ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

É possível, ainda, termos as seguintes restrições:

1. NOT NULL que especifica que uma coluna não pode receber valores nulos;

2. UNIQUE que define que uma coluna não pode ter valores duplicados;

3. DEFAULT que atribui um valor padrão para uma determinada coluna a ser utilizado, caso nenhum valor seja passado como parâmetro no momento da inserção;

4. CHECK utilizado para restringir os valores de domínio, verificando se eles estão contidos no conjunto de valores especificados.

O CHECK é a restrição mais flexível de todas. Sua sintaxe básica é relativamente simples. Para criar uma restrição do tipo CHECK em uma coluna, você deve utilizar a seguinte sintaxe:

<nome_da_coluna> { <tipo de dados> | <domain> } CHECK (<search condition>).

No caso da criação da restrição em uma tabela use:

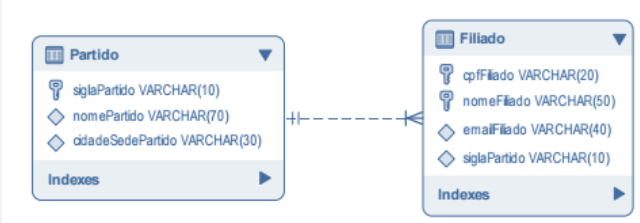
[CONSTRAINT <nome_da_restrição>] CHECK (<search condition>).

Agora que já aprendemos as restrições de integridade, já temos a capacidade de consolidar nosso conhecimento em DDL fazendo a questão abaixo.

HORA DE PRATICAR!

Ano: 2016 Órgão: AL-MS Prova: Técnico de Informática

Para responder à questão, considere o modelo mostrado na imagem abaixo, oriundo de uma situação hipotética:



Após a tabela Partido ser criada, para criar a tabela Filiado foi utilizada a instrução abaixo:

```
CREATE TABLE IF NOT EXISTS Filiado (
 cpfFiliado VARCHAR(20) NOT NULL,
 nomeFiliado VARCHAR(50),
 emailFiliado VARCHAR(40),
 siglaPartido VARCHAR(10) NOT NULL,
 PRIMARY KEY (cpfFiliado),
 FOREIGN KEY (siglaPartido)
I Partido (siglaPartido)
);
```

A lacuna I deverá ser corretamente preenchida por

a) CASCADE CONSTRAINT
b) REFERENCES
c) REFERENCE CONSTRAINT
d) EXTENDS
e) IMPLEMENTS

Comentário: Vejam que a questão está nos cobrando o conhecimento da sintaxe para a construção de uma restrição de integridade referencial. Neste caso, a lacuna deve ser completada com a palavra-chave REFERENCES. Veja que ela está fazendo um ponteiro para a coluna siglaPartido da tabela Partido. Sendo assim, podemos marcar a resposta na alternativa B.

Gabarito: B.

Agora vamos apresentar um esquema para ilustrar os diversos tipos de constraint, com suas respectivas sintaxes.

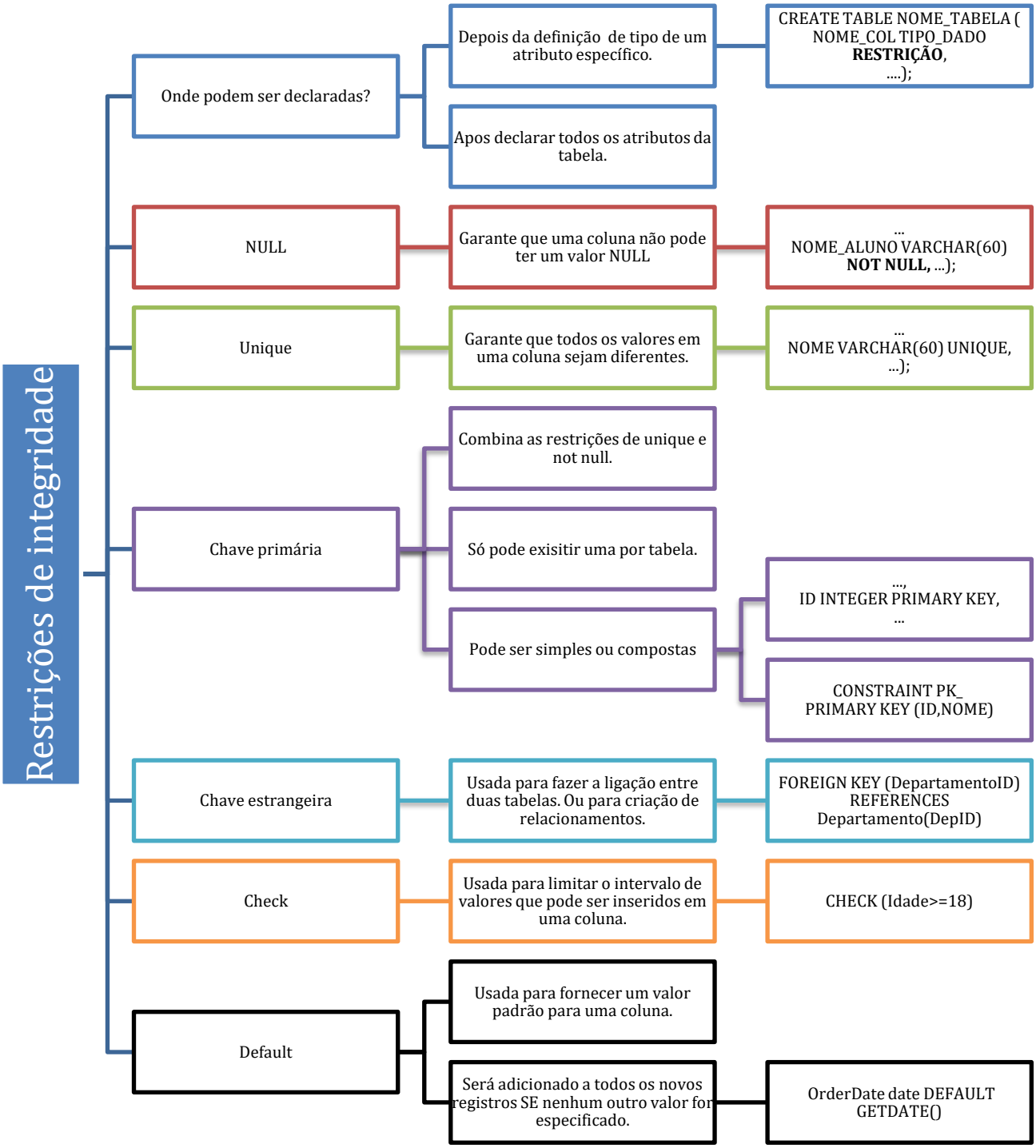


Figura 2 – Exemplos de restrições de integridade em SQL

DATA MANIPULATION LANGUAGE (DML)

São quatro os principais comandos DML: SELECT, INSERT, UPDATE e DELETE. Todos os SGBDs relacionais implementam esses comandos. Vamos conhecer a sintaxe de cada um deles. É importante perceber que a maioria das peculiaridades estão descritas para o comando SELECT. Antes, porém, vejamos uma descrição resumida de cada um dos comandos DML.

COMANDO	DESCRIÇÃO
SELECT	Comando utilizado para realizar consultas a dados de uma ou mais tabelas do banco de dados.
INSERT	Comando utilizado para inserir um registro em uma tabela do banco de dados.
UPDATE	Comando utilizado para mudar valores de dados de registros de uma tabela do banco de dados.
DELETE	Comando utilizado para remover linhas existentes de uma tabela do banco de dados.

O COMANDO SELECT

A instrução SELECT permite formar consultas complexas que podem retornar exatamente o tipo de dados que você deseja recuperar. Como você pode observar abaixo, as cláusulas exigidas na construção de um comando SELECT são a palavra reservada SELECT e a cláusula FROM. Todas as outras cláusulas são opcionais! É importante relembramos que este comando possui uma correspondência com a álgebra relacional. Assim, a projeção corresponde às colunas definidas no SELECT, e a seleção da álgebra relacional corresponde às condições de busca inseridas na cláusula WHERE. Podemos, ainda, observar um produto cartesiano entre as tabelas na cláusula FROM.

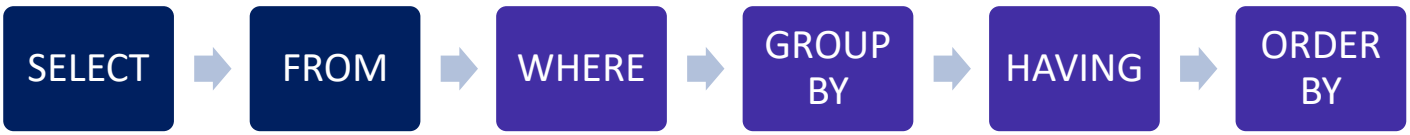

```
SELECT [ DISTINCT | ALL ] { * | <select list> }
FROM <table reference> [ { , <table reference> } . . . ]
[ WHERE <search condition> ]
[ GROUP BY <grouping specification> ]
[ HAVING <search condition> ]
[ ORDER BY <order condition> ]
```

Sobre o comando acima, a primeira informação importante é que o * (asterisco) é utilizado quando queremos extrair na nossa consulta todas as colunas da tabela, ou das relações descritas na cláusula FROM. Quando a nossa intenção for recuperar também todas as linhas, devemos omitir a cláusula WHERE. Nela são informadas as restrições que queremos fazer sobre a nossa busca. Podemos, por exemplo, restringir o domínio de um valor inteiro selecionando apenas as tuplas que satisfazem essa restrição.

Quando o SGBD recebe uma consulta com todas as palavras chaves acima, ele segue uma ordem lógica para avaliação do comando. Primeiramente ele analisa as tabelas ou relações envolvidas na consulta que estão presentes no FROM. Em seguida, o SGBD vai verificar as restrições de busca ou condições contidas na cláusula WHERE. Dando prosseguimento, caso exista uma especificação de agrupamento descrita no GROUP BY, ela é verificada.

Essa condição de agrupamento geralmente vem acompanhada de uma função de agregação sobre uma determinada coluna da tabela. Podemos pensar, por exemplo, na operação de soma (SUM) dos valores de um atributo numérico da tabela. Após o agrupamento é possível fazer uma restrição sobre os valores agrupados. Pela descrição da norma SQL/ANSI, você pode comparar o resultado dos agrupamentos e selecionar apenas aqueles que satisfaçam a uma condição de busca.

Após essa etapa, o SGBD vai avaliar quais colunas são descritas na cláusula SELECT do comando. Por fim, é possível ordenar a relação pelos atributos ou colunas listadas no ORDER BY. Resumindo temos:



CLÁSULAS	DESCRIÇÃO
SELECT	Comando utilizado para selecionar dados de uma tabela.
FROM	Comando utilizado para indicar de onde os dados devem ser selecionados.
WHERE	Comando utilizado para filtrar os dados.
GROUP BY	Comando utilizado para agregar um conjunto de dados.
HAVING	Comando utilizado para filtrar dados agregados.
ORDER BY	Comando utilizado para ordenar os dados recuperados.

No Oracle, por exemplo, é possível definir uma variável prefixada utilizando o & para avisar ao banco de dados que o valor será informado no momento da execução do comando. Veja, nas figuras abaixo, o uso do & para uma variável numérica, e para o caso do valor recebido ser uma data ou uma *string* de caracteres onde devem ser usadas as aspas simples.

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
```

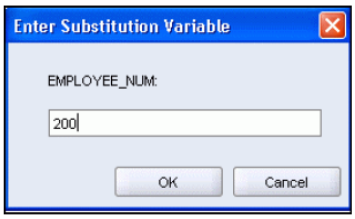
```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```

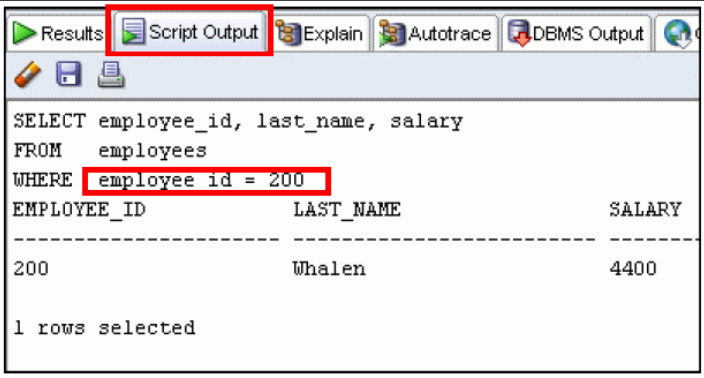
Outro ponto interessante no uso de variáveis é quando você quer reusar o valor da variável sem necessitar de digitar novamente ou repassar um novo valor. Neste caso, você deve usar dois &´s comerciais. Veja a figura abaixo:

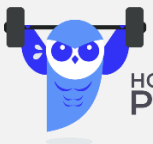
```
SELECT   employee_id, last_name, job_id, &&column_name
FROM     employees
ORDER BY &column_name ;
```

Use o comando VERIFY para alternar a exibição da variável de substituição, tanto antes como depois SQL Developer substitui variáveis de substituição com valores:

```
SET VERIFY ON
SELECT employee_id, last_name, salary
FROM   employees
WHERE  employee_id = &employee_num;
```







(Ano: 2017 Órgão: MPE-BA Prova: Analista Técnico – Tecnologia)
Atenção: Tabelas R e S referentes a um banco de dados relacional.

R	
a	b
1	2
2	3
4	5

S	
c	d
3	2
4	2
6	1

Considerando as tabelas R e S apresentadas anteriormente, o resultado

a	b
1	2
2	3

seria obtido pela execução do comando SQL:
a) select * from R
where not exists (select * FROM S where a=c)
b) select * from R
where not exists (select * FROM S where a=d)
c) select * from S
where not exists (select * FROM R where a=c)
d) select * from S
where not exists (select * FROM R where a=d)
e) select * from R
where exists (select * FROM S where b=d)

Comentário: Essa questão é interessante. Não é uma questão muito fácil, mas é importante para que você alinhe suas perspectivas em relação às bancas. Vamos resolvê-la passo a passo.
Primeira coisa que você tem que ter em mente é onde estão os registros que aparecem no resultado? Veja que os pares (1,2) e (2,3), além dos atributos a e b estão presentes na relação R. Logo, a primeira percepção que podemos ter é que o comando deve iniciar com: **select * from R**. Com isso, eliminamos as alternativas c e d.
Agora, vamos comparar as colunas de cada uma das alternativas que sobraram. Vejamos a alternativa A. A pergunta é: existe algum atributo onde a = c? Veja que apenas o valor 4 é igual nas duas colunas. Sendo assim, ele **não** irá aparecer na resposta por conta da cláusula NOT EXISTS. O resultado será, portanto, as duas primeiras linhas da tabela R. Desta forma, essa alternativa **é nossa resposta**.
Vamos, então, para a alternativa B. Neste caso, queremos saber se a=d. Neste caso, a consulta interna vai retornar um resultado para as duas primeiras linhas de R. E como o resultado desta consulta interna é verificado pela cláusula NOT EXISTS, o resultado será apenas as últimas linhas de R.
Por fim, vamos à alternativa E. Neste caso, comparamos b e d. Veja que agora usamos a cláusula EXISTS. Desta forma, o único resultado compatível, onde o valor de b da linha de R é igual a um dos valores de d em S, é a primeira linha de R. Sendo este resultado diferente do solicitado na questão. Portanto, essa não é a nossa resposta.

Gabarito: A.

ALIAS, DISTINCT E OPERADORES

A Linguagem SQL possui um mecanismo para renomear tanto as relações quanto os atributos por meio da cláusula **as**, da seguinte forma: nome_antigo AS novo_nome. Esse mecanismo é conhecido como **alias**. Quando utilizado no contexto de uma tabela na cláusula FROM, os alias são conhecidos como variáveis de tuplas.

Existem diversos modificadores de consulta que nos ajudam a adequar o retorno da nossa consulta. Para forçar a eliminação de tuplas duplicadas, devemos inserir a declaração **DISTINCT** depois do SELECT. A cláusula SELECT pode conter expressões aritméticas envolvendo os operadores **+**, **-** , ***** , e **/**, em operações que utilizam constantes ou atributos de tabelas.

Não é necessário que os atributos usados na cláusula WHERE estejam listados no SELECT. A cláusula corresponde ao predicado de seleção da álgebra relacional como já falamos. Ela consiste de um predicado envolvendo atributos das relações que aparecem na cláusula FROM. Podemos utilizar operadores aritméticos (+ , - , * , / , % (módulo)), operadores de comparação (<, <=, >, >=, = e <>) e ainda conectivos ou operadores lógicos (AND, OR e NOT).

Outra possibilidade de comparação existente é quando estamos comparando STRINGs. Neste caso, podemos utilizar os operadores LIKE, NOT LIKE, % (que combina qualquer substring, independentemente do tamanho) e _ (combina caractere a caractere). Basicamente o que fazemos com esses operadores é verificar se um valor de um atributo em uma determinada tupla é semelhante ao de uma constante passada como parâmetro. Vejamos um exemplo, se você quiser encontrar o nome de todos os clientes cuja rua contenha a substring 'Professor', você pode executar o comando abaixo.


```
select nome_cliente
from cliente
where rua_cliente like '% Professor %';
```

(Ministério da Economia – Especialista em Ciência de Dados - 2020) Julgue os itens a seguir, a respeito de conceitos de SQL.

O operador LIKE é usado para pesquisar um padrão especificado em uma coluna da tabela.

_____

Comentários: O operador LIKE é usado em uma cláusula WHERE para pesquisar um padrão especificado em uma coluna textual. Existem dois curingas geralmente usados em conjunto com o operador LIKE:

% - o sinal de porcentagem representa zero, um ou vários caracteres
_ - O sublinhado representa um único caractere

Gabarito: CERTO.

Quando construímos um predicado com vários operadores, precisamos entender que existe uma precedência na avaliação dos operadores. Você pode usar parênteses para sobrescrever as regras de precedência. A ordem de avaliação das regras de precedências pode ser vista na lista abaixo:

Tabela 1 - Ordem de precedência dos operadores em SQL

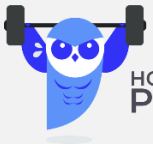
OPERADOR	SIGNIFICADO
1	Operadores aritméticos
2	Operadores de concatenação
3	Operadores condicionais
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	Operador lógico NOT
8	Operador lógico AND
9	Operador lógico OR

Depois de definir a cláusula WHERE podemos ordenar o resultado da consulta por meio do comando ORDER BY. É possível ordenar por um determinado atributo em ordem crescente e outro em ordem decrescente. Para isso, basta seguir a seguinte sintaxe: ORDER BY <nome(s)_coluna(s)> DESC ou ASC (**default**) e separar por vírgula os nomes das colunas com suas respectivas formas de ordenação. Lembrando que, se não for definida, a ordenação padrão é feita de forma ascendente.

Para executarmos a ordenação, pode-se optar por usar valores numéricos que referenciam a enésima coluna que aparece na cláusula SELECT. Veja o exemplo na figura abaixo, cujo valor **3** referencia a coluna *department_id*:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY 3;
```

Vejamos um exemplo de questão de provas anteriores.



(Ano: 2016 Órgão: TRE-SP Cargo: Analista Judiciário de TI – Discursiva. Q. 2)

Considere que, em uma situação hipotética, uma equipe de Analistas de Sistemas do TRE-SP irá participar do desenvolvimento de um novo sistema com base na metodologia ágil Scrum. Para a escolha do Time Scrum, algumas tarefas foram solicitadas para definição do Product Owner, do responsável pelo Backlog do Produto e dos membros da Equipe de Desenvolvimento. Como base do teste, foi criada uma tabela no banco de dados denominada ELEICOES, que tem os campos indicados na 1a linha e os conteúdos possíveis na 2a linha, conforme abaixo.

NomeCandidato	Cargo	VotosValidos	Sexo	CodigoPartido
Até 100 caracteres	Prefeito Vereador	Numérico inteiro positivo	Feminino Masculino	1 a 10

Neste contexto, solicita-se que sejam apresentadas soluções para o que se pede abaixo.
a. Escrever um comando SQL para mostrar, em uma linha, a quantidade de candidatos do sexo masculino como "CandidatosHomens" e a quantidade de candidatas do sexo feminino como "CandidatasMulheres".
b. Escrever um comando SQL para apresentar todos os dados dos candidatos a Prefeito por ordem crescente de VotosValidos.

_____

Comentário: Vamos tentar apresentar uma solução para cada uma das alternativas:
a. SELECT * FROM
(SELECT COUNT(SEXO) AS “CandidatosHomens”
FROM ELEICOES WHERE SEXO=”Masculino”) as M,
(SELECT COUNT(SEXO) AS “CandidatosMulheres”
FROM ELEICOES WHERE SEXO=”Feminino”) as F;
b. SELECT * FROM ELEICOES WHERE CARGO = “Prefeito”
ORDER BY VotosValidos ASC;

Gabarito: Discursiva!

CONSULTAS ANINHADAS (IN, EXISTS, ALL, SOME, ANY)

Algumas consultas precisam que os valores existentes no banco de dados sejam buscados e depois usados em uma condição de comparação. SQL provê um mecanismo para aninhamento de subconsultas. Uma subconsulta é uma expressão do tipo SELECT-FROM-WHERE que é aninhada dentro de outra consulta. As aplicações mais comuns para as subconsultas são testes para membros de conjuntos, comparação de conjuntos e cardinalidade de conjuntos.

Sempre que uma condição na cláusula WHERE de uma consulta aninhada referencia algum atributo de uma relação declarada na consulta externa, as duas consultas são consideradas correlacionadas. Vejamos um exemplo. Vamos encontrar todos os clientes que possuem conta e empréstimo em um banco.


```
SELECT DISTINCT nome_cliente
FROM devedor
WHERE nome_cliente IN (select nome_cliente
       from depositante)
```

Observem que utilizamos acima o construtor **IN**, ele testa se um valor existe no outro subconjunto. Outros construtores importantes são o **UNIQUE** que testa se a subconsulta tem alguma tupla repetida no seu resultado; o **EXISTS** que retorna o valor TRUE se a subconsulta usada como argumento é "não vazia".

É possível, ainda, usar a palavra-chave **NOT** em conjunto com os operadores EXISTS e IN. Nestes casos o operador **NOT IN** verifica se o valor não existe em outro subconjunto. Semelhantemente podemos usar o NOT EXISTS que retorna um valor booleano para aceitação ou não da tupla em questão, caso o argumento da subconsulta seja vazio.

Precisamos ainda entender as palavras-chave **ALL**, **ANY** e **SOME**. A norma padrão não trata da palavra SOME, mas ela funciona da mesma forma que a ANY. A utilização dessas palavras serve para comparar o valor da subconsulta externa com todos os valores da subconsulta interna. É possível, por exemplo, saber se um valor é maior que todos os valores da subconsulta. Vejam abaixo.

```
SELECT Name
FROM Product
WHERE ListPrice >= ALL
    (SELECT ListPrice
     FROM Product
     GROUP BY ProductSubcategoryID);
```

Uma classificação para as subconsultas é se eles retornam uma ou múltiplas linhas. Algumas dicas devem ser levadas em consideração quando estamos elaborando subconsultas: (1) coloque subconsultas entre parênteses; (2) subconsultas devem ocupar o lado direito das comparações condicionais para facilitar a legibilidade; e (3) use operadores de única linha com subconsultas de uma **única linha** (=, >, >=, <, <=, <>) e operadores de **múltiplas linhas** (ALL e ANY) com subconsultas de várias linhas.

FUNÇÕES AGREGADAS (GROUP BY E HAVING)

Vamos conhecer agora as funções agregadas que operam sobre conjuntos de valores de uma coluna de uma relação e retornam um valor para cada conjunto, são elas: COUNT (), SUM (), MAX (), MIN (), AVG (). Quando utilizamos funções agregadas, devemos utilizar o GROUP BY para os atributos na cláusula SELECT que não aparecem como parâmetros nas funções agregadas e a opção HAVING para predicados que são aplicados após a formação dos grupos.

Outra opção de subconsultas que retornam uma **única linha** é quando usamos funções de agregação como MIN e MAX. Vejam o exemplo abaixo para entendermos melhor a ideia:

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =          2500
                (SELECT MIN(salary)
                 FROM   employees);
```

Vejamos uma questão sobre o assunto:

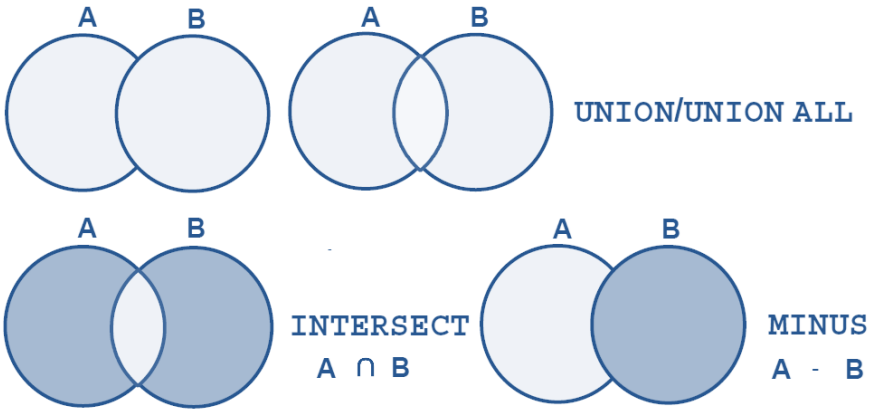
Ano: 2017Órgão: TRT-11 Cargo: Técnico Judiciário de TI – Q. 44
Considere que no TRT exista, em um banco de dados, a tabela TRAB que possui como campos: nome, sexo, salario de vários trabalhadores. Um Técnico foi solicitado a escrever um comando SQL para obter a média salarial dos trabalhadores do sexo FEMININO. O comando correto é:
(A) SELECT sexo="FEMININO" FROM TRAB WHERE AVG(salario);
(B) SELECT sexo, AVG(salario) as MediaSalarial FROM TRAB GROUP BY sexo;
(C) SELECT AVG(salario) FROM TRAB WHERE sexo='FEMININO';
(D) SELECT sexo, AVG(salario) FROM TRAB GROUP BY sexo="FEMININO";
(E) SELECT * FROM TRAB WHERE sexo='FEMININO' as AVG(salario);

Comentário: Vejam que a questão pede apenas uma informação: a média salarial dos trabalhadores do sexo feminino. Desta forma, basta usarmos a função agregado que retorna a média dos valores não nulos de uma coluna (AVG) e restringirmos nossa consulta na cláusula WHERE pelos empregados do sexo feminino (sexo='FEMININO'). Sendo assim, podemos encontrar nossa resposta na alternativa C.

Gabarito: C.

OPERAÇÕES DE CONJUNTOS

As operações de conjuntos **UNION**, **INTERSECT** e **EXCEPT** operam em relações e correspondem às operações ∪, ∩ e - (diferença) da álgebra relacional. Estas operações eliminam as duplicatas. Se desejarmos obter as repetições, devemos explicitar por meio da forma **UNION ALL**, **INTERSECT ALL** e **EXCEPT ALL**.



É interessante perceber que a união é feita sobre o resultado de duas relações, ou seja, é possível duas consultas sobre tabelas distintas passarem por uma operação de união. É importante entender, porém, que os atributos das duas relações que participam da operação devem operar sobre os mesmos domínios. Vejamos um exemplo. Suponha que você queira encontrar todos os clientes que possuam um empréstimo, uma conta ou ambos.

```
(SELECT nome_cliente FROM depositante ) UNION (select
nome_cliente FROM devedor)
```

JUNÇÃO

Por fim, vamos falar da composição de relações utilizando junção ou JOIN. As operações de junção tomam duas relações e retornam como resultado outra relação. São normalmente usadas na cláusula ***from*** e devem ser feitas seguindo uma condição e um tipo de junção. A **condição de junção** define quais tuplas das duas relações apresentam correspondência, e quais atributos serão apresentados no resultado de uma junção. O **tipo de junção** define como as tuplas em cada relação que não possuam nenhuma correspondência (baseado na condição de junção) com as tuplas da outra relação devem ser tratadas.

Observem a figura com as condições e os tipos de junção abaixo:

Tipos de junção	Condição de Junção
INNER JOIN	**NATURAL**

LEFT OUTER JOIN	**ON** <PREDICADO>
RIGTH OUTER JOIN	**USING** (A1, A2, ..., NA)
FULL OUTER JOIN	

Figura 1 - Tipos e condições de junção

Quando utilizamos a condição de junção **natural,** o SQL basicamente vai utilizar os atributos das duas relações que possuem os mesmos nomes. Caso não deseje utilizar todos os atributos com nomes iguais, você pode restringi-los utilizando a condição **using,** passando como argumentos apenas os atributos que você deseja.

Vejam um exemplo abaixo do uso do NATURAL JOIN E do USING:

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

Apenas uma consideração importante: você não deve usar qualificadores para as colunas que serão utilizadas na cláusula USING. Isso pode ocasionar um erro na execução da requisição (no caso do Oracle: ORA-25154).

Após esse breve resumo da operação de SELECT, passaremos agora para analisar os demais comandos DML.

Ano: 2016 Banca: IF-PE Órgão: IF-PE Prova: Técnico em Tecnologia da Informação - Desenvolvimento

Leia as afirmativas abaixo e responda à questão proposta.

I. Retorna linhas quando há, pelo menos, uma correspondência entre duas tabelas.

II. Operador usado para combinar o resultado do conjunto de duas ou mais instruções SELECT.

III. Operador usado em uma cláusula WHERE para pesquisar um padrão específico em uma coluna.

I, II e III correspondem, em SQL, respectivamente, aos comandos:

a) INNER JOIN, UNION e LIKE.

b) INNER JOIN, JOIN e DISTINCT.
c) LEFT JOIN, UNIQUE e LIKE.
d) SELECT, JOIN e BETWEEN.
e) SELECT, UNIQUE e BETWEEN.

_____

Comentário: Vamos analisar cada uma das afirmações.
A primeira trata de uma forma de junção que leva para o resultado apenas as tuplas que têm correspondência direta nas tabelas envolvidas na junção. Esse comando é conhecido como INNER JOIN.
Já a segunda afirmação pede um comando que faz a combinação ou união do resultado de duas consultas. Lembrem-se de que, para isso acontecer, os atributos ou as colunas da tabela devem possuir o mesmo domínio ou tipo de dado. Essa operação é conhecida como UNION.
Por fim, a última afirmação pergunta sobre uma palavra-chave que permite comparação de padrões específicos na cláusula WHERE. Sabemos que, para comparar *strings,* usamos o comando LIKE e os caracteres coringas "%" e "_".
Sendo assim, podemos marcar nosso gabarito na alternativa A.

Gabarito: A.

INSERT, UPDATE E DELETE

O comando INSERT INTO é utilizado para inserir novos registros em uma tabela. Sua sintaxe permite que você defina os valores para colunas de duas formas. Na primeira você não especifica os nomes das colunas nas quais os valores serão inseridos. Neste caso, o SGBD vai relacionar os valores na mesma ordem em que foram definidos no comando CREATE. A segunda forma seria definir explicitamente os nomes das colunas e os valores. Vejam as duas opções a seguir:

SINTAXE DO COMANDO I

```
INSERT INTO NOME_DA_TABELA
VALUES (VALOR_1, VALOR_2, VALOR_3, ...)
```

SINTAXE DO COMANDO II

```
INSERT INTO NOME_DA_TABELA (NOME_COLUNA1, NOME_COLUNA2, NOME_COLUNA3, ...)
VALUES (VALOR_1, VALOR_2, VALOR_3, ...)
```

EXEMPLOS DO COMANDO I

```
INSERT INTO ALUNO_ESTRATEGIA
VALUES ('ALICE', 11111111111, 'ALICE@ALICE.COM', 01-01-2001, 'BRASÍLIA', 200.00);

INSERT INTO ALUNO_ESTRATEGIA
VALUES ('BRUNO', 22222222222, 'BRUNO@BRUNO.COM', 02-02-2002, 'SÃO PAULO', 100.00);

INSERT INTO ALUNO_ESTRATEGIA
VALUES ('CAIO' , 33333333333, 'CAIO@CAIO.COM'  , 03-03-2003, 'GOIÂNIA, 150.00);

INSERT INTO ALUNO_ESTRATEGIA
VALUES ('DIEGO', 44444444444, 'DIEGO@DIEGO.COM', 04-04-2004, 'SALVADOR, 250.00);

INSERT INTO ALUNO_ESTRATEGIA
```

```
VALUES ('ELIS',  55555555555, 'ELIS@ELIS.COM'  , 05-05-2005, 'BRASÍLIA', 50.00);
```

EXEMPLOS DO COMANDO II

```
INSERT INTO ALUNO_ESTRATEGIA (NOME, CPF, EMAIL, DATA_NASCIMENTO, CIDADE, VALOR_PAGO)
VALUES ('FABIO', 66666666666, 'FABIO@FABIO.COM', 06-06-2006, 'SALVADOR', 125.00);

INSERT INTO ALUNO_ESTRATEGIA (NOME, CPF, EMAIL, DATA_NASCIMENTO, CIDADE, VALOR_PAGO)
VALUES ('GABI',  77777777777, 'GABI@GABI.COM'  , 07-07-2007, 'BRASÍLIA', 225.00);

INSERT INTO ALUNO_ESTRATEGIA (NOME, CPF, EMAIL, DATA_NASCIMENTO, CIDADE, VALOR_PAGO)
VALUES ('HUGO',  88888888888, 'HUGO@HUGO.COM'  , 08-08-2008, 'BRASÍLIA', 50.00);

INSERT INTO ALUNO_ESTRATEGIA (NOME, CPF, EMAIL, DATA_NASCIMENTO, CIDADE, VALOR_PAGO)
VALUES ('IGOR',  99999999999, 'IGOR@IGOR.COM'  , 09-09-2009, 'RECIFE'  , 75.00);

INSERT INTO ALUNO_ESTRATEGIA (NOME, CPF, EMAIL, DATA_NASCIMENTO, CIDADE, VALOR_PAGO)
VALUES ('JOÃO',  00000000000, 'JOAO@JOAO.COM'  , 10-10-2010, 'NATAL'  , 175.00);
```

RESULTADO DOS COMANDOS

ALUNO_ESTRATEGIA					
NOME	CPF	EMAIL	DATA_NASCIMENTO	CIDADE	VALOR_PAGO
ALICE	11111111111	ALICE@ALICE.COM	01-01-2001	BRASÍLIA	200.00
BRUNO	22222222222	BRUNO@BRUNO.COM	02-02-2002	SÃO PAULO	100.00
CAIO	33333333333	CAIO@CAIO.COM	03-03-2003	GOIÂNIA	150.00
DIEGO	44444444444	DIEGO@DIEGO.COM	04-04-2004	SALVADOR	250.00
ELIS	55555555555	ELIS@ELIS.COM	05-05-2005	BRASÍLIA	50.00
FABIO	66666666666	FABIO@FABIO.COM	06-06-2006	SALVADOR	125.00
GABI	77777777777	GABI@GABI.COM	07-07-2007	BRASÍLIA	225.00
HUGO	88888888888	HUGO@HUGO.COM	08-08-2008	BRASÍLIA	50.00
IGOR	99999999999	IGOR@IGOR.COM	09-09-2009	RECIFE	75.00
JOÃO	00000000000	JOAO@JOAO.COM	10-10-2010	NATAL	175.00

No ORACLE, se você quiser passar como parâmetro o valor corrente da data e hora é possível fazer uso da função SYSDATE. Outra possibilidade é usar como valores do comando INSERT uma subconsulta. Neste caso, você não precisa usar a cláusula VALUES e o número de coluna da subconsulta deve ser igual ao número de colunas do INSERT. Vejamos um exemplo para ajudar na fixação do assunto:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';

4 rows inserted
```

Outro comando que compõe a linguagem é o UPDATE. Utilizado para atualizar registros em uma tabela, ele basicamente define, na sua cláusula WHERE, quais linhas ou registros devem ser atualizados. Caso nenhuma verificação seja feita, todas as linhas serão atualizadas. As mudanças a serem executas são definidas na cláusula SET. Vejam a sintaxe do comando abaixo:

SINTAXE DO COMANDO

```
UPDATE NOME_DA_TABELA
SET NOME_DA_COLUNA_1 = VALOR_1, NOME_COLUNA2 = VALOR_2 ...
WHERE LISTA_DE_CONDICOES
```

EXEMPLO DO COMANDO

```
UPDATE ALUNO_ESTRATEGIA
SET NOME = 'DIOGO', EMAIL = 'DIOGO@DIOGO.COM'
WHERE CPF = 44444444444

UPDATE ALUNO_ESTRATEGIA
SET NOME = 'ELIAS', EMAIL = 'ELIAS@ELIAS.COM'
WHERE CPF = 55555555555
```

RESULTADO DOS COMANDOS

ALUNO_ESTRATEGIA					
NOME	CPF	EMAIL	DATA_NASCIMENTO	CIDADE	VALOR_PAGO
ALICE	11111111111	ALICE@ALICE.COM	01-01-2001	BRASÍLIA	200.00
BRUNO	22222222222	BRUNO@BRUNO.COM	02-02-2002	SÃO PAULO	100.00
CAIO	33333333333	CAIO@CAIO.COM	03-03-2003	GOIÂNIA	150.00
DIOGO	44444444444	DIOGO@DIOGO.COM	04-04-2004	SALVADOR	250.00
ELIAS	55555555555	ELIAS@ELIAS.COM	05-05-2005	BRASÍLIA	50.00
FABIO	66666666666	FABIO@FABIO.COM	06-06-2006	SALVADOR	125.00
GABI	77777777777	GABI@GABI.COM	07-07-2007	BRASÍLIA	225.00
HUGO	88888888888	HUGO@HUGO.COM	08-08-2008	BRASÍLIA	50.00
IGOR	99999999999	IGOR@IGOR.COM	09-09-2009	RECIFE	75.00
JOÃO	00000000000	JOAO@JOAO.COM	10-10-2010	NATAL	175.00

Valores para um conjunto de linhas específicas são atualizados ou modificados se você definir os critérios na cláusula WHERE. Se você quiser atribuir à coluna o valor NULL, basta usar SET nome_da_coluna = NULL. Lembre-se de que, caso nenhum critério for definido na cláusula WHERE, todas as linhas são atualizadas. Veja alguns exemplos do comando UPDATE:

```
UPDATE employees
SET    department_id = 50
WHERE  employee_id = 113;
1 rows updated
```

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated
```

HORA DE PRATICAR!

(Ministério da Economia – Desenvolvimento de Sistemas - 2020)

Tendo como referência o diagrama de entidade relacionamento precedente, julgue próximo item, a respeito de linguagem de definição de dados e SQL.

A seguinte expressão SQL permite alterar a nota de geografia do aluno de nome Beltrano para 9.5.

```
update matricula set nota=9.5
where aluno='Beltrano' and disciplina ='Geografia'.
```

Comentários: Na tabela de matrículas, não temos os nomes dos alunos e as descrições das disciplinas. Logo, para atualizar a nota de Beltrano precisamos conhecer seu Id, bem como o Id da disciplina Geografia. Outro ponto, se Beltrano cursou essa disciplina em mais de um ano letivo, temos que especificar a qual ano essa alteração se refere. Caso contrário, todas as notas de Geografia de Beltrano seriam alteradas para 9.5.

Assim, o comando acima não está correto, pois não consegue alterar a nota de Beltrano.

Gabarito Errado.

Vejamos algumas informações sobre o DELETE. A instrução DELETE é usada para deletar linhas de uma tabela. Veja que, se nenhuma condição for definida na cláusula WHERE do comando, todas as linhas serão removidas da tabela. A sintaxe do comando é apresentada a seguir:

SINTAXE DO COMANDO

```
DELETE FROM NOME_DA_TABELA WHERE LISTA_DE_CONDICOES
```

EXEMPLO DO COMANDO

```
DELETE FROM ALUNO_ESTRATEGIA WHERE VALOR_PAGO = 175.00 OR CIDADE = 'RECIFE';
```

RESULTADO DO COMANDO

ALUNO_ESTRATEGIA					
NOME	CPF	EMAIL	DATA_NASCIMENTO	CIDADE	VALOR_PAGO
ALICE	11111111111	ALICE@ALICE.COM	01-01-2001	BRASÍLIA	200.00
BRUNO	22222222222	BRUNO@BRUNO.COM	02-02-2002	SÃO PAULO	100.00
CAIO	33333333333	CAIO@CAIO.COM	03-03-2003	GOIÂNIA	150.00
DIOGO	44444444444	DIOGO@DIOGO.COM	04-04-2004	SALVADOR	250.00
ELIAS	55555555555	ELIAS@ELIAS.COM	05-05-2005	BRASÍLIA	50.00
FABIO	66666666666	FABIO@FABIO.COM	06-06-2006	SALVADOR	125.00
GABI	77777777777	GABI@GABI.COM	07-07-2007	BRASÍLIA	225.00
HUGO	88888888888	HUGO@HUGO.COM	08-08-2008	BRASÍLIA	50.00

Outro comando definido como **DDL,** mas que tem como funcionalidade remover, de forma rápida e fácil, todas as linhas de uma tabela, é o comando TRUNCATE. Ele remove todo conteúdo da tabela, zera os índices e as sequências associadas. Para tal, basta utilizar o comando DDL:

SINTAXE DO COMANDO

```
TRUNCATE TABLE NOME_DA_TABELA;
    [RESTART IDENTITY | CONTINUE IDENTITY] [CASCADE | RESTRICT]
```

EXEMPLO DO COMANDO

```
TRUNCATE TABLE ALUNO;
```

RESULTADO DO COMANDO

ALUNO					
NOME	CPF	SEXO	DATA_NASCIMENTO	CIDADE	VALOR_PAGO

Acima, foi possível observar o comando TRUNCATE com a sintaxe específica do POSTGRES. Nele é possível remover um conjunto de tabelas ao mesmo tempo. O primeiro parâmetro permite reiniciar,

ou não, as sequências associadas às colunas das tabelas cujos dados serão removidos, usando, respectivamente, o **restart identity** e **continue identity**. O outro parâmetro, **cascade** ou **restrict**, vai aplicar o comando truncate às tabelas que têm relacionamento ou chaves estrangeiras referenciando uma das tabelas removidas.

HORA DE PRATICAR!

(Ministério da Economia – Desenvolvimento de Sistemas - 2020)

Tendo como referência o diagrama de entidade relacionamento precedente, julgue próximo item, a respeito de linguagem de definição de dados e SQL.

A expressão SQL a seguir permite excluir as notas do aluno de nome Fulano.

truncate from matricula where aluno='Fulano'.

Comentários: O comando TRUNCATE serve para limpar uma tabela, removendo todos os seus registros de forma rápida e eficiente. Para deletar linhas específicas devemos utilizar o comando DELETE.

Gabarito Errado.

(Ano: 2017 Banca: FCC Órgão: TRF - 5ª REGIÃO Prova: Técnico Judiciário – Informática)

Após constatar que todos os dados em uma tabela estavam incorretos, foi solicitado ao Técnico em Informática para limpar os registros desta tabela mantendo sua estrutura, para que os dados corretos fossem posteriormente inseridos. Para realizar este trabalho o Técnico terá que utilizar a instrução SQL

a) DROP TABLE table_name.
b) REDO * FROM table_name.
c) DELETE TABLE table_name.
d) ERASE * FROM table_name.
e) TRUNCATE TABLE table_name.

Comentário: Essa questão apresenta o propósito do comando TRUNCATE. Ele vai "zerar" a tabela. O comando TRUNCATE TABLE remove todas as linhas de uma tabela sem registrar as exclusões de linhas individuais. O TRUNCATE TABLE funciona como uma instrução DELETE, porém, sem usar a cláusula WHERE. O fato de ele não guardar os logs de transação o faz ser classificado como uma instrução DDL por diversos autores. Observando as alternativas, podemos encontrar nossa resposta na **letra E**.

Gabarito: E.

DDL COMPLEMENTOS: VIEW

Outro objeto SQL que podemos criar dentro dos nossos bancos de dados são as VIEWs. A view é um comando SQL que é armazenado no banco de dados e possui um nome associada a ela. Eles têm algumas funções básicas. A primeira é facilitar a visualização dos dados dispersos em diversas tabelas, tornando-os mais naturais ou intuitivos ao entendimento humano.

Outra função importante para a *view* está relacionada à segurança dos dados. É possível restringir o acesso aos campos e às colunas de uma tabela por meio de uma *view*. Desta forma, o usuário teria visão apenas a parte dos dados ou informações. Esse grupo de informações deve ser compatível com as funções e necessidades de acesso do usuário.

Outra opção para o uso de *view* é sumarizar dados de diferentes tabelas gerando relatórios. Se pensarmos no INFORMATION_SCHEMA, que tratamos no início da aula, ele geralmente é composto por um conjunto de visões que trazem os dados referentes às tabelas dos esquemas do seu banco de dados.

Vejamos, abaixo, dois exemplos do uso de **VIEWs**. Lembrando que ela pode ser criada sobre uma ou múltiplas tabelas. Observe que o comando basicamente inclui a sintaxe CREATE VIEW nome AS antes de uma consulta ao banco de dados.

```
CREATE VIEW profs_estrategia AS
SELECT pf.primeironome, pf.ultimonome, pf.telefone, pf.email
FROM professores pf
NATURAL JOIN disciplina d
WHERE d.nome = 'Informática';
```

Para visualizarmos os dados de uma visão, basta escrevermos um comando SELECT sobre ela, vejam o exemplo sobre a view **profs_estrategia** criada acima.

```
SELECT * FROM profs_estrategeia;
```

A view é considerada uma tabela virtual porque ela só existe durante o período em que você está utilizando-a. Todas as operações que são feitas sobre a tabela podem ser realizadas em uma VIEW, mas a tabela é virtual, e, na teoria, não deve ser armazenada no banco de dados.

Como falamos, é possível usar os comandos DML (Select, Insert, Update e Delete) sobre uma visão. Essas modificações podem, ou não, ter seus efeitos sobre os dados armazenados nas tabelas subjacentes, ou seja, associadas à visão. Uma visão que permite a atualização da tabela subjacente tem uma associação direta entre as linhas da visão e as linhas da tabela.

O padrão SQL/ANSI define algumas regras que indicam se a visão pode ser atualizada. Primeiramente, a visão deve ser construída com base em apenas uma tabela. As cláusulas GROUP

BY, HAVING e SELECT DISTINCT **não** podem estar presentes. **Não** podem existir nenhuma função de agregação ou colunas calculadas (atributos derivados).

> Se você está usando valores agregados na visão (por exemplo, SUM, COUNT e AVG), você não tem a permissão para alterar os dados das tabelas subjacentes. Da mesma forma, se na sua view tiver as palavras-chave GROUP BY, DISTINCT, ou HAVING, também não é permitido fazer alterações nos dados.

Por implicação das regras acima, uma das colunas da view deve ser a chave primária da tabela subjacente. Enfim, nós precisamos ter certeza de que, a partir de uma linha da visão, conseguimos referenciar uma linha da tabela para que possamos atualizar o conteúdo dela.

É possível, ainda, usar a cláusula WITH CHECK OPTION ao final do comando de definição da VIEW. Nestes casos, qualquer INSERT ou UPDATE que seja executado sobre a visão irá verificar se os dados inseridos ou modificados estão de acordo com as restrições descritas na cláusula WHERE.

Por fim, é possível apagar a visão quando você não precisar mais dela por meio do comando DROP VIEW nome_da_visao.

Ano: 2016 Órgão: TCE-PA Prova: Auditor de Controle Externo - Área Informática - Analista de Suporte

No que concerne à linguagem SQL, julgue o item seguinte.

[1] Ao se criar uma view, não é necessário que os nomes dos atributos da view sejam os mesmos dos atributos da tabela.

[2] Para que um usuário possa executar o comando select em uma view, não é necessário que ele tenha esse privilégio diretamente na view, mas apenas na tabela a que a view faz referência.

Comentário: Vamos analisar cada uma das afirmações acima.

A primeira trata dos nomes associados à visão e aos seus atributos. Lembre-se de que você consegue "renomear" as colunas que fazem parte da visão utilizando a sintaxe do comando. Sendo assim, podemos concluir que a primeira afirmação está correta.

Já a segunda afirmação pode ser considerada um pouco mais complexa, em especial neste ponto da aula, pois ainda não tratamos dos aspectos relacionados às permissões em SQL. A verdade é que você pode adicionar as permissões de acesso às tabelas e visões. Normalmente, os privilégios podem ser aplicados a quaisquer objetos do SGBD como views, tabelas, procedures de forma independente. Sendo assim, ter o privilégio para visualizar uma tabela não implica que terá acesso à view derivada. Logo, a alternativa II está **errada**!

Gabarito: C E.

SQL EMBUTIDO

O padrão SQL define que a SQL será embutida em uma variedade de linguagens de programação, como Pascal, PL/I, Fortran, C e Cobol. A linguagem na qual são embutidas consultas SQL é chamada linguagem hospedeira, e as estruturas SQL permitidas na linguagem hospedeira são denominadas SQL embutida.

EXEC SQL é usado para identificar os pedidos em SQL embutida para o pré-processador EXEC SQL <comando SQL embutido> END EXEC. Suponha que tenhamos, na linguagem hospedeira, uma variável chamada total e que desejamos encontrar os nomes e cidades dos clientes que tenham mais de um total em dólares em qualquer conta. Podemos escrever essa consulta da seguinte maneira:


```
EXEC SQL
  declare c cursor for
  select nome_cliente,cidade_cliente
     from deposito,cliente
     where deposito.nome_cliente = cliente.nome_cliente
                  and depósito.saldo > :total
END-EXEC
```

O SQL embutido permite, ainda, o uso de alguns comandos. O comando **open** faz com que a consulta seja avaliada:

```
EXEC SQL open c END-EXEC
```

O comando **fetch** determina os valores de uma tupla que serão colocados em variáveis da linguagem de host.

```
EXEC SQL fetch c into :cn :an END-EXEC
```

Pode-se utilizar um laço para processar cada tupla do resultado. Uma variável na área de comunicação do SQL indica que não há mais tupla a ser processada.

O comando **close** faz com que o sistema de banco de dados remova a relação temporária mantida para o resultado da consulta.

```
EXEC SQL close c END-EXEC
```

TRANSAÇÕES EM SQL

Uma transação é um conjunto de instruções SQL que realizam um único trabalho. O exemplo clássico de transação é a transferência de dinheiro de uma conta para outras. Primeiro é necessário saber o saldo da conta X, depois retirar o dinheiro desta conta e creditar em outra conta Y. Essas três ações se caracterizam em um trabalho que deve ser executado em um único bloco.

Durante a transação, todos os passos devem ser executados. Isso nos leva ao conceito de **atomicidade** da transação. Caso um passo não possa ser executado, o banco de dados deve ser posto no seu estado inicial válido. Isso garante a **consistência** da base após a execução de uma transação.

Além da atomicidade e consistências, são consideradas propriedades de uma transação o **isolamento** e a **durabilidade**. O primeiro se refere ao fato de que uma transação não deve enxergar os dados e a execução de outras transações. O segundo afirma que, caso uma transação termine com sucesso seus dados, as suas modificações feitas sobre a base de dados tornam-se persistentes.

Uma *transaction* é um mecanismo que podemos utilizar para manter a integridade e a consistência dos dados. SQL nos ajuda a gerenciar transações. SQL padrão definiu transações logo no início da criação do modelo e vem aprimorando o conceito durante iterações subsequentes. De acordo com a norma, uma transação é iniciada pelo SGBDR, e continua até que uma instrução de COMMIT ou ROLLBACK seja emitida.

Os detalhes foram deixados para o SGBD implementar. Os comandos de gerenciamento de transações do SQL padrão estão listados na tabela abaixo:

Comando	Descrição
START (BEGIN) TRANSACTION	Inicializa uma transação SQL e seta as suas características.
SET TRANSACTION	Determina as propriedades da próxima transação SQL para o SQL Agent
SET CONSTRAINTS	Se a transação SQL estiver executando, estabelece o modo de restrições para a transação SQL na sessão corrente. Se não existe nenhuma transação em andamento na sessão, determina ao SQL Agent o modo de execução para a próxima transação.

SAVEPOINT	Estabelece um *savepoint*, ponto intermediário da transação para o qual o *rollback* deve retornar em caso de falha.
RELEASE SAVEPOINT	Destrói um *savepoint*
COMMIT	Termina a transação SQL corrente com um *commit*
ROLLBACK	Termina a transação corrente com um *rollback*, ou desfaz todas as modificações até o último *savepoint*.

O padrão SQL utiliza, de forma implícita, o START TRANSACTION, com uma instrução de COMMIT explícita, nos casos em que todas as unidades lógicas de trabalho de uma transação são concluídas com sucesso. Podemos ter a instrução de ROLLBACK quando mudanças ainda não efetivadas precisam ser desfeitas devido a alguma exceção ou erro.

As implementações dos SGBDs para controle de transação são um pouco diferentes para cada um deles, contudo, a análise detalhada destes comandos foge do escopo deste curso. Vamos passar agora para uma rápida explicação dos níveis de isolamento de transações. Mais detalhes sobre esse assunto serão vistos em aulas específicas sobre transações e controle de concorrência.

Quando executamos várias transações de forma concorrente sobre o mesmo banco de dados, podemos limitar o acesso às informações para impedir que elas sejam vistas por uma transação no momento que estão sendo usadas por outra transação. São quatro os níveis de isolamento definidos pelo SQL ANSI: READ UNCOMMITED, READ COMMITED, REPEATABLE READ e SERIALAZABLE.

No primeiro caso, READ UNCOMMITED, é permitido ler dados que ainda não sofreram o COMMIT de outra transação X que ainda está em execução. Isso acaba possibilitando a existência do problema de leitura suja ou dirty read. Esse problema ocorre se a transação X sofrer ROLLBACK. Neste caso, qualquer outra transação que tiver feito uma consulta sobre o dado modificado por X terá feito uma leitura suja.

Para resolver esse problema, surge o próximo nível de isolamento conhecido como READ COMMITED. Neste caso, uma transação Y só pode ler os dados modificados pela transação X após o COMMIT. Agora, ainda permanecemos com outro problema denominado leitura não repetível. Imagine que Y precisa ler os dados modificados por X duas vezes durante a transação. Na primeira leitura, Y lê os dados modificados por X, contudo, antes da segunda leitura, outra transação Z faz uma nova modificação sobre os dados. Veja que Y faz duas leituras sobre o mesmo dado durante a transação e eles possuem valores distintos.

A solução para os problemas da leitura não repetível está no REPEATABLE READ. Neste caso, a transação Y vai ler os mesmos dados, pois não será permitida que a transação Z faça modificações sobre eles durante a execução de Y. Agora, contamos com um último problema, qual seja, a possibilidade da existência de registros fantasmas. Neste caso, Y faz duas leituras sobre um conjunto de dados ou tuplas de uma relação que satisfação a uma condição descrita na cláusula WHERE.

Suponha que entre a primeira e a segunda leitura seja inserido um registro que também satisfaz a essa condição de busca. Esse registro é considerado um fantasma!

Para resolver todos esses problemas, temos o último nível de isolamento conhecido como SERIALAZABLE. A ideia é executar concorrentemente apenas as transações que produzam o mesmo resultado caso fossem executadas em série.

Além do nível de isolamento, é possível determinarmos o modo de acesso de uma transação que pode ser READ ONLY ou READ WRITE e o tamanho da área de diagnóstico que especifica um valor inteiro n, indicando o número de posições que podem ser manipuladas simultaneamente na área de diagnóstico. Essas condições fornecem informações sobre as condições de execução (erros e exceções), ao usuário ou ao programa, para a maior parte das declarações SQL executadas mais recentemente.

Para executar uma transação usando SQL, podemos utilizar a seguinte sintaxe:


```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSATION
       READ WRITE
       DIAGNOSTIC SIZE 5
       ISOLATION LEVEL SERIALAZABLE
EXEC SQL INSERT INTO EMPREGADO (PNOME, UNOME, SSN, DNO, SALARIO) VALUES
(´Thiago´, ´Cavalcanti´, 000457878, 2, 12000);
EXEC SQL UPDATE EMPREGADO
       SET SALARIO = SALARIO *1,1 WHERE DNO =2;
EXEC COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ...;
```

INDEX (DDL)

Índices são estruturas opcionais associadas às tabelas. Logo, não existe índice se não estiver associado a uma tabela. É possível que a criação dos índices seja feita de forma explícita para melhorar o desempenho na execução de comandos sobre uma tabela. Semelhante a um índice de um livro, os índices em bancos de dados podem prover um caminho de acesso mais rápido para os dados em uma tabela.

Os índices, quando usados apropriadamente, vão permitir uma redução das operações de I/O sobre o disco. A criação de índices pode ser feita sobre um ou mais campos da sua tabela. Essa possibilidade nos leva à primeira taxonomia possível para classificação dos índices, que pode ser simples ou composto.

A criação do índice pode ser feita sobre uma tabela em branco ou uma tabela que já contenha tuplas armazenadas.

Outra classificação para índice está relacionada com uma associação entre os valores existentes no arquivo de índice e os valores armazenados em uma tabela sobre os quais os índices são estruturados. Quando essa relação é de um para um dizemos que o índice é denso. Quando essa relação não é direta entre a chave de pesquisa e a quantidade de valores possíveis, então dizemos que o índice é esparso.

Quando tratamos de SQL, precisamos pensar que a criação do índice é um comando DDL. Ele vai melhorar o desempenho das consultas que recuperam registros por meio do comando SELECT, desde que a(s) coluna(s) que compõem o índice estejam presentes na condição de busca. Por outro lado, a utilização de índices vai degradar o desempenho dos comandos UPDATE, INSERT e DELETE. A utilização destes comandos implica a atualização, mesmo que transparente, do arquivo de índices.

A sintaxe básica do comando descrita na norma SQL/ANSI é a seguinte:

```
CREATE INDEX ix_name ON table_name (col_name);
```

Para destruir ou apagar um índice, basta usar o comando:

```
DROP INDEX ix_name;
```

Apenas para terminar o assunto, gostaria de fazer uma consideração histórica sobre a possibilidade do uso da cláusula UNIQUE no comando de criação dos índices. Essa opção é usada para garantir a unicidade do valor da coluna, implicitamente criada para a(s) coluna(s) da chave primária. Porém, foi descontinuada na maioria dos SGBDs, agora, usamos a restrição de UNIQUE para coluna.

EXTENSÕES PROCEDURAIS

Quando falamos em procedimentos armazenados do ponto de vista de SQL, podemos invocar três diferentes estruturas: as **stored procedures**, os **triggers** e as **user defined functions** (UDFs). Esses padrões estão descritos no SQL/PSM (Persitent Stored Module), uma parte da documentação oficial que tenta incorporar esses conceitos, falamos dela no início da aula.

Desde a sua formulação original no padrão SQL-86, um dos principais inconvenientes que impediram as pessoas de usar o SQL foi a sua falta de fluxo de instruções de controle. Foi então que o SQL/PSM foi incluído no padrão SQL, você não poderia ramificar a ordem sequencial de execução sem utilizar outra linguagem como C ou BASIC. SQL/PSM introduz o fluxo tradicional de estruturas de controle que existem em outras linguagens, permitindo, assim, que programas SQL possam executar funções necessárias sem a utilização de outras linguagens.

Esses recursos processuais definem, entre outros componentes, a criação de rotinas SQL que podem ser invocada explicitamente como procedimentos e funções.

PROCEDIMENTOS ARMAZENADOS

Os **procedimentos armazenados** ou ***stored procedures*** residem no servidor de banco de dados, em vez de executar no cliente - onde todos os procedimentos eram localizados antes do SQL/PSM. Depois de definir um procedimento armazenado, você pode invocá-lo com uma instrução CALL. Mantendo o procedimento armazenado no servidor é possível reduzir o tráfego da rede, acelerando, assim, o desempenho. O único tráfego que precisa ser feito a partir do cliente ao servidor é a instrução CALL. Você pode criar este procedimento da seguinte maneira:

```
EXEC SQL
   CREATE PROCEDURE MatchScore
      ( IN result CHAR (3),
        OUT winner CHAR (5) )
   BEGIN ATOMIC
      CASE result
         WHEN '1-0' THEN
            SET winner = 'white' ;
         WHEN '0-1' THEN
            SET winner = 'black' ;
         ELSE
            SET winner = 'draw' ;
      END CASE
   END ;
```

Depois de ter criado um procedimento armazenado como o descrito neste exemplo, você pode invocá-lo com uma instrução CALL semelhante à seguinte declaração:

```
CALL MatchScore ('1-0', winner);
```

O primeiro argumento é um parâmetro de entrada que alimenta o procedimento MatchScore. O segundo argumento é o parâmetro de saída que o procedimento usa para retornar o resultado da execução da rotina. Neste caso, ele retorna **'white'**. Vejamos como esse assunto já foi cobrado anteriormente.

(Ano: 2016 Órgão: CRO – PR Prova: Analista de Informática)
Qual conceito é uma coleção de comandos em SQL para otimização de Banco de dados e que encapsula tarefas repetitivas, aceita parâmetros de entrada e retorna um valor de status (para indicar aceitação ou falha na execução)?
a) Campos.
b) Índices.
c) Registros.
d) Triggers.
e) Stored Procedures.

Comentário: Perceba que não temos muita dificuldade para responder à questão. Stored Procedure é uma coleção de comandos em SQL para otimização de banco de dados. Encapsula tarefas repetitivas, aceita parâmetros de entrada e retorna um valor de status (para indicar aceitação ou falha na execução). Sendo assim, podemos marcar nossa resposta na alternativa E.

Gabarito: E.

FUCTIONS

A função ou ***function*** armazenada ou UDF é semelhante, em muitos aspectos, a um procedimento armazenado. Coletivamente, os dois são referidos como rotinas armazenadas. Eles são diferentes em vários aspectos, incluindo a forma como são chamados ou executados. Um procedimento armazenado é chamado por meio da instrução CALL, e uma função armazenada é chamado como uma chamada de função, que pode substituir um argumento de uma instrução SQL. A seguir, temos um exemplo de uma definição de função, seguido por um exemplo de uma chamada para essa função:

```
CREATE FUNCTION PurchaseHistory (CustID)
   RETURNS CHAR VARYING (200)
   BEGIN
      DECLARE purch CHAR VARYING (200)
         DEFAULT '' ;
      FOR x AS SELECT *
         FROM transactions t
         WHERE t.customerID = CustID
      DO
         IF a <> ''
            THEN SET purch = purch || ', ' ;
         END IF ;
         SET purch = purch || t.description ;
      END FOR
      RETURN purch ;
   END ;
```

Esta definição de função cria uma lista de compras feitas por um cliente, baseada em um número de cliente especificado, extraída da tabela TRANSACTIONS e separada por vírgulas. A instrução UPDATE a seguir contém uma chamada de função de PurchaseHistory que insere o histórico mais recente de compras do cliente de número 314259 em seu registro na tabela CLIENTE:

```
SET customerID = 314259 ;
UPDATE customer
   SET history = PurchaseHistory (customerID)
   WHERE customerID = 314259 ;
```

TRIGGERS

Até este ponto da aula, você aprendeu a criar uma série de objetos de esquema que você pode acessar ou invocar usando instruções SQL. Por exemplo, você aprendeu como criar tabelas, visões e rotinas. Em cada caso, uma vez que você crie esses objetos, é preciso tomar algum tipo de ação para interagir diretamente com eles, como executar uma instrução SELECT para recuperar dados de uma tabela ou usando uma instrução **CALL** para invocar um procedimento.

No entanto, SQL oferece suporte a objetos que executam ações automaticamente. Esses objetos de esquema, que são conhecidos como gatilhos, respondem às modificações feitas nos dados em uma tabela. Se uma modificação especificada é feita, o gatilho é invocado automaticamente, ou disparado, causando uma ação adicional.

Como resultado, você nunca vai invocar diretamente uma ação definida no gatilho. O trigger vai implicitamente fazer a chamada e a execução dos seus comandos. Abaixo, vamos explorar gatilhos e como eles são usados quando os dados de uma tabela são modificados. A sintaxe básica para a criação de um gatilho é definida abaixo.

```
CREATE TRIGGER <trigger name>
{ BEFORE | AFTER | INSTEAD OF }
{ INSERT | DELETE | UPDATE [ OF <column list> ] }
ON <table or view name> [ REFERENCING <alias options> ]
[ FOR EACH { ROW | STATEMENT } ]
[ WHEN ( <search condition> ) ]
<triggered SQL statements>
```

Vamos dar uma olhada em cada linha. A primeira linha é bastante simples. Você simplesmente fornece um nome para o gatilho após as palavras-chave CREATE TRIGGER. Na segunda linha, você deve designar se o gatilho é chamado antes, depois ou ao invés da modificação sobre os dados especificados na instrução SQL que acionou o gatilho.

Por exemplo, se você definiu um gatilho de inserção, você pode especificar se as instruções SQL acionadas sejam executadas antes (BEFORE) dos dados serem inseridos na tabela. A opção BEFORE é particularmente útil quando uma das tabelas é configurada com uma restrição de integridade referencial.

Na terceira linha da sintaxe, você especifica se o gatilho é de inserção, exclusão ou atualização. Se você definir um **TRIGGER** de atualização, você tem a opção de aplicar o gatilho para uma ou mais colunas específicas. Se mais de uma coluna for especificada, você deve separar os nomes das colunas com vírgulas.

Na próxima linha (4) da sintaxe, você deve especificar uma cláusula **ON**, que inclui o nome da tabela ou visão sobre a qual o gatilho é aplicado. O gatilho pode ser aplicado a apenas uma tabela ou vista. Tenha em mente que gatilhos de **BEFORE** e **AFTER** podem ser aplicados somente a tabelas, enquanto o gatilho de **INSTEAD OF** pode ser aplicado apenas a visões.

Até este ponto, toda a sintaxe que aprendemos é obrigatória, exceto o nome das colunas em definições de gatilhos de atualização, que é opcional. No entanto, as próximas cláusulas não são obrigatórias, mas elas podem adicionar recursos importantes para o seu gatilho.

A primeira destas cláusulas é a cláusula **REFERENCING**. Esta cláusula permite que você especifique como os dados, que utiliza no contexto de execução do gatilho, são referenciados dentro da cláusula WHEN ou nas instruções SQL. Você pode usar uma referência para os valores novos ou antigos das colunas que estão sendo modificadas pelo comando que dispara o gatilho. Veja um exemplo abaixo:

```
CREATE TRIGGER UPDATE_TITLE_COSTS
  AFTER UPDATE ON TITLES_IN_STOCK
  REFERENCING NEW ROW AS New
  FOR EACH ROW
  WHEN ( New.INVENTORY > 20 )
  BEGIN ATOMIC
    UPDATE TITLE_COSTS c
       SET RETAIL = ROUND(RETAIL * 0.9, 2)
     WHERE c.CD_TITLE = New.CD_TITLE;
END;
```

A próxima linha de sintaxe contém a cláusula FOR EACH, que inclui duas opções: ROW ou STATEMENT. Se você especificar ROW, o gatilho é invocado sempre que uma linha for inserida, atualizada ou excluída. Se você especificar STATEMENT, o gatilho é chamado somente uma vez para cada instrução de modificação de dados aplicável que é executada, não importa quantas linhas foram afetadas. Se você não incluir esta cláusula em sua definição de gatilho, a opção STATEMENT é assumida.

Ao avançar na sintaxe temos a cláusula **WHEN** que é opcional e não é suportado para gatilhos de INSTEAD OF. A cláusula WHEN permite definir um critério de pesquisa que limita o escopo do gatilho criando uma condição para ele ser invocado. A cláusula **WHEN** é semelhante à cláusula WHERE de uma instrução SELECT.

Você pode especificar um ou mais predicados que definem uma condição de pesquisa. Se a cláusula WHEN for avaliada como verdadeira, o gatilho é acionado; caso contrário, nenhuma ação será tomada pelo gatilho. No entanto, isso não afeta a instrução de modificação de dados inicial que foi executada contra a tabela em questão; somente as instruções SQL acionadas na definição do gatilho são afetadas.

Finalmente, o último componente que seu **CREATE TRIGGER** deve incluir é uma ou mais instruções SQL que são executados quando o gatilho é chamado. Veja no exemplo mostrado anteriormente uma instrução de UPDATE que é executada pelo gatilho quando a condição definida pela cláusula WHEN é avaliada como verdadeira.

HORA DE PRATICAR!

Ano: 2017 Órgão: UPE Prova: Analista de Sistemas - Banco de Dados

Sobre as estruturas de banco de dados, analise as afirmativas abaixo:

I. Trigger define uma estrutura, que dispara mediante alguma ação, como inserção, exclusão e atualização de dados.

II. Uma trigger não precisa estar associada a uma tabela.

III. Stored Procedure corresponde a um conjunto de comandos em SQL, que podem ser executados de uma só vez, a partir de sua chamada.

IV. Stored Procedure não aceita parâmetros de entrada.

Estão CORRETAS

a) I e II.
b) I e III.
c) I e IV
d) II e IV.
e) II e III.

Comentário: Essa questão trata de gatilhos e procedimentos armazenados. O primeiro refere-se a um recurso de programação executado sempre que um **evento** associado a ele ocorrer. Trigger é um tipo especial de procedimento armazenado, que é executado sempre que há uma tentativa de modificar os dados de uma tabela que é protegida por ele. Veja que, a partir deste comentário, podemos considerar as afirmações I e II, respectivamente, como correta e incorreta.

Já a Stored Procedure é um conjunto de comandos ao qual é atribuído um nome. Este conjunto fica armazenado no **banco de dados** e pode ser chamado a qualquer momento tanto pelo **SGBD** quanto por um sistema que faz interface com ele. Sendo assim, podemos concluir que a afirmação III está correta. Sobre a afirmação IV, se lembrarmos do que vimos anteriormente nesta aula, podemos concluir que está incorreta, pois as SP aceitam parâmetros de entrada.

Gabarito: B.

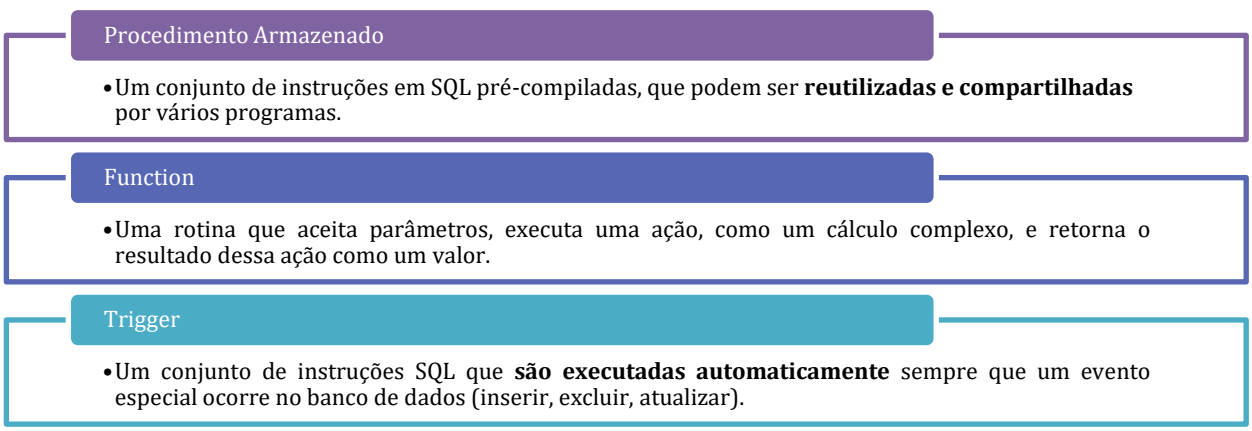
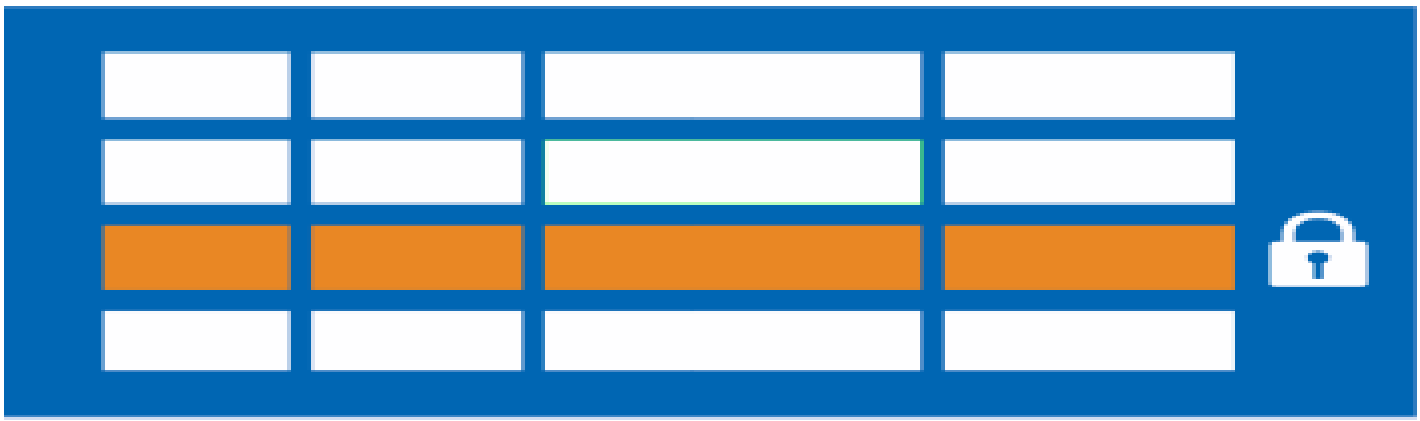


Figura 1 - Extensões procedurais

SEGURANÇA (UTILIZANDO A DCL)



As instruções SQL que você usa para criar bancos de dados formam um grupo conhecido como a linguagem de definição de dados (DDL). Depois de criar um banco de dados, você pode usar outro conjunto de instruções SQL - conhecidos coletivamente como Data Manipulation Language (DML) - para adicionar, alterar e remover os dados do banco de dados. SQL inclui declarações adicionais que não se enquadram em nenhuma dessas categorias.

Os programadores, por vezes, referem-se a estas declarações coletivamente como a Linguagem de Controle de Dados (DCL). Declarações ou instruções DCL protegem o banco de dados contra acessos não autorizados, a partir da interação prejudicial entre vários usuários de banco de dados. Neste bloco, tratamos da proteção contra acesso não autorizado.

O SQL fornece acesso controlado a nove funções de gerenciamento de banco de dados: INSERT, SELECT, UPDATE, DELETE, REFERENCES, USAGE, UNDER, TRIGGER e EXECUTE.

Os quatro primeiros estão relacionados a instruções de manipulação de dados. O REFERENCES envolve a autorização do uso da integridade referencial em uma tabela que depende de outra tabela. A palavra-chave USAGE diz respeito a domínios, conjuntos de caracteres, *collations* e *translations*. Usamos o UNDER quando estamos tratando de tipos definidos pelo usuário. O TRIGGER garante a autorização para executar o comando quando um evento predeterminado acontece. O EXECUTE roda uma determinada rotina.

Quando possuímos permissões destes tipos sobre os objetos podemos executar as funções autorizadas sobre eles.

A HIERARQUIA DO PRIVILÉGIO DE ACESSO

Para entendermos o que essa hierarquia significa, vamos observar a figura abaixo. Na maioria das instalações, existe uma hierarquia de privilégios de usuário, em que o DBA está no mais alto nível e o PUBLIC no menor.

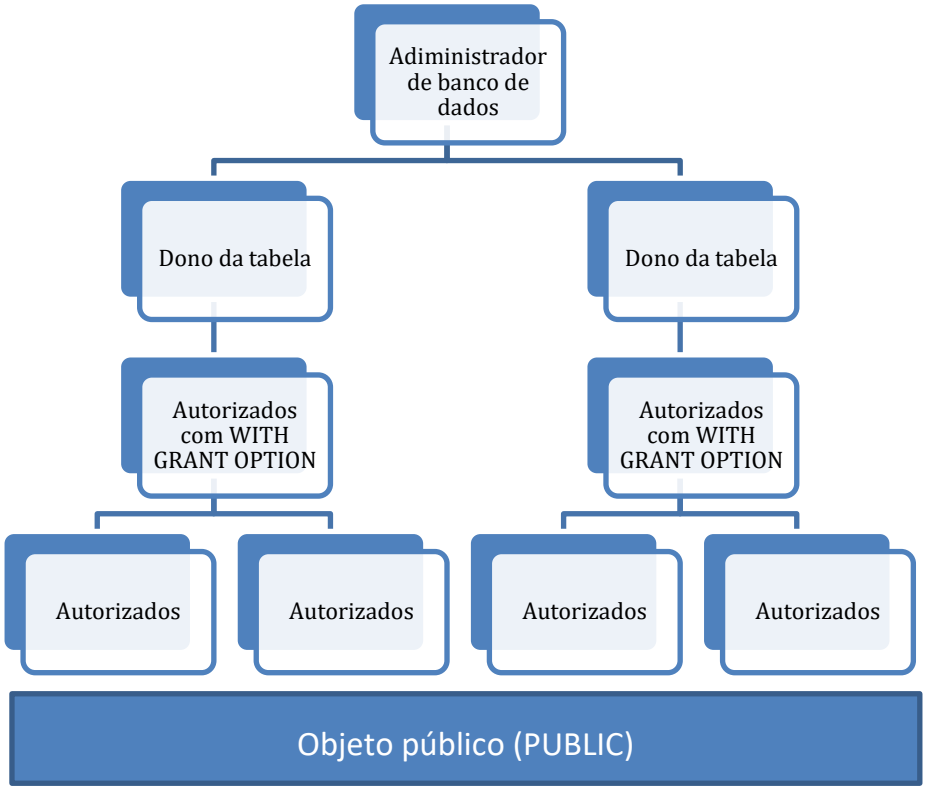


Figura 1 - Hierarquia de privilégios e permissões

Em termos de rede, "PUBLIC" diz respeito a todos os usuários que não são usuários privilegiados (isto é, nem DBAs ou proprietários de objetos) e para quem um usuário privilegiado não concedeu direitos de acesso especificamente. Se um usuário concede determinados direitos de acesso privilegiados ao PUBLIC, então, todos que podem acessar o sistema ganham esses direitos.

GARANTINDO PRIVILÉGIOS AOS USUÁRIOS

O DBA, em virtude de sua posição, tem todos os privilégios em todos os objetos do banco de dados. O proprietário de um objeto também tem todos os privilégios em relação a esse objeto – o próprio banco de dados é um objeto. Ninguém mais tem qualquer privilégio em relação a qualquer objeto, a menos que alguém que já tem esses privilégios (e autoridade para passá-los) concede especificamente os privilégios. Você concede privilégios para alguém usando a instrução GRANT, que tem a seguinte sintaxe:

```
GRANT lista-de-privilégios
   ON objeto
   TO lista-de-usuários
   [WITH GRANT OPTION]
   [GRANTED BY autorizador];
```

Neste comando, a lista de privilégios pode ser uma lista separada por vírgulas ou ALL PRIVILEGES. Este último vai garantir o privilégio em todo o conjunto dos nove privilégios que tratamos anteriormente, são eles:

```
SELECT
| DELETE
| INSERT [(column-name [, column-name] ...)]
| UPDATE [(column-name [, column-name] ...)]
| REFERENCES [(column-name [, column-name] ...)]
| USAGE
| UNDER
| TRIGGER
| EXECUTE
```

Os objetos definidos no comando podem ser os seguintes:

```
[ TABLE] <table name>
| DOMAIN <domain name>
| COLLATION <collation name>
| CHARACTER SET <character set name>
| TRANSLATION <transliteration name>
| TYPE <schema-resolved user-defined type name>
| SEQUENCE <sequence generator name>
| <specific routine designator>
```

Para finalizar o comando, a lista de usuários pode ser um conjunto separado por vírgulas ou PUBLIC. E o garantidor/autorizador (grantor) é o usuário corrente (CURRENT_USER) ou o papel (CURRENT_ROLE). Vamos aproveitar que tratamos de ROLES para descrever um pouco mais sobre suas características.

ROLES

Um nome de usuário é um tipo de identificador de autorização, mas não é o único. Ele identifica uma pessoa (ou um programa) autorizada a executar uma ou mais funções em uma base de dados. Em uma grande organização com muitos usuários, a concessão de privilégios para cada funcionário individualmente pode ser tediosa e demorada. SQL resolve este problema introduzindo a noção de papéis.

Um papel, identificado por um nome, é um conjunto de zero ou mais privilégios que podem ser concedidos a várias pessoas, pois todas elas exigem o mesmo nível de acesso ao banco de dados. Por exemplo, todos os que desempenham o papel contador têm os mesmos privilégios. Esses privilégios são diferentes dos concedidos às pessoas que têm a função de balconista.

Essa não é uma característica mencionada na última versão da especificação SQL, mas está disponível em todas as implementações de SGBDs. Verifique a documentação antes de tentar usar papéis.

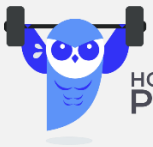
Você pode criar um papel usando uma sintaxe semelhante a seguinte:

```
CREATE ROLE balconista;
```

Depois de criar um papel, você pode atribuir pessoas ao papel com a instrução GRANT, da seguinte forma:

```
GRANT balconista TO Jose;
```

Você pode conceder privilégios a um papel exatamente da mesma maneira que você conceder privilégios para usuários. Vamos ver como esse assunto já foi cobrado em provas anteriores por meio de uma questão da FCC.



Ano: 2017 Órgão: TST Prova: Analista Judiciário – Suporte em Tecnologia da Informação

Um Database Administrator − DBA deseja criar uma função chamada analista, atribuir o privilégio create table a ela e atribuí-la ao usuário pedro. Para isso, terá que usar as instruções

a) CREATE FUNCTION analista;
ADD PRIVILEGE create table TO analista;
GRANT analista TO pedro;

b) CREATE ROLE analista;
GRANT ADD create table TO analista;
ADD ROLE analista TO pedro;

c) CREATE FUNCTION analista;
GRANT create table TO analista;
ADD FUNCTION analista TO pedro;

d) CREATE ROLE analista;
GRANT create table TO analista;
GRANT analista TO pedro;

e) CREATE ROLE analista;
GRANT create table TO analista;
ADD analista TO Pedro WITH GRANT OPTION;

Comentário: Questão muito interessante! Aborda de forma consistente o que acabamos de aprender. Perceba que a função que agrega permissões tem o nome de ROLE (perfil) no SQL. Após criar esse perfil, é possível associar a ele diversas permissões. No caso da questão, queremos permitir que o perfil possa criar tabelas. Por fim, vamos associar o perfil ao usuário de nome Pedro. Perceba que essa sequência de passos pode ser corretamente executada pelos seguintes comandos:

```
CREATE ROLE analista;
GRANT create table TO analista;
GRANT analista TO pedro;
```

Essa sequência está presente na alternativa D, que é a nossa resposta.

Gabarito: D.

WITH GRANT OPTION

O DBA pode conceder qualquer privilégio. Um proprietário de um objeto pode conceder quaisquer privilégios sobre esse objeto a qualquer pessoa ou role. Mas, os usuários que recebem privilégios desta forma não podem, por sua vez, conceder esses privilégios para outra pessoa. Esta restrição ajuda o DBA ou o proprietário da tabela a manter o controle sobre o objeto. Apenas os usuários que

o DBA ou o proprietário do objeto atribui competência para executar a operação em questão pode fazê-la.

Do ponto de vista de segurança, colocar limites na capacidade de delegar privilégios de acesso faz muito sentido. Muitas vezes surgem, no entanto, situações em que os usuários precisam do poder de delegar ou repassar seus privilégios. O trabalho não pode sofrer uma parada brusca cada vez que alguém está doente, em férias, ou saiu para almoçar.

Você pode confiar a alguns usuários o poder de delegar seus direitos de acesso. Para repassar esse direito de delegação para um usuário, o GRANT usa a cláusula **WITH GRANT OPTION**. A declaração a seguir mostra um exemplo de como você pode usar essa cláusula:

```
GRANT UPDATE (BonusPct)
  ON BONUSRATE
  TO SalesMgr
  WITH GRANT OPTION;
```

Agora, o gerente de vendas pode delegar o privilégio UPDATE emitindo a seguinte declaração:

```
GRANT UPDATE (BonusPct)
  ON BONUSRATE
  TO AsstSalesMgr;
```

Após a execução desta declaração, qualquer pessoa com o papel de gerente de vendas assistente pode fazer alterações para a coluna BonusPct da tabela de BONUSRATE. Você faz uma troca entre segurança e conveniência quando você delega direitos de acesso a um suplente. O proprietário da tabela de BONUSRATE abandona o controle considerável na concessão do privilégio UPDATE para o gerente de vendas usando o WITH GRANT OPTION.

REMOVENDO PRIVILÉGIOS

Se você tem uma maneira de dar privilégios de acesso para as pessoas, você também deve ter uma forma de tirar os privilégios. As funções de trabalho das pessoas mudam, e com essas mudanças as suas necessidades de acesso de dados mudam. Digamos que um funcionário deixe a organização para trabalhar em um concorrente. Você provavelmente deve revogar todos os privilégios de acesso dessa pessoa - imediatamente.

SQL permite remover privilégios de acesso usando a instrução REVOKE. Esta declaração age como a instrução GRANT faz, exceto que ele tem o efeito inverso. A sintaxe para essa instrução é a seguinte:

```
REVOKE [GRANT OPTION FOR] privilege-list
  ON object
  FROM user-list [RESTRICT|CASCADE];
```

Você pode usar essa estrutura para revogar os privilégios especificados. A principal diferença entre a instrução REVOKE e a instrução GRANT é a presença da palavra-chave opcional RESTRICT ou CASCADE na instrução REVOKE.

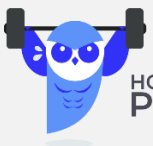
Por exemplo, suponha que você usou a cláusula WITH GRANT OPTION quando concedeu determinados privilégios a um usuário. Eventualmente, quando você deseja revogar os privilégios, você pode usar CASCADE na instrução REVOKE. Quando você revogar os privilégios de um usuário, desta forma, você também retira os privilégios de qualquer pessoa a quem essa pessoa tenha concedido privilégios.

Por outro lado, a declaração REVOKE com a opção RESTRICT funciona apenas se o beneficiário não delegou os privilégios especificados. Nesse caso, a instrução REVOKE revoga privilégios do beneficiário apenas. Mas, se o usuário passou os privilégios especificados, a instrução REVOKE com a opção RESTRICT não revoga nada – e, em vez disso, retorna um código de erro. Este é um aviso claro para você que você precisa descobrir para quem foram concedidos os privilégios pela pessoa cujos privilégios você está tentando revogar. Você pode ou não querer revogar os privilégios dessa pessoa.

Você pode usar uma instrução REVOKE com a cláusula opcional GRANT OPTION FOR de revogar apenas a opção de concessão de privilégios especificados ao mesmo tempo permitindo que o beneficiário mantenha os privilégios para si mesmo. Se a cláusula GRANT OPTION FOR e a palavra-chave CASCADE estiverem presentes, você vai revogar todos os privilégios que o beneficiado concedeu, juntamente com o direito do beneficiário de conceder tais privilégios – funciona como se você nunca tivesse concedido a opção de concessão. Se a cláusula GRANT OPTION FOR e a cláusula RESTRICT estiverem presentes, uma de duas coisas acontece:

1. Se o beneficiário não concedeu a qualquer pessoa um dos privilégios que você está revogando, a instrução REVOKE executa e remove a capacidade do beneficiário de conceder privilégios.
2. Se o beneficiário já concedeu pelo menos um dos privilégios que você está revogando, a instrução REVOKE não é executada e retorna um código de erro.

Antes de continuarmos com nosso assunto teórico, vamos fazer mais uma questão que trata do assunto.



(Ano: 2016 Órgão: IF-BA Prova: Analista de Tecnologia da Informação – Infraestrutura)
Um dos mecanismos de segurança em um sistema de banco de dados é o subsistema de autorização, que permite a usuários que têm privilégios específicos concederem de forma seletiva e dinâmica esses privilégios a outros usuários e, subsequentemente, revogarem esses privilégios, se desejarem. Os comandos SQL que permitem a um usuário conceder privilégios a outros usuários e revogar privilégios concedidos a outros usuários são, respectivamente:

a) INSERT PRIVILEGES e DELETE PRIVILEGES.
b) CREATE ROLE e DROP ROLE.
c) CONCEDE e EXCLUDE.
d) GRANT e REVOKE.
e) ALLOW e DISALLOW.

Comentário: A questão pede para que você associe o comando que concede privilégios e que os retira. Acabamos de conhecer esses dois comandos, são, respectivamente, o **GRANT** TO e **REVOKE** FROM. Por que eu coloquei as preposições? Porque em algumas provas as bancas colocam proposições erradas para invalidar algumas alternativas. Então, muito cuidado! 😊 Para a questão em voga não precisamos deste detalhe, conseguimos observar nossa resposta na alternativa D.

Gabarito: D.

QUESTÕES COMENTADAS

1. (MPU-TO - DBA - CESPE – 2024)

Julgue os itens subsequentes, com relação à linguagem de consulta estruturada (SQL).

O comando HAVING é usado para unir duas ou mais tabelas.

Comentário:

O item está incorreto. O comando HAVING em SQL não é usado para unir duas ou mais tabelas. Na verdade, o HAVING é usado em conjunto com a cláusula GROUP BY para filtrar registros após a agregação, enquanto a cláusula WHERE é usada para filtrar registros antes da agregação. Portanto, o HAVING é usado para especificar uma condição de filtragem em grupos resultantes de uma operação GROUP BY, e não para unir tabelas.

Gabarito: Errada

2. (MPU-TO - DBA - CESPE – 2024)

Julgue os itens subsequentes, com relação à linguagem de consulta estruturada (SQL).

O comando TRUNCATE é usado para remover todas as linhas de uma tabela.

Comentário:

O item está correto. O comando TRUNCATE em SQL é usado para remover todas as linhas de uma tabela, deixando a estrutura da tabela intacta. Ele é mais rápido do que o comando DELETE, pois não gera logs de transações para cada linha excluída, e também não aciona acionadores (triggers) associados à tabela. Portanto, o comando TRUNCATE é comumente utilizado quando se deseja limpar rapidamente o conteúdo de uma tabela.

Gabarito: Certa

3. (MPU-TO - DBA - CESPE – 2024)

A respeito de arquitetura de bancos de dados, julgue os itens a seguir.

O esquema do banco de dados é alterado a cada comando de INSERT, UPDATE ou DELETE em uma de suas tabelas.

Comentário:

O item está incorreto. O esquema de um banco de dados, que define a estrutura geral do banco de dados, não é alterado a cada comando de INSERT, UPDATE ou DELETE em uma de suas tabelas. Esses comandos modificam os dados dentro das tabelas, mas não afetam a estrutura do esquema do banco de dados em si. Alterações no esquema ocorrem quando são executadas instruções DDL (Data Definition Language), como CREATE, ALTER ou DROP, que são usadas para criar, modificar ou excluir objetos de banco de dados, como tabelas, índices, visões, entre outros. Portanto, a afirmação de que o esquema do banco de dados é alterado a cada comando de INSERT, UPDATE ou DELETE está incorreta. O gabarito está errado.

Gabarito: Errada

4. (MPU-TO - DBA - CESPE – 2024)

Quanto a linguagens de consulta, julgue os próximos itens.

Em uma consulta SQL, tuplas duplicadas são automaticamente eliminadas pelo SGBD, com o objetivo de aumentar o desempenho.

Comentário:

O item está incorreto. Em uma consulta SQL, as tuplas duplicadas não são automaticamente eliminadas pelo SGBD. Se houver necessidade de eliminar tuplas duplicadas, é necessário explicitamente utilizar a cláusula DISTINCT na consulta. A cláusula DISTINCT é usada para garantir que apenas valores únicos sejam retornados pela consulta. Portanto, a afirmação de que as tuplas duplicadas são automaticamente eliminadas pelo SGBD está incorreta. O gabarito está errado.

Gabarito: Errada

5. (MPU-TO - DBA - CESPE – 2024)

Quanto a linguagens de consulta, julgue os próximos itens.

Em uma tabela com 100 registros, uma consulta SQL com a cláusula WHERE pode apresentar resultados com quantidade variando de zero a 100 linhas.

Comentário:

O item está correto. Uma consulta SQL com a cláusula WHERE pode retornar qualquer quantidade de linhas entre zero e o total de registros na tabela, dependendo das condições especificadas na cláusula WHERE e dos dados presentes na tabela. Se a condição especificada na cláusula WHERE não for satisfeita por nenhum registro na tabela, nenhum resultado será retornado (zero linhas). Por outro lado, se a condição for satisfeita por todos os registros na

tabela, todos os registros serão retornados (100 linhas, no caso de uma tabela com 100 registros). A quantidade de linhas retornadas pode variar dependendo das condições da consulta e dos dados na tabela. Portanto, o item está correto.

Gabarito: Certa

6. (MPU-TO - DBA - CESPE – 2024)

Quanto a linguagens de consulta, julgue os próximos itens.

A SQL permite a consulta simultânea de atributos com o mesmo nome, desde que a ordem desses atributos seja a mesma da ordem das tabelas onde eles estão armazenados.

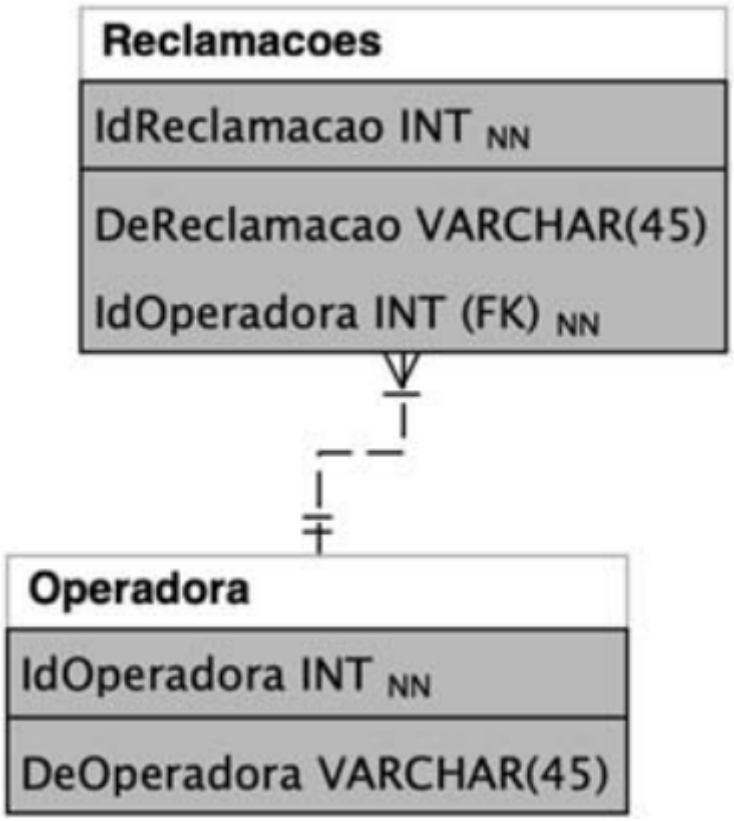
Comentário:

O item está incorreto. Em SQL, é possível realizar consultas que envolvam a seleção de atributos com o mesmo nome de tabelas diferentes, independentemente da ordem em que esses atributos são listados nas tabelas. Isso é conhecido como junção de tabelas. Na verdade, a ordem dos atributos nas tabelas não importa para a consulta. O que importa é que os atributos correspondentes em cada tabela sejam referenciados corretamente na cláusula SELECT da consulta. Portanto, o item está errado.

Gabarito: Errada

7. (ANAC - Analista de Sistemas - CESPE – 2024)

Figura CB1A5



No que se refere a banco de dados, julgue os itens a seguir, tendo o modelo apresentado na figura CB1A5 como referência.

```
CREATE TABLE Operadora (
  IdOperadora INT NOT NULL,
  DeOperadora VARCHAR(45) NULL,
  PRIMARY KEY (IdOperadora));

CREATE TABLE Reclamacoes (
  IdReclamacao INT NOT NULL AUTO_INCREMENT,
  DeReclamacao VARCHAR(45) NULL,
  IdOperadora INT NOT NULL,
  PRIMARY KEY (IdReclamacao),
    FOREIGN KEY (IdOperadora)
    REFERENCES Operadora (IdOperadora));
```

37

A execução dos comandos SQL precedentes resulta na criação de duas tabelas e no relacionamento descrito no modelo apresentado na figura CB1A5.

Comentário:

Esses comandos SQL estão relacionados à criação de tabelas em um banco de dados, e foram construídos com base no diagrama apresentado. Vamos analisar cada comando:

1. CREATE TABLE Operadora: Este comando cria uma tabela chamada "Operadora". A tabela terá uma coluna chamada "IdOperadora" do tipo INT (inteiro) que não pode ser nula (NOT NULL) e uma coluna chamada "DeOperadora" do tipo VARCHAR (cadeia de caracteres) com tamanho máximo de 45 caracteres que pode ser nula (NULL). A coluna "IdOperadora" é definida como a chave primária (PRIMARY KEY) da tabela, o que significa que cada valor nessa coluna deve ser único e não nulo.

2. CREATE TABLE Reclamacoes: Este comando cria uma tabela chamada "Reclamacoes". A tabela terá uma coluna chamada "IdReclamacao" do tipo INT que não pode ser nula e terá valores autoincrementados (AUTO INCREMENT), ou seja, o banco de dados irá gerar automaticamente valores para essa coluna. A tabela também terá uma coluna chamada "DeReclamacao" do tipo VARCHAR com tamanho máximo de 45 caracteres que pode ser nula. Além disso, haverá uma coluna chamada "IdOperadora" do tipo INT que não pode ser nula. A coluna "IdReclamacao" é definida como a chave primária da tabela. A última linha desse comando define uma restrição de chave estrangeira (FOREIGN KEY) na coluna "IdOperadora", referenciando a coluna "IdOperadora" da tabela "Operadora". Isso significa que o valor na coluna "IdOperadora" da tabela "Reclamacoes" deve corresponder a um valor existente na coluna "IdOperadora" da tabela "Operadora", garantindo assim a integridade referencial entre as duas tabelas.

Esses comandos SQL foram construídos com base em um diagrama de banco de dados que representa as entidades (Operadora e Reclamacoes) e seu relacionamento. O diagrama incluí informações sobre os atributos de cada entidade e o relacionamento entre elas, o que foi traduzido para os comandos SQL para criar as tabelas correspondentes no banco de dados.

Gabarito: Certa

8. (ANAC - Analista de Sistemas - CESPE – 2024)

IdOperadora	DeOperadora
1	Operadora A
2	Operadora B

IdReclamacao	DeReclamacao	IdOperadora
10	CDC	1
20	CDC	1
30	CDC	1
50	ABC	2
60	ABC	2

38

Considerando que as tabelas do modelo apresentado na figura CB1A5 possuam os registros precedentes, é correto afirmar que a execução do seguinte comando

```
SELECT o.Deoperadora, COUNT(r.idreclamacao)
AS Quantidade
FROM Reclamacoes r
INNER JOIN Operadora o ON o.idoperadora =
r.IdOperadora
GROUP BY o.Deoperadora
```

terá como resultado a tabela a seguir.

DeOperadora	Quantidade
Operadora A	3
Operadora B	2

Comentário:

O comando SQL realiza uma consulta em um banco de dados e retorna o nome da operadora junto com a contagem de reclamações associadas a cada operadora. Vamos explicar cada parte do comando:

1. SELECT o.Deoperadora, COUNT(r.idreclamacao) AS Quantidade: Esta parte do comando especifica quais colunas devem ser retornadas como resultado da consulta. "o.Deoperadora" representa o nome da operadora, enquanto "COUNT(r.idreclamacao)" calcula o número de reclamações para cada operadora. O alias "Quantidade" é atribuído à contagem de reclamações para facilitar a referência posterior.

2. FROM Reclamacoes r INNER JOIN Operadora o ON o.idoperadora = r.IdOperadora: Aqui, é especificada a fonte dos dados para a consulta. O comando FROM indica que estamos consultando as tabelas "Reclamacoes" (com alias "r") e "Operadora" (com alias "o"). O INNER JOIN é usado para combinar registros das duas tabelas com base em uma condição específica. Neste caso, estamos combinando os registros onde o "IdOperadora" na tabela "Reclamacoes" é igual ao "IdOperadora" na tabela "Operadora". Isso cria uma relação entre as reclamações e as operadoras associadas a elas.

3. GROUP BY o.Deoperadora: Esta cláusula agrupa os resultados da consulta com base nos valores únicos na coluna "Deoperadora" da tabela "Operadora". Isso significa que todas as reclamações associadas a uma determinada operadora serão agrupadas juntas.

Em resumo, este comando SQL retorna o nome de cada operadora juntamente com o número de reclamações recebidas por cada uma delas, agrupadas por operadora. Isso permite uma análise rápida e resumida do número de reclamações associadas a cada operadora no banco de dados.

Gabarito: Certa

9. TST/ Cesbraspe/2024/TI

Assinale a opção correspondente ao comando utilizado para adicionar, remover ou modificar colunas de tabelas de bancos de dados SQL.

a) delete
b) update
c) insert
d) create
e) alter

Comentário:

A opção correta é: e) alter. O comando utilizado para adicionar, remover ou modificar colunas de tabelas de bancos de dados SQL é o comando ALTER. Este comando permite fazer alterações na estrutura de uma tabela, como adicionar, modificar ou remover colunas.

No SQL, o comando ALTER é utilizado para fazer alterações na estrutura de objetos de banco de dados, como tabelas. Especificamente, o ALTER TABLE é usado para modificar uma tabela existente. Aqui está uma explicação mais detalhada:

1. Adicionar coluna: Você pode adicionar uma nova coluna a uma tabela existente usando o comando ALTER TABLE seguido da cláusula ADD COLUMN. Por exemplo:

```
ALTER TABLE nome_da_tabela ADD COLUMN nome_da_nova_coluna tipo_de_dados;
```

Isso adiciona uma nova coluna à tabela especificada.

2. Modificar coluna: Para modificar uma coluna existente, você pode usar o ALTER TABLE com a cláusula ALTER COLUMN. Isso permite modificar o tipo de dados ou as restrições de uma coluna. Por exemplo:

```
ALTER TABLE nome_da_tabela ALTER COLUMN nome_da_coluna TIPO_NOVO;
```

Isso altera o tipo de dados da coluna especificada.

3. Remover coluna: Para remover uma coluna de uma tabela, use o ALTER TABLE com a cláusula DROP COLUMN. Por exemplo:

```
ALTER TABLE nome_da_tabela DROP COLUMN nome_da_coluna;
```

Isso removerá a coluna especificada da tabela.

O comando ALTER TABLE oferece grande flexibilidade para fazer mudanças na estrutura de tabelas existentes, permitindo ajustes conforme necessário sem ter que recriar a tabela do zero.

Gabarito: E

10. TST/ Cesbraspe/2024/TI

Assinale a opção que indica o comando que só pode ser definido para executar em nível de instrução no banco de dados PostgreSQL.

a) after
b) truncate
c) update
d) instead of
e) before

Comentário:

Comandos que podem ser executados em nível de instrução no PostgreSQL geralmente são relacionados à manipulação de dados ou à execução de consultas.

O comando TRUNCATE é um exemplo de um comando que é executado em nível de instrução no PostgreSQL. O TRUNCATE é usado para remover todos os registros de uma tabela, mas ao contrário do comando DELETE, ele não dispara triggers e é mais rápido, pois não registra cada exclusão individualmente no log de transações. Em vez disso, ele libera todo o espaço ocupado pela tabela de uma vez.

Aqui está um exemplo de uso do comando TRUNCATE:

TRUNCATE TABLE nome_da_tabela;

Este comando remove todos os registros da tabela especificada (nome_da_tabela) sem acionar gatilhos associados à exclusão de registros.

Gabarito: B

11. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

Com pertinência à linguagem SQL, julgue o item abaixo.

Considere-se o seguinte *script* SQL.

```
select report_code, year, month, day,
wind_speed,
case
  when wind_speed>= 40 then 'HIGH'
  when wind_speed>= 30 then 'MODERATE'
```

```
else 'LOW'
end as wind_severity
from station_data
```

O resultado da execução do script resultará em erro, pois, caso haja, na tabela station_data, algum registro no campo wind_speed com valor superior a 40, não será possível predizer se o valor da variável wind_severity será igual a 'HIGH'.

Comentário:

O item está incorreto. O script SQL fornecido não resultará em erro com base na condição fornecida. Na verdade, o script está definindo a variável **wind_severity** com base em uma condição de **CASE**.
O que o script faz é o seguinte:

- Ele seleciona várias colunas, incluindo **report_code**, **year**, **month**, **day** e **wind_speed** da tabela **station_data**.
- A coluna **wind_severity** é criada com base em uma condição usando a expressão **CASE**. Se o valor de **wind_speed** for maior ou igual a 40, ele será definido como 'HIGH'. Se for maior ou igual a 30, será definido como 'MODERATE'. Caso contrário, será definido como 'LOW'.

Portanto, não importa se há registros na tabela **station_data** com valores de **wind_speed** superiores a 40. A consulta simplesmente avaliará cada linha da tabela com base na condição fornecida e definirá o valor de **wind_severity** de acordo com essa condição. Não há erro na consulta.

Gabarito: Errado

12. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

A respeito de banco de dados, julgue o próximo item.

Em um comando SELECT, a cláusula WHERE define que o resultado da consulta é o produto cartesiano das tabelas envolvidas.

Comentário:

O item está incorreto. A cláusula **WHERE** em um comando **SELECT** não define que o resultado da consulta é o produto cartesiano das tabelas envolvidas. Na verdade, a cláusula **WHERE** é usada para filtrar as linhas de uma tabela com base em uma condição específica. Ela não afeta a forma como as tabelas são combinadas em uma consulta.
A combinação de tabelas em uma consulta é geralmente feita usando cláusulas como **FROM**, que especificam como as tabelas estão relacionadas entre si. O produto cartesiano ocorre quando não há uma cláusula **JOIN** e nenhuma relação explícita entre as tabelas, o que é raro na prática.

Portanto, a cláusula **WHERE** é usada para restringir as linhas retornadas em uma consulta, mas não define o tipo de junção entre tabelas.

Gabarito: Errado

13. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

A respeito de banco de dados, julgue o próximo item.

Em SQL, o comando DISTINCT é utilizado para eliminar resultados repetidos em consultas a tabelas do banco de dados.

Comentário:

O item está correto. Em SQL, o comando DISTINCT é usado para eliminar resultados repetidos em consultas a tabelas do banco de dados. Quando você utiliza DISTINCT em uma consulta, o SGBD (Sistema de Gerenciamento de Banco de Dados) retorna apenas valores únicos para a(s) coluna(s) especificada(s) na consulta, removendo duplicatas.
Por exemplo:

```
SELECT DISTINCT nome FROM clientes;
```

O comando acima retornará apenas nomes únicos da tabela clientes, eliminando qualquer nome duplicado que possa estar presente na tabela.

Gabarito: Certo

14. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

Julgue o item seguinte a respeito dos conceitos de administração de dados.

Os comandos TRUNCATE e DROP TABLE removem todas as linhas de uma tabela, porém o comando DROP TABLE exclui também a estrutura da tabela do banco de dados bem como todos os dados armazenados na tabela.

Comentário:

O item menciona corretamente que os comandos TRUNCATE e DROP TABLE removem todas as linhas de uma tabela.

- O comando TRUNCATE remove todas as linhas de uma tabela, mas mantém a estrutura da tabela no banco de dados. Ele é usado para excluir todos os registros de uma tabela, mas a tabela em si ainda existe.

- O comando DROP TABLE, por outro lado, não apenas remove todas as linhas de uma tabela, mas também elimina completamente a estrutura da tabela do banco de dados. Isso

significa que a tabela é excluída juntamente com todos os seus dados e metadados. Depois de executar o DROP TABLE, a tabela não existe mais no banco de dados.

Portanto, o item está correto! O DROP realmente remove a estrutura da tabela.

Gabarito: Certo

15. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

Considerando os conceitos de tuning de banco de dados, julgue o item a seguir.

O comando EXPLAIN permite otimizar tabelas que executam muitas operações de UPDATE e DELETE em detrimento de operações de INSERT.

Comentário:

É importante saber que o MySQL, assim como alguns outros sistemas de gerenciamento de banco de dados (SGBDs), permite o uso do comando EXPLAIN em comandos de INSERT, UPDATE, DELETE, entre outros. O EXPLAIN é uma ferramenta muito útil para analisar como o otimizador de consulta do SGBD planeja executar uma operação e pode ser usada para otimizar o desempenho das operações mencionadas.

Entretanto a afirmativa está errada! E onde está o erro da questão, professor? O erro está em dizer que o EXPLAIN otimiza tabelas quando na realidade a otimização é de consultas e outros comandos. A questão que menciona o uso do EXPLAIN para otimizar tabelas ou operações de INSERT, UPDATE e DELETE está incorreta na forma como foi apresentada. O EXPLAIN fornece informações sobre como o otimizador de consulta planeja executar uma consulta, ajudando os desenvolvedores a ajustar as consultas para obter um melhor desempenho, mas não é uma ferramenta direta para otimizar tabelas ou operações de manipulação de dados.

Gabarito: Errado

16. CEBRASPE (CESPE) - Per Crim (POLC AL)/POLC AL/Análise de Sistemas, Ciências da Computação, Informática. Processamento de Dados ou Sistemas da Informação/2023

Com relação aos componentes de um computador, aos barramentos de E/S, à aritmética computacional e à linguagem SQL, julgue o próximo item.

Em SQL, para que não haja erro de construção (sintaxe), as cláusulas GROUP BY e HAVING, quando usadas, devem ser definidas sempre antes da cláusula WHERE.

Comentário:

A alternativa está incorreta. No SQL, a ordem das cláusulas em uma consulta não é estritamente definida pela sintaxe em termos de cláusulas GROUP BY, HAVING e WHERE. A ordem das cláusulas depende do que você deseja realizar na consulta, mas existem algumas regras gerais a serem seguidas:

1. A cláusula FROM deve aparecer antes das cláusulas WHERE, GROUP BY, HAVING, SELECT, ORDER BY e LIMIT.
2. A cláusula WHERE geralmente é colocada após a cláusula FROM para filtrar os registros antes de qualquer agregação ou agrupamento.
3. A cláusula GROUP BY é usada para agrupar registros após a cláusula WHERE, se houver, e antes da cláusula HAVING.
4. A cláusula HAVING é usada para filtrar grupos após a cláusula GROUP BY.
5. A cláusula SELECT é usada para especificar as colunas a serem retornadas na consulta.
6. A cláusula ORDER BY é usada para ordenar o resultado da consulta.
7. A cláusula LIMIT é usada para limitar o número de linhas retornadas.

Portanto, a cláusula GROUP BY deve aparecer antes da cláusula HAVING na maioria dos casos, pois você primeiro agrupa os registros e, em seguida, aplica condições de filtragem aos grupos usando o HAVING. No entanto, a cláusula WHERE é geralmente colocada antes de ambas.

Em resumo, a ordem geral das cláusulas em uma consulta SQL é importante, mas a ordem específica entre GROUP BY, HAVING e WHERE pode variar dependendo da lógica da consulta que você está escrevendo. Portanto, o item está incorreto ao afirmar que GROUP BY e HAVING devem ser definidos sempre antes da cláusula WHERE.

Gabarito: Errado

17. CEBRASPE (CESPE) - Per Crim (POLC AL)/POLC AL/Análise de Sistemas, Ciências da Computação, Informática. Processamento de Dados ou Sistemas da Informação/2023

Com relação aos componentes de um computador, aos barramentos de E/S, à aritmética computacional e à linguagem SQL, julgue o próximo item.

Ao ser executado, o comando SQL a seguir mostrará o CPF e o nome de todas as pessoas que cometeram pelo menos um crime do tipo hediondo.

```
select P.CPF, P.NOME
from POLITEC.PESSOA P
where P;CPF exists (select C.CPF
                    from POLITEC.CRIME C
                    where C.CPF=P.CPF
                    and C.TIPO = 'HEDIONDO');
```

Comentário:

O comando SQL apresentado tem um erro de sintaxe. A cláusula **exists** não está sendo utilizada corretamente. A forma correta de escrever esse comando SQL para selecionar o CPF e o nome de todas as pessoas que cometeram pelo menos um crime do tipo hediondo é:

```
SELECT P.CPF, P.NOME
FROM POLITEC.PESSOA P
WHERE EXISTS (SELECT C.CPF
FROM POLITEC.CRIME C
WHERE C.CPF = P.CPF
AND C.TIPO = 'HEDIONDO');
```

O comando correto utiliza a cláusula **EXISTS** e a subconsulta entre parênteses para verificar a existência de registros na tabela **POLITEC.CRIME** que atendem aos critérios especificados.

Portanto, o item está incorreto devido ao erro na sintaxe do comando SQL apresentado.

Gabarito: Errado

18. CEBRASPE (CESPE) - Ana Reg (AGER MT)/AGER MT/Ciências da Computação e Sistemas de Informação/2023

Assinale a opção em que é apresentada a instrução utilizada para alterar alguma propriedade de campo de uma tabela em um banco de dados.

a) INSERT

b) UPDATE

c) ALTER

d) DROP

e) TRUNCATE

Comentário:

A instrução utilizada para alterar alguma propriedade de campo de uma tabela em um banco de dados é: **c) ALTER**. Portanto, a opção correta é a alternativa C.

Gabarito: C

19. CEBRASPE (CESPE) - Ana Reg (AGER MT)/AGER MT/Ciências da Computação e Sistemas de Informação/2023

Em linguagem de manipulação de dados DML, o operador SQL BETWEEN serve para

a) delimitar o valor de uma coluna na cláusula WHERE.

b) delimitar as colunas a serem apresentadas na cláusula WHERE.

c) delimitar os limites de um campo para a cláusula INSERT.

d) restringir a quantidade de linhas a serem recuperadas na cláusula UPDATE.

e) restringir a quantidade de campos na cláusula DELETE.

Comentário:

Em linguagem de manipulação de dados (DML), o operador SQL BETWEEN serve para **delimitar o valor de uma coluna na cláusula WHERE.** Portanto, a opção correta é: **a) delimitar o valor de uma coluna na cláusula WHERE.**

Gabarito: A

20. CEBRASPE (CESPE) - Ana (SERPRO)/SERPRO/Tecnologia/2023

Considerando que existem diferentes tipos de banco de dados, como os bancos de dados relacionais e os não relacionais (ou NoSQL), julgue o item a seguir.

Considere-se que, em um banco de dados, constem duas tabelas — clientes e pedidos — e que na primeira tabela haja informações dos clientes, como ID, nome e endereço, e na segunda tabela, informações dos pedidos realizados pelos clientes, como ID do pedido, data e valor total. Nessa situação hipotética, para retornar ao nome e ao endereço de clientes que já realizaram pedidos, é correto usar o comando SELECT em conjunto com a cláusula LEFT JOIN.

Comentário:

Comentário: Se você inverter a ordem das tabelas e usar **LEFT JOIN**, poderá obter os resultados desejados, retornando o nome e o endereço de todos os clientes, mesmo daqueles que não realizaram pedidos. A consulta SQL seria assim:

```
SELECT clientes.nome, clientes.endereco
FROM pedidos LEFT JOIN clientes
```

```
ON pedidos.ID_cliente = clientes.ID;
```

Neste caso, você obterá uma lista de todos os clientes, incluindo aqueles que não realizaram pedidos. Se um cliente não tiver pedidos correspondentes, as colunas **nome** e **endereco** na saída terão valores nulos.

Logo, temos uma alternativa correta. Entretanto a banca anulou a questão.

Gabarito: Anulada

21. CEBRASPE (CESPE) - AIS (EMPREL)/EMPREL/Banco de Dados/2023

Na linguagem SQL, a função que é usada para verificar se o resultado de uma subconsulta correlacionada é vazio ou não, a qual retorna como resultado o valor booleano TRUE ou FALSE, é

a) HAVING.

b) EXISTS.

c) LIKE.

d) IN.

e) AVG.

Comentário:

A função na linguagem SQL usada para verificar se o resultado de uma subconsulta correlacionada é vazio ou não, e que retorna como resultado o valor booleano TRUE ou FALSE, é a função **EXISTS**. Portanto, a opção correta é: b) EXISTS.

Gabarito: B

22. CEBRASPE (CESPE) - AFM (Pref Fortaleza)/Pref Fortaleza/Ciência da Computação, Informática, Processamento de Dados/2023

Julgue o item a seguir, a respeito de arquitetura de dados, metadados e linguagens de bancos de dados.

Usando-se SQL (*structured query language*), é possível unir o resultado de duas instruções SELECT quaisquer, por meio do operador UNION.

Comentário:

O item está incorreto. Para utilizar o operador UNION em SQL, é fundamental que as duas instruções SELECT envolvidas tenham o mesmo número de colunas, e as colunas correspondentes devem ter tipos de dados compatíveis. Isso é necessário para garantir a integridade e a consistência dos resultados. Portanto, a afirmação inicial no item está equivocada, e a utilização do operador UNION depende da compatibilidade na estrutura das instruções SELECT.

Gabarito: Errado

23. CEBRASPE (CESPE) - AFM (Pref Fortaleza)/Pref Fortaleza/Ciência da Computação, Informática, Processamento de Dados/2023

Julgue o item a seguir, a respeito de arquitetura de dados, metadados e linguagens de bancos de dados.

Comandos expressos em linguagem de definição de dados (DDL) são utilizados para criar estruturas de um banco de dados, e o seu processamento irá incluir ou alterar metadados desse mesmo banco de dados.

Comentário:

O item está correto. Comandos expressos em Linguagem de Definição de Dados (DDL) são utilizados para definir e modificar a estrutura do banco de dados, incluindo a criação, alteração e exclusão de tabelas, índices, visões, procedimentos armazenados, entre outros elementos. O processamento desses comandos não apenas afeta os dados armazenados, mas também atualiza ou cria metadados associados ao banco de dados.

Metadados são informações sobre os dados, e no contexto de um banco de dados, eles descrevem a estrutura, restrições, relacionamentos e outras propriedades dos objetos no banco de dados. Quando você executa comandos DDL, está essencialmente definindo ou alterando metadados.

Portanto, a criação e modificação de estruturas do banco de dados utilizando comandos DDL têm um impacto direto nos metadados associados a essas estruturas.

Gabarito: Certo

24. CEBRASPE (CESPE) - AFM (Pref Fortaleza)/Pref Fortaleza/Ciência da Computação, Informática, Processamento de Dados/2023

Julgue o item a seguir, a respeito de arquitetura de dados, metadados e linguagens de bancos de dados.

A manipulação de dados armazenados em um banco de dados é feita por meio de DML (*Data Manipulation Language*), linguagem na qual é possível apenas descrever os dados que se deseja acessar, sem especificar como obtê-los.

Comentário:

O item está incorreto. O SQL (Structured Query Language) é uma linguagem de consulta declarativa, o que significa que os usuários especificam o que desejam que seja feito, e não como fazer. Vamos analisar a questão por partes ...

"A manipulação de dados armazenados em um banco de dados é feita por meio de DML (Data Manipulation Language)"

Até aqui, o texto está ok! Ele descreve uma função genérica da sublinguagem DML do SQL. A DML no contexto do SQL é usada para realizar operações de consulta e manipulação de dados, como SELECT, INSERT, UPDATE e DELETE. Embora os usuários forneçam instruções sobre as operações desejadas, eles não precisam especificar como o banco de dados deve executar essas operações. Em vez disso, o banco de dados determina o plano de execução mais eficiente para atender à consulta ou à manipulação de dados.

"linguagem na qual é possível apenas descrever os dados que se deseja acessar, sem especificar como obtê-los."

A Linguagem de Manipulação de Dados (DML - Data Manipulation Language) é uma linguagem que permite especificar não apenas os dados que se deseja acessar, mas também dados que precisamos atualizar, inserir ou excluir. O erro da questão está em dizer que estamos dizendo que DML contém apenas comandos de SELECT, sem considerar os demais.

Gabarito: Errado

25. CEBRASPE (CESPE) - AFM (Pref Fortaleza)/Pref Fortaleza/Geografia/2023

No que diz respeito a banco de dados relacional e banco de dados geográfico, julgue o item a seguir.

A SQL (*structured query language*) pode ser subdividida em duas sublinguagens: a linguagem de definição de dados (DDL), que fornece comandos visando definir e modificar esquemas de tabelas, remover tabelas, criar índices e definir restrições de integridade; e a linguagem de manipulação de dados (DML), formada por comandos para consulta, inserção, modificação e remoção de dados no banco de dados.

Comentário:

O item está correto. A SQL (Structured Query Language) é tradicionalmente subdividida em duas sublinguagens principais:

Linguagem de Definição de Dados (DDL): Esta sublinguagem inclui comandos que são usados para definir e modificar a estrutura dos objetos no banco de dados. Exemplos de comandos DDL incluem CREATE (criar), ALTER (alterar) e DROP (eliminar). Comandos DDL são usados para criar ou modificar esquemas de tabelas, índices, restrições de integridade, entre outros.

Linguagem de Manipulação de Dados (DML): Esta sublinguagem envolve comandos que lidam com os dados propriamente ditos. Exemplos de comandos DML incluem SELECT (consulta), INSERT (inserção), UPDATE (atualização) e DELETE (exclusão). Comandos DML são usados para recuperar, inserir, modificar e excluir dados no banco de dados.

A divisão entre DDL e DML permite uma clara distinção entre a definição da estrutura do banco de dados e as operações realizadas sobre os dados armazenados nele.

Gabarito: Certo

26. CEBRASPE (CESPE) - Ana TI (FUB)/FUB/2023

Acerca de bancos de dados relacionais, julgue o item seguinte.

Triggers e *stored procedures* são componentes do modelo físico, enquanto relacionamentos e *primary keys* são componentes do modelo lógico.

Comentário:

O item está correto. Vamos aproveitar e esclarecer a distinção entre triggers e stored procedures em relação ao modelo físico e lógico de um banco de dados relacional:

- Triggers (gatilhos): Triggers são acionados automaticamente por eventos, como a execução de uma instrução INSERT, UPDATE ou DELETE. Triggers são usados para impor regras de negócios ou aplicar ações específicas quando certos eventos ocorrem nas tabelas. Portanto, é mais apropriado associá-los ao modelo físico.

- Stored Procedures (procedimentos armazenados): Stored procedures são coleções de instruções SQL que podem ser executadas como uma única unidade lógica. Stored procedures encapsulam lógica de negócios e são independentes da estrutura física do banco de dados. A definição de stored procedures é armazenada no banco de dados, e elas são tratadas como parte do modelo físico.

- Relacionamentos e Primary Keys: Relacionamentos e chaves primárias (primary keys) estão associados ao modelo lógico de um banco de dados relacional. Relacionamentos definem como as tabelas estão conectadas, enquanto as chaves primárias garantem a unicidade e integridade referencial nas tabelas. Esses conceitos estão mais relacionados à estrutura lógica e ao design das tabelas do que à implementação física.

Portanto, para uma visão mais precisa, triggers e stored procedures são frequentemente considerados como parte do modelo físico, enquanto relacionamentos e primary keys são mais associados ao modelo lógico.

Gabarito: Certo

27. CEBRASPE (CESPE) - Ana (MPE RO)/MPE RO/Programador/2023

Assinale a opção que apresenta a instrução em SQL que permite substituir por Pedro o nome do advogado atual da parte autora (nome_adv_autor) do processo de número 2023001, em uma tabela de nome processo.

a) UPDATE processo

SET nome_adv_autor = 'Pedro'

WHERE numero_processo = 2023001;

b) CHANGE processo

SET nome_adv_autor = 'Pedro'

WHERE numero_processo = 2023001;

c) ALTER TABLE processo

SET nome_adv_autor = 'Pedro'

WHERE numero_processo = 2023001;

d) UPDATE processo

SET nome_adv_autor := 'Pedro'

WHERE numero_processo = 2023001;

e) UPDATE processo

SET nome_adv_autor LIKE 'Pedro'

WHERE numero_processo = 2023001;

Comentário:

A instrução SQL correta para substituir o nome do advogado da parte autora na tabela de processo é:

```
UPDATE processo
SET nome_adv_autor = 'Pedro'
WHERE numero_processo = 2023001;
```

Portanto, a opção correta é a alternativa a). Vamos analisar a instrução:

- UPDATE processo: Indica que a operação a ser realizada é uma atualização na tabela chamada processo.
- SET nome_adv_autor = 'Pedro': Especifica a coluna a ser modificada (nome_adv_autor) e o valor que será atribuído a essa coluna após a atualização ('Pedro').
- WHERE numero_processo = 2023001: Define a condição para aplicar a atualização apenas às linhas que atendem a essa condição. Neste caso, o UPDATE será aplicado apenas às linhas onde o valor da coluna numero_processo for igual a 2023001.

Portanto, o significado geral desta instrução SQL é: "Atualize o valor da coluna nome_adv_autor para 'Pedro' na tabela processo apenas nas linhas onde o numero_processo é igual a 2023001."

Gabarito: A

28. Cebraspe – Técnico (Analista de Sistemas) Infraestrutura e Segurança da Informação (BNB)/2022

O cursor armazena, em memória, uma única linha do comando de SELECT de uma consulta que tenha sido efetuada em tabelas do banco de dados.

Comentário:

A afirmativa está incorreta. Um cursor não armazena em memória uma única linha do comando SELECT de uma consulta. Na verdade, um cursor é uma estrutura de controle que permite aos programas ou consultas percorrerem um conjunto de resultados linha por linha, geralmente em uma consulta SELECT.

Quando um cursor é usado, o DBMS mantém um conjunto de resultados em um estado chamado "aberto" e, à medida que o cursor é percorrido, o DBMS recupera linhas sob demanda, mas normalmente não armazena todas as linhas em memória ao mesmo tempo. Portanto, um cursor não armazena uma única linha, mas sim fornece acesso sequencial a todas as linhas do conjunto de resultados, uma por vez.

Gabarito: Errado

29. Cebraspe – Técnico (Analista de Sistemas) Infraestrutura e Segurança da Informação (BNB)/2022

Quando acionado, um trigger pode ser executado em substituição ao comando que o disparou.

Comentário:

A afirmação está correta. Um trigger é um objeto em um banco de dados que é acionado automaticamente em resposta a certos eventos, como a inserção, atualização ou exclusão de dados em uma tabela. Um trigger é definido para executar uma ação específica quando ocorre o evento que o aciona.

No contexto da afirmação, é possível criar um trigger que substitua ou modifique o comportamento do comando que o disparou. Isso significa que, quando o evento associado ao trigger ocorre (por exemplo, uma inserção em uma tabela), o trigger pode executar um conjunto de instruções personalizadas em vez do comando original que acionou o evento.

Por exemplo, se um trigger é definido para disparar antes de uma inserção em uma tabela, ele pode modificar os valores a serem inseridos ou até mesmo cancelar a inserção, se certas condições não forem atendidas. Portanto, um trigger pode, de fato, ser executado em substituição ao comando que o disparou, dependendo da lógica definida no próprio trigger.

Gabarito: Certo

30. Cebraspe – Técnico (Analista de Sistemas) Infraestrutura e Segurança da Informação (BNB)/2022

A linguagem de controle de dados (DCL) é um dos componentes da SQL e permite gerenciar as autorizações aos dados constantes do banco de dados.

Comentário:

A afirmação está correta. A Linguagem de Controle de Dados (Data Control Language - DCL) é de fato um dos componentes da SQL (Structured Query Language) e é usada para gerenciar as autorizações e permissões de acesso aos dados em um banco de dados.

A DCL inclui comandos como GRANT e REVOKE, que são usados para conceder e revogar privilégios e permissões de acesso a usuários, grupos de usuários ou papéis no banco de dados. Esses comandos permitem controlar quem pode realizar operações específicas em tabelas, visões, procedimentos armazenados e outros objetos do banco de dados.

Portanto, a DCL é uma parte importante da SQL que se concentra no gerenciamento da segurança e das autorizações dos dados no banco de dados, garantindo que apenas usuários autorizados possam acessar e manipular os dados.

Gabarito: Certo

31. Cebraspe – Técnico – Analista de Sistemas – Desenvolvimento de Sistemas (BNB)/2022

Julgue os itens que se seguem, acerca dos conceitos de linguagem de consulta estruturada (SQL).

Considere-se a tabela e o script SQL a seguir.

Tabela: colaboradores

id	sexo	idade
1	Masculino	32
2	Feminino	0
3	Feminino	30

```
SELECT avg(idade), sexo
FROM colaboradores
GROUP BY sexo
```

O resultado da consulta SQL é

```
avg(idade)      sexo
15              Feminino
32              Masculino
```

Comentário:

A consulta SQL fornecida calcula a média das idades dos colaboradores para cada valor único na coluna "sexo" na tabela "colaboradores". Aqui está o que a consulta faz em detalhes:

- SELECT AVG(idade): Isso calcula a média (AVG) dos valores na coluna "idade" da tabela "colaboradores". Ou seja, para cada grupo de registros com o mesmo valor na coluna "sexo", a consulta calcula a média das idades desses registros.
- , sexo: Isso indica que a coluna "sexo" também será incluída no resultado da consulta.
- FROM colaboradores: Isso especifica a tabela "colaboradores" da qual os dados serão selecionados.
- GROUP BY sexo: Isso agrupa os resultados pela coluna "sexo". A média das idades será calculada para cada valor único encontrado na coluna "sexo". Portanto, se houver registros com "sexo" igual a "Masculino" e "Feminino", a consulta retornará a média das idades para ambos os grupos separadamente.

O resultado da consulta será uma lista que inclui a média das idades para cada valor único encontrado na coluna "sexo" na tabela "colaboradores". Logo, temos uma afirmação correta.

Gabarito: Certo

32. Cebraspe – Técnico – Analista de Sistemas – Desenvolvimento de Sistemas (BNB)/2022

A linguagem de manipulação de dados (DML) inclui instruções que modificam a estrutura de um banco de dados.

Comentário:

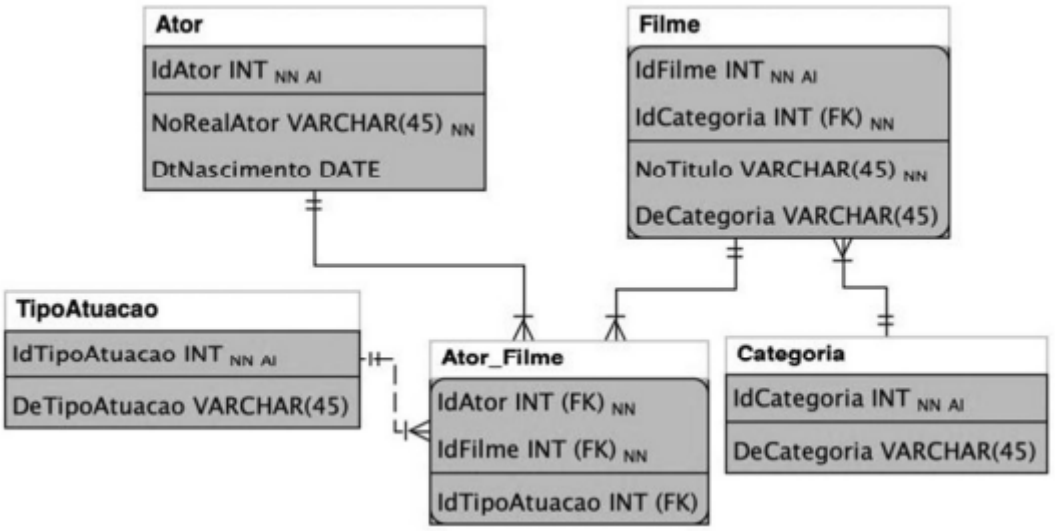
A afirmação está incorreta. A Linguagem de Manipulação de Dados (Data Manipulation Language - DML) não inclui instruções que modificam a estrutura de um banco de dados. A DML é responsável por operações que manipulam os dados dentro do banco de dados, como inserção, atualização, exclusão e recuperação de dados.

As instruções que modificam a estrutura de um banco de dados pertencem à Linguagem de Definição de Dados (Data Definition Language - DDL). A DDL inclui comandos para criar, alterar e excluir objetos de banco de dados, como tabelas, visões, índices e outras estruturas relacionadas.

Portanto, é importante distinguir entre DML e DDL, uma vez que são componentes diferentes da linguagem SQL que atendem a finalidades distintas: manipulação de dados e definição de estruturas.

Gabarito: Errado

33. Cebraspe – Analista Judiciário – Tecnologia da Informação (TRT-AP/PA)/2022



Considere o modelo precedente, em que os campos IdFilme, IdAtor, IdTipoAtuacao e IdCategoria são chaves primárias em suas respectivas tabelas, como referência.

Considere ainda que

FK descreve que o campo é uma foreign key;

AI descreve que o campo é auto incremente;

NN descreve que o campo é not null.

A partir dessas informações, assinale a opção que apresenta o correto script SQL para criação desta tabela.

a)
```
CREATE TABLE Categoria (
  IdCategoria INT NOT NULL AUTO_INCREMENT,
  DeCategoria VARCHAR(45) NULL,
  PRIMARY KEY (IdCategoria),
    FOREIGN KEY (IdCategoria)
    REFERENCES Filme (IdCategoria)
) ENGINE = InnoDB;
```

b)
```
CREATE TABLE Filme (
  IdFilme INT NOT NULL AUTO_INCREMENT,
  NoTitulo VARCHAR(45) NOT NULL,
  DeCategoria VARCHAR(45) NULL,
  IdCategoria INT NOT NULL,
  PRIMARY KEY (IdFilme, IdCategoria)
) ENGINE = InnoDB;
```

c)
```
CREATE TABLE Ator (
  IdAtor INT NOT NULL PRIMARY KEY
AUTO_INCREMENT,
  NoRealAtor VARCHAR(45) NOT NULL,
  DtNascimento DATE NULL,
    FOREIGN KEY (IdAtor)
    REFERENCES Ator_Filme (IdAtor)
) ENGINE = InnoDB;
```

d)
```
CREATE TABLE Ator_Filme (
  IdAtor INT NOT NULL,
  IdFilme INT NOT NULL,
  IdTipoAtuacao INT NULL,
  PRIMARY KEY (IdAtor, IdFilme),
    FOREIGN KEY (IdAtor)
    REFERENCES Ator (IdAtor),
    FOREIGN KEY (IdFilme)
    REFERENCES Filme (IdFilme),
    FOREIGN KEY (IdTipoAtuacao)
    REFERENCES TipoAtuacao (IdTipoAtuacao)
) ENGINE = InnoDB;
```

e)

```
CREATE TABLE TipoAtuacao (
  IdTipoAtuacao  INT  NOT  NULL  AUTO_INCREMENT
PRIMARY KEY,
  DeTipoAtuacao VARCHAR(45) NULL,
    FOREIGN KEY (IdTipoAtuacao)
    REFERENCES Ator_Filme (IdTipoAtuacao)
) ENGINE = InnoDB;
```

Comentário:

Com base nas informações fornecidas, podemos identificar as tabelas que contêm chaves estrangeiras (FK): Filme e Ator_Filme. Isso significa que as letras A, C e E podem ser eliminadas como opções corretas.

Ao analisar a opção B, notamos que ela cria a tabela Filme e define as colunas IdFilme e IdCategoria. No entanto, parece que houve um erro na criação da chave estrangeira, pois falta a cláusula "FOREIGN KEY (IdCategoria) REFERENCES Categoria (IdCategoria)" para estabelecer a relação com a tabela Categoria. Portanto, essa opção também está incorreta.

Assim, nos resta a alternativa D como resposta para a questão.

Gabarito: Letra D

34. Cebraspe – Quality Assurance (QA) e Analista de Teste (BANRISUL)/2022

Na linguagem SQL, os comandos GRANT e REVOKE são exemplos do subconjunto DDL (linguagem de definição de dados), pois definem o acesso aos dados do banco.

Comentário:

Os comandos GRANT e REVOKE são exemplos de comandos SQL que fazem parte do subconjunto DCL (Data Control Language), e não do DDL (Data Definition Language).

O DDL é responsável por definir a estrutura dos dados no banco de dados, como criar, alterar ou excluir tabelas, índices, visões (views) e outros objetos de esquema. Exemplos de comandos DDL incluem CREATE, ALTER e DROP.

Por outro lado, o DCL é responsável por controlar o acesso aos dados no banco de dados, ou seja, gerenciar as permissões de acesso dos usuários e definir a segurança. Os comandos GRANT e REVOKE são utilizados no DCL para conceder ou revogar privilégios e permissões de acesso a usuários e papéis no banco de dados.

Portanto, a parte correta da afirmação é que os comandos GRANT e REVOKE definem o acesso aos dados do banco de dados, mas a parte incorreta é que eles fazem parte do subconjunto DDL. Na verdade, esses comandos são parte do DCL, portanto temos uma alternativa **incorreta**.

Gabarito: Errado

35. Cebraspe – Quality Assurance (QA) e Analista de Teste (BANRISUL)/2022

No que se refere à álgebra relacional e a SQL, julgue os itens a seguir.

Considerando-se uma tabela nomeada empregados que contém, entre outras colunas, uma identificada como nome, o comando SQL que retorna o nome de todos os empregados que tenham o nome luiz ou luis é o seguinte. SELECT nome FROM empregados WHERE nome like '%lui_%'

Comentário:

O comando SQL apresentado está incorreto ou no mínimo impreciso para a finalidade descrita.

Para retornar o nome de todos os empregados que tenham o nome "luiz" ou "luis", o comando correto deve utilizar o operador OR e o uso correto do padrão "LIKE" para permitir diferentes possibilidades de ortografia. O comando correto seria algo como:

```
SELECT nome FROM empregados WHERE nome LIKE 'luiz%' OR nome LIKE 'luis%';
```

O operador OR é usado para combinar as duas condições (nome começa com "luiz" ou nome começa com "luis") e o uso de % permite que qualquer sequência de caracteres possa vir após "luiz" ou "luis". Se você usar o % da forma como foi descrita na questão, nomes como luimar, lui, etc seriam considerados consistentes com o predicado e apareceriam no resultado

Portanto, o comando apresentado inicialmente contém um erro na construção da consulta SQL para atingir o resultado desejado. Entretanto, a banca deu o gabarito como certo.

Gabarito: Certo/Gabarito do professor: Errado

36. Cebraspe – Analista de Contas Públicas – Especialidade: Administração (MPC-SC)/2022

Acerca do conceito de view, da modelagem dimensional, do modelo de referência CRISP-DM e da linguagem SQL, julgue os itens subsequentes.

Em banco de dados relacionais, as views são conhecidas como tabelas físicas, uma vez que elas armazenam apenas as informações de interesse do usuário, que podem ser coletadas de uma ou mais tabelas do banco.

Comentário:

Essa afirmação não está correta. Em banco de dados relacionais, as views não são conhecidas como tabelas físicas. Views são estruturas virtuais que permitem aos usuários acessar e consultar dados de uma ou mais tabelas, mas elas não armazenam dados físicos.

Aqui estão algumas diferenças entre tabelas e views:

Armazenamento de Dados: Tabelas armazenam dados físicos, enquanto as views não armazenam dados. As views são consultas predefinidas que retornam dados com base em uma ou mais tabelas subjacentes.

Estrutura: Tabelas têm uma estrutura física definida, incluindo colunas e tipos de dados. As views têm uma estrutura lógica definida pela consulta SQL que as cria.

Atualizações: Geralmente, é possível inserir, atualizar e excluir dados diretamente de tabelas. Em contraste, as views são frequentemente usadas para consultas somente leitura, embora algumas views permitam atualizações, desde que sejam configuradas para isso.

Uso: As views são usadas para simplificar consultas complexas, ocultar detalhes de implementação, fornecer uma camada de segurança e criar uma representação lógica dos dados. Elas não armazenam dados por si mesmas.

Portanto, views não são tabelas físicas; são consultas definidas que facilitam o acesso e a consulta de dados de uma maneira mais conveniente e eficiente.

Gabarito: Errado

37. Cebraspe – Analista de Contas Públicas – Especialidade: Administração (MPC-SC)/2022

O comando a seguir tem a finalidade de mostrar o nome e o email de todos os procuradores que recebem salários acima de R$ 30.000,00.

```
SELECT nome, email
FROM procurador
WHEN salario >= 30.000,00;
```

Comentário:

A afirmação está incorreta. A cláusula correta para filtrar registros em SQL é WHERE, não WHEN. Portanto, o comando SQL corrigido para mostrar o nome e o email de todos os procuradores que recebem salários acima de R$ 30.000,00 é o seguinte:

```
SELECT nome, email FROM procurador WHERE salario > 30000;
```

Neste comando SQL, estamos selecionando as colunas nome e email da tabela procurador onde a coluna salario é maior do que R$ 30.000,00. Isso retornará os nomes e emails dos procuradores que atendem a esse critério de salário.

Logo, temos uma afirmação incorreta.

Gabarito: Errado

38. Cebraspe – Analista de Contas Públicas – Especialidade: Administração (MPC-SC)/2022

O CREATE é o principal comando da linguagem SQL, pois permite criar diversos objetos no banco de dados, como, por exemplo, tabelas e domínios. Entretanto, esse comando não é utilizado para criar views.

Comentário:

A afirmação está incorreta. O comando CREATE na linguagem SQL é, de fato, utilizado para criar uma variedade de objetos no banco de dados, incluindo tabelas, domínios, índices, procedimentos armazenados, funções e também views.

Portanto, o comando CREATE é usado para criar views em SQL. As views são objetos que representam consultas SQL armazenadas que podem ser tratadas como tabelas virtuais. Elas permitem aos usuários ou aplicativos consultar dados de maneira mais conveniente, mascarando a complexidade subjacente das tabelas reais ou combinando dados de várias tabelas em uma única estrutura.

Aqui está um exemplo simples de criação de uma view:

```
CREATE VIEW NomeDaView AS SELECT coluna1, coluna2 FROM tabela WHERE condição;
```

Neste exemplo, uma view chamada "NomeDaView" está sendo criada com base na consulta SQL especificada. Portanto, o comando CREATE é usado para criar essa view.

Gabarito: Errado

39. Ano: 2018 Banca: CESPE Órgão: STJ Cargo: Técnico Judiciário – Suporte Técnico

A respeito de sistemas gerenciadores de banco de dados (SGBD), julgue os próximos itens.

79 O comando GRANT é utilizado para conceder privilégios em um objeto do SGBD, ao passo que o comando REVOKE serve para cancelar um privilégio já concedido.

80 A linguagem de manipulação de dados (DML – data definition language) é usada para, entre outras finalidades, criar e alterar estruturas de tabelas em um SGBD.

81 Em um SGBD, o trigger pode substituir a instrução que o originou, sendo a instrução original descartada e apenas o trigger executado.

Comentário: Essa questão apresenta três afirmações que tratam do assunto de SQL. Vejamos cada uma delas de forma separada.

Na alternativa 79 temos uma descrição dos comandos de controle de dados GRANT e REVOKE. O primeiro serve para conceder privilégios, já o segundo é utilizado para remoção de permissões de usuários ou perfis. Sendo assim, podemos afirmar que a alternativa 79 está correta.

Já a alternativa 81 faz afirmações totalmente desconexas a respeito da linguagem de manipulação de dados. Primeiro tenta mudar a descrição do termo em inglês, lembre-se: DML – *data manipulation language*. Depois fala que a DML é usada para criação e alteração de estruturas, o que também não é verdade. Desta forma, podemos marcar nossa afirmativa como errada.

Por fim, a afirmação 82 trada dos gatilhos ou triggers. O trigger pode ser descrito de três formas distintas em relação ao momento que o código deve ser executado: BEFORE, AFTER e INSTEAD OF. Esse último permite que a instrução seja executada em substituição a instrução original. Desta forma, a alternativa 82 está correta.

Gabarito: C E C

40. Ano: 2018 Banca: CESPE Órgão: STJ Cargo: Técnico Judiciário – Suporte Técnico

Acerca de banco de dados, julgue os itens que se seguem.

76 A diferença entre materialized view e view comum em um banco de dados é o fato de que a primeira é armazenada em cache como uma tabela física, enquanto a segunda existe apenas virtualmente.

Comentário: Uma visão (*view*) é uma consulta armazenada no banco de dados. Nós podemos realizar consultas sobre uma *VIEW* como se fosse uma tabela. Muitas pessoas se referem às VIEWs como uma tabela virtual. Uma das principais funções da VIEW é controlar a segurança do banco de dados. Geralmente se cria a VIEW (uma consulta armazenada no banco de dados) com os campos que determinado perfil de usuário pode acessar, e concede-se ao usuário acesso apenas a essa VIEW e não à(s) tabela(s) diretamente.

Também utiliza-se VIEWs para apresentar informações mais organizadas para o usuário sem que ele precise elaborar uma consulta complexa. Esta já estaria pronta e armazenada no próprio banco de dados para uso.

Já a Visão Materializada é uma tabela simples que é derivada de outras tabelas e existe necessariamente em sua forma física, ou seja, não é uma tabela virtual. Podemos, portanto, assinalar a alternativa como correta.

Gabarito: C

41. Ano: 2018 Banca: CESPE Órgão: STJ Cargo: Técnico Judiciário – Desenvolvimento de Sistemas

Julgue os seguintes itens, relativos a métricas de qualidade desoftware, JUnit, SQL, Delphi e desenvolvimento mobile.

107 A sentença SQL seguinte produzirá como resultado a lista de todos os funcionários de uma empresa. Para aqueles em que seja verdadeira a condição Funcionarios.CodigoDep = Departamentos.CodigoDep, será apresentado também o nome do departamento.

```
SELECT Funcionarios.Nome, Departamentos.NomeDep
FROM Funcionarios
INNER JOIN Departamentos ON Funcionarios.CodigoDep
= Departamentos.CodigoDep
ORDER BY
Funcionarios.Nome;
```

Comentário: Veja que a junção utilizada na questão é um INNER JOIN, logo os funcionários que não tiverem um departamento associado não aparecerão no resultado final da consulta. Desta forma, o gabarito da mesma é errado.

Gabarito: E

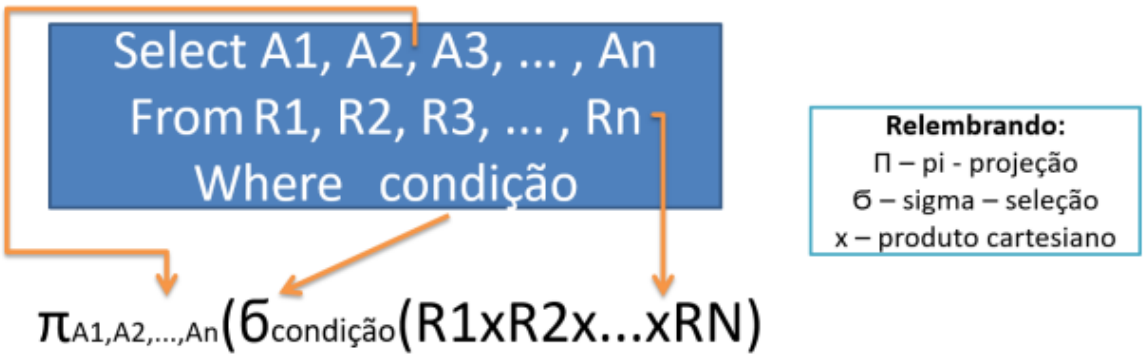
42. Ano: 2018 Banca: CESPE Órgão: ABIN Cargo: Área 08 Questão: 144

A respeito de sistemas gerenciadores de banco de dados, julgue os próximos itens.

144 O comando SQL select campo from tabela corresponde a uma operação de projeção da álgebra relacional.

Comentário:

Sabemos que quando estamos relacionando as cláusulas do comando SELECT de SQL à álgebra relacional temos a seguinte associação:



Assim, podemos observar que a afirmação está correta.

Gabarito: C

43. Ano: 2018 Banca: CESPE Órgão: ABIN Cargo: Área 09 Questões: 144 a 147

```
SELECT nome FROM funcionario
WHERE area = 'INTELIGENCIA'
AND endereco LIKE '%BRASILIA,DF%';
```

Tendo como referência o código SQL precedente, julgue os itens a seguir.

144 Na cláusula WHERE, a condição de seleção area = 'INTELIGENCIA' escolhe a tupla de interesse em particular na tabela funcionario, pois area é um atributo de funcionario.

145 O código em apreço realiza uma consulta que mostra o nome dos funcionários da área de INTELIGENCIA e que têm, como parte do endereço, a cidade de BRASILIA,DF.

146 A palavra INTELIGENCIA está entre aspas simples por pertencer a um atributo, area, o qual tem o tipo de dados definido como caractere.

147 Em LIKE '%BRASILIA,DF%', o recurso LIKE foi definido de forma incorreta, uma vez que a utilização da vírgula (,), sem a inclusão da palavra-chave ESCAPE, impedirá que o código seja executado.

Comentário:

Vamos comentar cada uma das alternativas acima.

144. Vejam que a cláusula WHERE vai restringir as tuplas nas quais o valor do atributo área seja igual a 'INTELIGENCIA'. Desta forma, a alternativa está correta.

145. Exatamente, a afirmação está perfeita. As restrições ou predicados do código SQL são exatamente as descritas na afirmação. Desta forma, a alternativa está correta.

146. Confesso que quando olhei para essa questão pensei que ela estivesse incorreta, mas acho que foi excesso de preciosismo meu. Veja que a palavra INTELIGENCIA é de fato uma constante do tipo caractere que vai servir como referência para a busca na coluna área. Para cada funcionário cujo valor do atributo área seja igual a INTELIGENCIA temos uma tupla no retorno da consulta. Mesmo assim, o CESPE deu o gabarito como correto.

147. Quando usamos as aspas simples, as palavras reservadas ou caracteres especiais que aparecem entre as duas aspas não precisam de caracteres ou palavras chave de ESCAPE. Desta forma a alternativa está incorreta.

Gabarito: C C C E

44. BANCA: CESPE ANO: 2015 ÓRGÃO: TJDFT CARGO: PROGRAMAÇÃO DE SISTEMAS – QUESTÕES 63 E 64

Acerca de linguagens de definição e manipulação de dados, julgue os itens subsecutivos.

Apelido ou column alias não pode ser utilizado na cláusula WHERE.

Em uma coluna definida como NUMBER (7,2), o valor 34567.2255 será armazenado como 34567.23.

Comentários:

Vamos comentar cada uma das alternativas acima.

A primeira, refere-se ao alias, que pode ser utilizado implicitamente ou representado pela palavra-chave AS. Pode ser usado sobre colunas na cláusula SELECT ou em tabelas na cláusula FROM. Contudo, não deve ser usado na cláusula WHERE, geralmente referenciamos o alias definido anteriormente.

A segunda alternativa trata da sintaxe de definição da variável NUMBER, o primeiro número refere-se à quantidade total de algarismos do número, já o segundo valor entre parênteses diz respeito a quantidade de casas decimais. Observem que a questão se encontra correta.

Gabarito: C C

45. BANCA: CESPE ANO: 2015 ÓRGÃO: TJDFT CARGO: PROGRAMAÇÃO DE SISTEMAS – QUESTÕES 65 A 66

Julgue os próximos itens, relativos a SQL.

O comando SQL ilustrado a seguir atualiza os dados dos empregados do departamento (id_departamento) 50 que têm como função (id_funcao) VENDEDOR para o departamento 80 e gerente (id_gerente) 145.

UPDATE empregados

SET id_departamento = 80, id_gerente = 145

WHERE id_departamento = 50 AND funcao = 'VENDEDOR';

O comando SQL mostrado a seguir fará uma consulta na tabela empregados e retornará os campos primeiro_nome, sobrenome e salario de todos os empregados do departamento (id_departamento) 40, ordenados pelo campo sobrenome.

SELECT primeiro_nome, sobrenome, salario FROM empregados

WHERE id_departamento = 40 ORDER BY sobrenome

Comentários:

Vamos comentar as alternativas acima.

Na alternativa 65 o CESPE cometeu um erro proposital na digitação do atributo função (id_funcao), percebam que não temos o id na descrição do comando. A ideia é a seguinte. Pense que a tabela empregados terá um identificador para departamento e função. Esses podem ter suas descrições e outros atributos descritos em suas tabelas. Desta forma, na tabela empregados teríamos apenas o identificador. Alternativa errada.

A questão 66 está perfeitamente correta! Ela apresenta a construção de um comando select com a sintaxe e as cláusulas descritas na ordem certa.

Gabarito: E C

46. BANCA: CESPE ANO: 2016 ÓRGÃO: TCE-SC CARGO: AUDITOR DE TI

Com relação aos bancos de dados relacionais, julgue os próximos itens.

94 O catálogo de um sistema de gerenciamento de banco de dados relacional armazena a descrição da estrutura do banco de dados e contém informações a respeito de cada arquivo, do tipo e formato de armazenamento de cada item de dado e das restrições relativas aos dados.

95 Denomina-se visão uma tabela única derivada de uma ou mais tabelas básicas do banco. Essa tabela existe em forma física e viabiliza operações ilimitadas de atualização e consulta.

96 Em bancos de dados relacionais, as tabelas que compartilham um elemento de dado em comum podem ser combinadas para apresentar dados solicitados pelos usuários.

Comentário:

O dicionário de dados ou catálogo de dados contém as descrições das estruturas dos objetos presentes na base de dados. Presente em todos os SGBDs relacionais ele guarda os metadados ou informações a respeito dos objetos armazenados. Podemos marcar como correta a assertiva 94.

A definição de visão presente no padrão SQL/ANSI é de uma estrutura temporária que armazena informações advindas de uma ou mais tabelas. A visão não é armazenada fisicamente em disco e é removida ou apagada ao final da sua utilização. Sendo assim, a alternativa 95 encontra-se incorreta.

Dentro do contexto de bancos de dados relacionais, é possível usar as operações de junção.

Essas operações utilizam atributos que operam sobre o mesmo domínio presentes em cada uma das tabelas. Esses atributos são utilizados para juntar ou relacionar uma tabela com a outra, sempre que tivermos os mesmos valores em ambas as tabelas. Vejam que temos mais uma vez uma alternativa correta.

Gabarito: C E C

47. BANCA: CESPE ANO: 2016 ÓRGÃO: TRT-08 CARGO: ANALISTA DE TI - QUESTÃO 13

Acerca de SQL (structured query language), assinale a opção correta.

a) A otimização semântica de consultas utiliza restrições existentes no banco de dados (como atributos únicos, por exemplo) com o objetivo de transformar um SELECT em outro mais eficiente para ser executado.

b) Quando os registros de uma tabela estão ordenados fisicamente em um arquivo, segundo um campo que também é campo-chave, o índice primário passa a se chamar índice cluster.

c) Em uma mesma base de dados, uma instrução de SELECT com união de duas tabelas (INNER JOIN) e uma instrução de SELECT em uma tabela utilizando um SELECT interno de outra tabela (subquery) produzem o mesmo resultado e são executadas com o mesmo desempenho ou velocidade.

d) A normalização de dados é uma forma de otimizar consultas SQL, ao apresentar um modelo de dados com um mínimo de redundância. Isso é atingido quando o modelo estiver na quinta forma normal (5FN).

e) Para obter a quantidade de linhas que atendem a determinada instrução SQL, o processo mais eficiente e rápido é executar o comando SELECT e aplicar uma estrutura de loop para contar as linhas resultantes.

Comentário:

Vamos então analisar cada uma das alternativas acima.

a) Uma transformação que é válida somente porque certa restrição de integridade está em efeito é chamada transformação semântica e a otimização resultante é chamada otimização semântica. A otimização semântica pode ser definida como o processo de transformar uma consulta especificada em outra consulta, qualitativamente diferente, mas da qual se garante que produzirá o mesmo resultado que a original, graças ao fato de que os dados com certeza satisfazem a uma determinada restrição de integridade. Vejam que a definição está de acordo com o descrito na alternativa, logo essa é a nossa resposta.

b) Quando tratamos de índexes podemos de forma resumida classificá-los em três categorias: Índice primário, baseado na chave de ordenação; Índice de agrupamento (clustering), baseado no campo de ordenação não-chave de um arquivo e Índice secundário, baseado em qualquer campo não ordenado de um arquivo. Observem que a alternativa tenta confundir a definição de índice primário com índice de cluster.

c) Não existe nenhuma garantia para saber qual das duas instruções será executada de forma mais rápida, vai depender, por exemplo, do perfil dos dados, do algoritmo utilizado pelo SGBD na execução das consultas e da forma como os índices são criados para cada uma delas. Desta forma, a alternativa está incorreta.

d) A normalização é uma atividade que reduza a redundância dos dados dentro do banco de dados e as anomalias de atualização. Não existe um compromisso do processo de normalização com a otimização de consultas. Se lembramos da estrutura dos modelos dimensionais, sabemos que eles são desnormalizados por uma questão de performance.

e) Mais uma vez, a velocidade da consulta ou seu desempenho vai depender da forma como os dados e os índices estão estruturados. Existem vários algoritmos que pode trazer a quantidade de valores possíveis, escolher qual o melhor deles é uma tarefa delicada. Desta forma, a alternativa encontra-se errada.

Gabarito: A

48. BANCA: CESPE ANO: 2016 ÓRGÃO: TRT-08 CARGO: ANALISTA DE TI - QUESTÃO 3

```
CREATE TABLE predio (
id numeric(7,0), nome varchar(50), local varchar(150),
mnemonico varchar(10),
CONSTRAINT pk_sede PRIMARY KEY (id),
CONSTRAINT uq_sede UNIQUE (mnemonico)
);
CREATE TABLE salas (
codigo numeric(7,0) NOT NULL, local varchar(10),
descricao varchar(50), area numeric(10,2),
CONSTRAINT pk_salas PRIMARY KEY (codigo), CONSTRAINT fk_sede_sala FOREIGN KEY (local) REFERENCES predio (mnemonico)
);
```

Considerando os algoritmos acima, em que são criadas as tabelas predio e salas, assinale a opção cuja expressão SQL apresenta informações do registro da maior sala existente.

a) select c1.local, c1.nome, c2.descricao, c2.area from predio as c1, salas as c2
where c2.local=c1.mnemonico having max(c2.area)

b) select c1.local, c1.nome, c2.descricao, max(c2.area) from predio as c1, salas as c2
where c2.local=c1.id
group by c1.local, c1.nome, c2.descricao

c) select c1.local, c1.nome, c2.descricao from predio as c1, (
select local, descricao, area from salas as c1 where area = (select max(area) from salas as c2 where area>0)
) as c2 where c2.local=c1.mnemonico;

d) select c1.local, c1.nome, c2.descricao, c2.area from predio as c1, (

select local, descricao, area from salas as c1 where area = (select max(area) from salas as c2 where area>0)

) as c2 where c2.id=c1.codigo;

e) select c1.local, c1.nome, c2.descricao, max(c2.area) from predio as c1 join salas as c2

on c2.codigo=c1.id

group by c1.local, c1.nome, c2.descricao

Comentário:

Vamos procurar achar os erros das alternativas distintas da resposta da questão, analisaremos, portanto, na ordem que aparece na questão.

A O erro desta alternativa está no uso da função agregada sem que apareça o group by. Vejam também que não temos nenhuma função agregada descrita na cláusula select, por fim, a sintaxe do comando "having max(c2.area)" está incorreta, seria necessário comparar o valor de max(coluna) com uma constante ou outra variável, por exemplo max(c2.area) > 200;

Veja que na alternativa B existe um erro lógico na comparação "c2.local=c1.id", ela não faz sentido. Não traz para o resultado as tuplas necessárias.

A alternativa C é a nossa resposta. Veja que primeiro é feita uma consulta interna para saber qual a sala que tem a área maior. Depois, outra subconsulta retorna os atributos que fazem parte da tupla que possui a sala com a maior área. Num terceiro momento, na consulta externa, recuperamos a informação do prédio, baseado na comparação do atributo mnemônico que funciona como atributo de ligação ou chave estrangeira, que relaciona as duas tabelas.

A letra D usa o mesmo raciocínio da C, mas peca ao considerar a chave estrangeira outro atributo (c2.id=c1.codigo) desta forma não é possível correlacionar as duas tabelas.

E A alternativa E também não retorna o resultado adequado, precisaríamos de uma restrição sobre a função de agregação para retornar apenas a linha que possuísse o valor máximo, isso poderia ser feito por meio do uso da cláusula having e de uma subconsulta que retornasse o valor máximo.

Gabarito: C

49. BANCA: CESPE ANO: 2015 ÓRGÃO: FUB PROVA: ANALISTA ADMINISTRATIVO - ANALISTA DE TECNOLOGIA DA INFORMAÇÃO

Julgue os itens seguintes, no que se refere à linguagem SQL.

Supondo que seja necessário buscar dados em duas tabelas distintas, o comando select não deve ser escolhido por não possuir os recursos para efetuar a busca em ambas as tabelas e exibir o resultado.

A função max, utilizada conjuntamente com o comando select, retorna o maior valor em um determinado campo que tenha sido incluído na busca.

Comentário:

O primeiro item falha ao dizer que o comando select não possui recurso para buscar dados em duas tabelas distintas. Sabemos que a cláusula FROM pode ser usada para listar ou ainda

relacionar diferentes tabelas. A forma mais simples de usar diferentes colunas é apenas listando seus nomes separados por vírgula. Neste caso você vai forçar o SGBD a executar um produto cartesiano entre as relações passadas.

A outra opção de retornar no resultado dados de mais de uma relação seria por meio do comando JOIN, neste caso teríamos uma execução otimizada visto que os atributos de junção fariam uma restrição no resultado durante o processamento da consulta.

O item 92 trata da função agregada max() que de fato retorna o valor máximo de uma determinada coluna de uma tabela. Vamos aproveitar para falarmos um pouco mais sobre as funções de agregação.

Uma função de agregação recebe um conjunto de valores e retorna um valor de saída. Uma das funções de agregação mais comuns é COUNT, que conta valores não nulos em uma coluna. Por exemplo, para contar o número de cachoeiras associados a um estado, especificar:

SELECT COUNT (u.county_id) AS county_count FROM upfall u;

É possível adicionar DISTINCT na consulta anterior para contar o número de municípios que contenham cachoeiras:

SELECT COUNT (DISTINCT u.county_id) AS county_count FROM upfall u;

O comportamento ALL é o padrão, ele conta todos os valores: COUNT (expressão) é equivalente a COUNT (ALL expressão). COUNT é um caso especial de uma função de agregação porque você pode passar o asterisco (*) para contar a quantidade de linhas retornadas:

SELECT COUNT (*) FROM upfall;

A nulidade é irrelevante quando COUNT (*) é usado porque o conceito de nulo se aplica apenas a colunas, não as linhas como um todo. Todas as outras funções agregadas ignoram valores nulos.

Abaixo apresentamos uma lista com algumas funções agregadas comumente usadas. No entanto, a maioria dos fornecedores de banco de dados implementam funções de agregação além das mostradas.

Função - Descrição

AVG (x) - Retorna a média.

COUNT (x) - Conta os valores não nulos. MAX (x)- Retorna o maior valor.

MEDIAN (x) - Devolve a mediana MIN (x) - Retorna o menor valor.

STDDEV (x) - Retorna o desvio padrão. SUM (x) - Resume todos os números.

VARIANCE (x) - Retorna a variância estatística.

Gabarito: E C

50. BANCA: CESPE ANO: 2008 ÓRGÃO: TJ-DF PROVA: ANALISTA JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO

Quanto a bancos de dados, sistemas gerenciadores de bancos de dados e técnicas correlacionadas de modelagem de dados, julgue os próximos itens.

Na linguagem de consulta SQL (structured query language), é possível obter o resultado de uma consulta SELECT ordenado pelo valor de um ou mais atributos.

Comentário: Sim! Basta utilizar a cláusula ORDER BY e as opções ASC ou DESC, para ordenação, ascendente ou descendente, respectivamente.

Gabarito C.

51. Ano: 2016 Banca: CESPE Órgão: TCE-PR Cargo: Analista de Controle Área: Tecnologia da Informação Questão: 62

A respeito de SQL, assinale a opção correta.

a) A função STDDEV é utilizada para calcular a média aritmética de determinada coluna

b) O produto cartesiano é o resultado da combinação de mais de uma tabela, havendo pelo menos uma coluna em comum entre elas, de maneira que se apresentem os registros que constam simultaneamente em todas as tabelas

b) O resultado de uma *subquery* é utilizado como argumento para uma *query* superior e pode conter uma única linha, múltiplas linhas ou múltiplas linhas e colunas.

d) Uma instrução SQL de *insert* deve citar nominalmente todas as colunas da tabela.

e) As instruções *insert*, *update* e *delete* são processadas no banco de dados após serem executadas pelo usuário, dispensando-se o uso de outro comando para a disponibilização de seus resultados a outros usuários.

Comentário:

Vamos analisar cada uma das alternativas acima.

A alternativa "A" apresenta a função de desvio padrão (STDDEV) e associa esse termo a média aritmética. Portanto, está incorreta.

Na alternativa "B" existe uma definição equivocada de produto cartesiano. O produto cartesiano é uma operação advinda da teoria dos conjuntos relaciona todos os elementos de um conjunto aos elementos de outro conjunto. Quando pensamos no modelo relacional, o relacionamento é feito entre as tuplas das tabelas ou relações participantes da operação. Não existe a necessidade de uma coluna em comum.

C é a alternativa correta! Ele fala de consultas correlacionadas.

O comando insert não precisa citar nominalmente as colunas, basta repassar os valores dos atributos na mesma ordem que foram declarados na definição da tabela. Desta forma, a alternativa C encontra-se errada.

A alternativa E tem uma pegadinha. Você precisa se lembrar do contexto de transações em SQL. A depender do nível de isolamento não basta o usuário executar a instrução de *insert*, *update* ou *delete*. Ele tem que executar a instrução de COMMIT. Sendo assim, devemos assinalar a alternativa como incorreta.

Gabarito: C

52. Ano: 2016 Banca: CESPE Órgão: TCE-PR Cargo: Analista de Controle Área: Tecnologia da Informação Questão: 63

No que se refere a banco de dados, assinale a opção correta

a) DDL (*data definition language*) e DML (*data manipulation language*) são linguagens utilizadas pelos usuários e desenvolvedores para manipular os dados em um banco de dados.

b) Os bancos de dados objeto-relacionais representam uma evolução dos bancos de dados relacionais, pois, incorporam várias funcionalidades anteriormente implementadas nos bancos de dados orientados a objetos.

c) A restrição de asserção de um banco de dados permite a execução de ações automáticas a partir de eventos previamente definidos, por exemplo, a entrada de um CPF com formatação incorreta.

d) Uma *view* representa uma tabela em forma física consolidada a partir de outras tabelas previamente definidas.

e) Em chaves primárias compostas, formadas por mais de um atributo, o valor NULL, é adotado para qualquer atributo exceto para o primeiro na ordem de formação da chave.

Comentário:

Comentaremos abaixo cada uma das alternativas. O CESPE aproveitou essa questão para misturar diversos conceitos de banco de dados. Confesso que gostei da questão.

Na alternativa "A" ele trata de manipular dados, essas operações seriam feitas apenas com DML. Alternativa incorreta, portanto.

A alternativa "B" é a resposta. Essa ideia de que os banco de dados objetos relacionais foi a união dos bancos OO com os relacionais é bem difundida na literatura.

A letra C descreve em uma parte da alternativa o conceito trigger. As asserções são na realidade verificações sobre valores em tabelas de bancos de dados. Elas não vêm associadas a alguma ação. Quando pensamos no modelo de evento- condição-ação devemos associá-lo diretamente aos gatilhos de bancos de dados. Desta forma, podemos considerar a alternativa incorreta.

Uma visão ou *view* é uma construção temporária armazenada na memória principal. Geralmente, não existe do ponto de vista físico. Ela é carregada durante a sua execução. Determinamos, então, a alternativa D como errada.

A alternativa "E" é no mínimo bizarra de tão errada. Dizer que o valor nulo pode fazer parte da chave primária não faz o menor sentido.

Gabarito: B

53. Ano: 2016 Banca: CESPE Órgão: TRT-08 Cargo: Técnico de TI - QUESTÃO 54

tabela t1

codigo	descricao
1	Britain
2	Rich CA
3	Columbia

tabela t2

codigo	valor
1	200
1	150
3	300
4	130

resultado

codigo	descricao	soma
1	Britain	350
3	Columbia	300
2	Rich CA	

a) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 join t2 on t1.codigo=t2.codigo

group by t1.codigo, t1.descricao;

b) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 left join t2 on t1.codigo=t2.codigo

group by t1.codigo, t1.descricao;

c) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 right join t2 on t1.codigo=t2.codigo

group by t1.codigo, t1.descricao;

d) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 full join t2 on t1.codigo=t2.codigo

group by t1.codigo, t1.descricao;

e) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 , t2

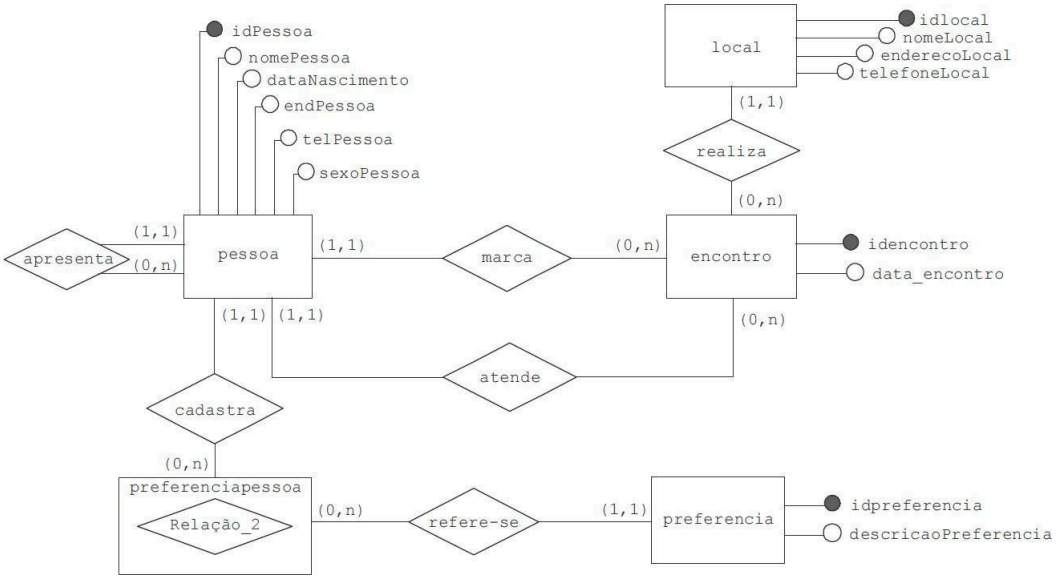
where t1.codigo=t2.codigo group by t1.codigo, t1.descricao;

Comentário:

A tabela Resultado consiste da junção das tabelas t1 e t2 e com uma coluna soma que contém os valores obtidos por meio do agrupamento da coluna valor quando o número de código for igual. Notem que a linha 3 não traz valor algum. Isto acontece por causa do uso do left outer join, que, neste caso, trará os elementos de t1 mesmo não existindo na tabela t2 (considera a tabela da esquerda independentemente da existência de código em t2). Desta forma podemos marcar o gabarito na alternativa B.

Gabarito: B.

Texto para as próximas duas questões: a seguir, são apresentados um modelo entidade-relacionamento conceitual e, na tabela, características dos seus atributos.



pessoa	idPessoa	int(5)
	nomePessoa	varchar(45)
	dataNascimento	Date
	endPessoa	varchar(45)
	telPessoa	varchar(45)
	sexoPessoa	char(1)
local	idlocal	Int(3)
	nomeLocal	varchar(45)
	enderecoLocal	varchar(45)
	telefoneLocal	varchar(45)
encontro	idencontro	int(5)
	data_encontro	Date
preferencia	idpreferencia	int(3)
	descricaoPreferencia	varchar(45)

54. Ano: 2016 Banca: CESPE Órgão: TRE-PE - Cargo 2 TÉCNICO JUDICIÁRIO – ÁREA APOIO ESPECIALIZADO – ESPECIALIDADE PROGRAMAÇÃO DE SISTEMAS

Considerando que os relacionamentos do modelo mostrado tenham sido mapeados, assinale a opção que apresenta comando em linguagem SQL capaz de criar fisicamente a tabela encontro, incluindo-se as chaves estrangeiras necessárias.

a) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL PRIMARY KEY,

'idlocal' INT(3) NULL, 'idpessoa_marca' INT(5) NULL, 'idpessoa_atende' INT(5) NULL, 'data_encontro' DATE NULL,

FOREIGN KEY ('idPessoa') REFERENCES 'pessoa' ('idPessoa'), FOREIGN KEY ('idlocal') REFERENCES 'local' ('idlocal'));

b) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL PRIMARY KEY,

'idlocal' INT(3) NULL, 'idpessoa_marca' INT(5) NULL, 'idpessoa_atende' INT(5) NULL, 'data_encontro' DATE NULL,

FOREIGN KEY fk_pessoa_marca ('idPessoa') REFERENCES 'pessoa' ('idPessoa'), FOREIGN KEY fk_pessoa_atende ('idPessoa') REFERENCES 'pessoa' ('idPessoa'), FOREIGN KEY ('idlocal') REFERENCES 'local' ('idlocal'));

c) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL, 'idpessoa_marca' INT(5) NULL, 'idpessoa_atende' INT(5) NULL, 'data_encontro' DATE NULL, 'idlocal' INT(3) NULL,

PRIMARY KEY ('idencontro'),

CONSTRAINT 'fkpessoa_marca' FOREIGN KEY ('idpessoa_marca') REFERENCES 'pessoa' ('idPessoa'),

CONSTRAINT 'fkpessoa_atende' FOREIGN KEY ('idpessoa_atende') REFERENCES 'pessoa' ('idPessoa'),

CONSTRAINT 'fklocal_encontro' FOREIGN KEY ('idlocal') REFERENCES 'local' ('idlocal'));

d) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL, 'idpessoa_marca' INT(5) NULL,

'idpessoa_atende' INT(5) NULL, 'data_encontro' DATE NULL, 'idlocal' INT(3) NULL,

PRIMARY KEY CONSTRAINT ('idencontro'),

FOREIGN KEY CONSTRAINT 'fkpessoa_marca' ('idpessoa_marca') REFERENCES 'pessoa' ('idPessoa'),

FOREIGN KEY CONSTRAINT 'fkpessoa_atende' ('idpessoa_atende') REFERENCES 'pessoa' ('idPessoa'),

FOREIGN KEY CONSTRAINT 'fklocal_encontro' ('idlocal') REFERENCES 'local' ('idlocal'));

e) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL PRIMARY KEY,

'data_encontro' DATE NULL,

'idpessoa_marca' INT(5) NULL FOREIGN KEY ('idpessoa_marca') REFERENCES 'pessoa' ('idPessoa'),

'idpessoa_atende' INT(5) NULL FOREIGN KEY ('idpessoa_atende') REFERENCES 'pessoa' ('idPessoa'),

'idlocal' INT(3) NULL FOREIGN KEY ('idlocal') REFERENCES 'local' ('idlocal'));

Comentário:

Vamos tentar partir do diagrama. Perceba que além dos atributos da entidade (id e data do encontro), temos que incluir no diagrama as chaves das entidades que fazem parte do relacionamento. Observe também que a chave da entidade pessoa aparece duas vezes, uma como a pessoa que marca o encontro e outra como a que atende o encontro. Além da chave de pessoa, precisamos também definir o local do encontro.

Vejam que a composição dos atributos acima, deve vir acompanhada também, na criação da tabela, pela restrição de integridade referencial. Esse formato encontra-se na alternativa C.

Percebam que precisamos definir nomes diferentes para as restrições que fazem o relacionamento com pessoa, para isso precisamos utilizar a sintaxe correta da criação de CONSTRAINT.

Gabarito: C.

55. Ano: 2016 Banca: CESPE Órgão: TRE-PE - Cargo 2 TÉCNICO JUDICIÁRIO – ÁREA APOIO ESPECIALIZADO – ESPECIALIDADE PROGRAMAÇÃO DE SISTEMAS

QUESTÃO 30 - Considere que, quando da realização do mapeamento do modelo conceitual para o modelo relacional-conceitual, tenham sido criadas na tabela encontro as chaves estrangeiras idpessoa_marca e idpessoa_atende, que identificam, respectivamente, quem marcou um encontro e quem atendeu a pessoa nesse mesmo encontro. A partir do modelo apresentado e dessas informações, assinale a opção que apresenta comando que permite apresentar uma listagem que mostre uma vez o nome de quem não marcou nenhum encontro, mas atendeu pelo menos a um.

a) select distinct nomePessoa

from encontro inner join pessoa on idpessoa_atende = idPessoa and idPessoa

not in (select idpessoa_marca from encontro);

b) select distinct nomePessoa

from pessoa where idPessoa not in (select idpessoa_marca from encontro);

c) select distinct nomePessoa

from encontro full join pessoa on idpessoa_atende = idPessoa;

d) select distinct nomePessoa

from encontro left join pessoa on idpessoa_atende = idPessoa;

e) select distinct nomePessoa

from encontro right join pessoa on idpessoa_atende = idPessoa and

idPessoa not in (select idpessoa_marca from encontro);

Comentário:
Uma das soluções possíveis seria você descobrir todas as pessoas que atenderam e, em seguida, fazer uma restrição das pessoas que não marcaram nenhum encontro. Percebam que o comando que mais se aproxima deste fato é o presente na alternativa A, contudo a ausência da cláusula WHERE pode invalidar a questão. Creio que não tivemos recurso, por isso, o gabarito se manteve inalterado.

Gabarito: A

56. Ano: 2016 Banca: CESPE Órgão: PCPE Cargo: PERITO CRIMINAL Questão: 38.

Na linguagem SQL, o comando *create table* é usado para criar uma tabela no banco de dados; enquanto o relacionamento entre duas tabelas pode ser criado pela declaração

a) null.

b) primary key.

c) constraint.

d) auto_increment.

e) not null.

Comentário:

A sintaxe do comando *create table* presente no SQL/ANSI, apresenta uma cláusula que permite a definição das restrições de integridade de uma tabela: a cláusula constraint. Uma destas restrições é a restrição de integridade referencial. Nela você define o relacionamento entre duas tabelas por meio da criação de chaves estrangeiras. O interessante da definição das restrições de chaves após a definição das colunas é que você pode definir chaves, primárias ou estrangeiras, compostas.

Analisando o exposto no parágrafo anterior podemos assinalar a alternativa C como a nossa resposta. O gabarito do CESPE aponta na mesma direção, não temos, portanto, o que questionar em relação a questão.

Gabarito: C

57. Ano: 2016 Banca: CESPE Órgão: PCPE Cargo: PERITO CRIMINAL Questão: 39.

Na linguagem SQL, quando for necessário obter uma lista e criar uma condição, pode-se utilizar a cláusula

a) min.

b) sum.

c) where.

d) avg.

e) max.

Comentário:

O comando select de SQL permite que você restrinja os valores retornados por uma consulta por meio de condições ou predicados descritos na cláusula where. Perceba que as outras alternativas da questão apresentam funções agregadas, elas fazem agrupamentos de tuplas de uma tabela. Se quisermos fazer restrições sobre valores das funções agregadas utilizamos a cláusula having.
Mais uma vez a resposta está na alternativa C.

Gabarito: C

58. Ano: 2016 Banca: CESPE Órgão: PCPE Cargo: PERITO CRIMINAL Questão: 40.

Em SQL, para alterar a estrutura de uma tabela do banco de dados e incluir nela uma nova *foreign key*, é correto utilizar o comando

a) convert.

b) group by.

c) alter table.

d) update.

e) insert.

Comentário:

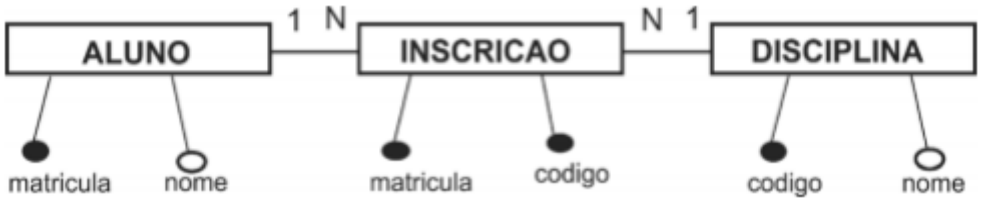
Para fazer alterações em objetos já criados dentro do banco de dados utilizamos o comando *ALTER*. No caso específico de alteração de tabelas fazemos uso do comando *ALTER TABLE*. Por meio desse comando é possível, por exemplo, inserir novas colunas na tabela, modificar nomes de colunas e criar ou modificar restrições de integridade. Para criar uma nova chave estrangeira na tabela basta usar o comando abaixo:

```
ALTER  TABLE
Orders
ADD  CONSTRAINT fk_PerOrders
FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

Percebam que *P_Id* é um atributo da tabela Orders que referencia outro atributo da tabela Persons. Mais uma vez, nosso gabarito está na alternativa C.

Gabarito: C

59. Ano: 2015 Banca: CESPE Órgão: TRE-PI – Questão 58



O modelo lógico apresentado dá origem às tabelas ALUNO, INSCRICAO e DISCIPLINA. Considerando esse modelo e sabendo que não há nenhum procedimento armazenado no

banco de dados, assinale a opção que apresenta código em SQL ANSI que resultará corretamente na listagem de matricula e nome dos alunos que estão inscritos (INSCRICAO) em mais de duas disciplinas.

a) SELECT aluno.matricula, aluno.nome FROM inscricao, aluno, disciplina

WHERE inscricao.matricula=aluno.matricula AND inscricao.codigo=disciplina.codigo GROUP BY aluno.matricula, aluno.nome HAVING COUNT(*) > 2

b) SELECT aluno.matricula, aluno.nome FROM inscricao, aluno, disciplina

WHERE inscricao.matricula=aluno.matricula AND inscricao.codigo=disciplina.codigo AND COUNT(*) > 2

c) SELECT aluno.matricula, aluno.nome GROUP BY aluno.matricula, aluno.nome FROM inscricao, aluno, disciplina

WHERE inscricao.matricula=aluno.matricula AND inscricao.Codigo=diciplina. Codigo AND COUNT

d) SELECT aluno.matricula, aluno.nome FROM inscricao, aluno, disciplina

WHERE inscricao.matricula=aluno.matricula AND quantidade > 2

GROUP BY inscricao, aluno, disciplina

e) SELECT aluno.matricula, aluno.nome, SUM() > 2 FROM inscricao, aluno, disciplina

WHERE inscricao.matricula=aluno.matricula AND inscricao.codigo=disciplina.codigo GROUP BY aluno.matricula, aluno.nome

Comentários:

Para resolver essa questão basta perceber que vamos precisar agrupar os alunos de acordo com a quantidade de disciplinas que eles cursam. Logo precisamos agrupar pelo nome e matrícula e em seguida fazer uma contagem da quantidade de tuplas para cada aluno. Tal valor corresponde a quantidade de disciplinas nas quais cada aluno está matriculado. A próxima etapa é usar a cláusula having para verificar se esse valor é maior que dois (count (*)>2). Esse comando está descrito na alternativa A.

Gabarito: A

60. Ano: 2015 Banca: CESPE Órgão: TRE-PI – Questão 59

Considerando um SGBD que respeite os padrões SQL ANSI-99, assinale a opção que apresenta corretamente um comando SQL para apagar determinados registros de uma tabela pessoa (cpf, nome, sexo) que contém registros cujo campo sexo apresenta valores iguais a 'M' e 'F'.

a) DROP pessoa WHERE sexo='M'

b) UPDATE pessoa SET sexo=NULL WHERE sexo='M'

c) FROM pessoa WHERE sexo='M' DELETE sexo

d) DELETE sexo FROM pessoa WHERE sexo='M'

e) DELETE FROM pessoa WHERE sexo='M'

Comentários:
Essa questão quer saber do nosso conhecimento a respeito do comando usado para apagar determinados registros de uma tabela, no caso usamos para tal finalidade o comando DELETE FROM tabela WHERE restrição; que pode ser verificado na alternativa E.

Adicionalmente, o enunciado da questão parece confuso e pode gerar interpretações equivocadas. Se a intenção é apagar os registros cujo campo sexo contenha valores 'M' ou 'F', o comando SQL correto seria semelhante à opção E:

DELETE FROM pessoa WHERE sexo = 'M' OR sexo = 'F';

O uso do operador lógico OR é adequado para especificar múltiplos valores possíveis para o campo sexo.

Gabarito: E

QUESTÕES COMENTADAS

1. (UFMT - Analista de Sistemas - CESGRANRIO – 2024)

A respeito do uso de procedimentos armazenados e de gatilhos em um banco de dados relacional, verifica-se, em relação à sua aplicabilidade, que

(A) os gatilhos são ideais para encapsular lógica de apresentação em um banco de dados.

(B) os gatilhos são usados, principalmente, para encapsular lógica de negócios complexa e reutilizável.

(C) os procedimentos armazenados são preferíveis para impor restrições de integridade referencial.

(D) os procedimentos armazenados são adequados para automatizar a execução de ações em resposta a eventos específicos.

(E) ambos, procedimentos armazenados e gatilhos, são exclusivamente usados para consultas complexas em bancos de dados.

Comentário: Vamos analisar cada afirmativa:

A: Gatilhos não são usados para lógica de apresentação.

B: Gatilhos são mais reativos a eventos do que focados em lógica de negócios complexa e reutilizável.

C: Procedimentos armazenados não são preferíveis para integridade referencial (isso é gerenciado por chaves estrangeiras e restrições).

D: Procedimentos armazenados podem automatizar ações em resposta a eventos, especialmente quando invocados por gatilhos ou agendados.

E: A afirmação de uso exclusivo para consultas complexas é incorreta.

Portanto, a única afirmativa correta é a opção (D).

Gabarito: D

2. (UFMT - Analista de Sistemas - CESGRANRIO – 2024)

Considere a criação de um banco de dados relacional para a biblioteca de uma universidade.

Nesse contexto, Data Definition Language, DDL; Data Manipulation Language, DML; e Data Query Language, DQL, são utilizados para

(A) DDL: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DML: Inserir registros nas tabelas com detalhes específicos de um novo livro.

DQL: Recuperar todos os livros de um determinado autor.

(B) DDL: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DML: Atualizar a quantidade disponível de exemplares de um livro.

DQL: Inserir registros nas tabelas com detalhes específicos de um novo livro.

(C) DDL: Atualizar a quantidade disponível de exemplares de um livro.

DML: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DQL: Recuperar todos os livros emprestados por um usuário específico.

(D) DDL: Inserir registros nas tabelas com detalhes específicos de um novo livro.

DML: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DQL: Atualizar a quantidade disponível de exemplares de um livro.

(E) DDL: Recuperar todos os livros de um determinado autor.

DML: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DQL: Inserir registros nas tabelas com detalhes específicos de um novo livro.

Comentário: A resposta correta é a letra A. No contexto da criação de um banco de dados relacional para uma biblioteca universitária, as linguagens DDL, DML e DQL são utilizadas para as seguintes finalidades:

- DDL (Data Definition Language): É utilizada para definir a estrutura do banco de dados, incluindo a criação, alteração e exclusão de tabelas e outros objetos do banco de dados.

Portanto, a DDL é responsável por criar as tabelas para armazenar informações sobre livros, autores e editoras.

- DML (Data Manipulation Language): É utilizada para manipular os dados dentro do banco de dados, incluindo operações como inserção, atualização, exclusão e consulta de registros nas tabelas. Assim, a DML é empregada para inserir registros nas tabelas com detalhes específicos de um novo livro.

- DQL (Data Query Language): É utilizada para fazer consultas aos dados armazenados no banco de dados, recuperando informações específicas conforme os critérios definidos pelo usuário. Portanto, a DQL é utilizada para recuperar todos os livros de um determinado autor.

Portanto, a letra A está correta porque descreve corretamente as finalidades das linguagens DDL, DML e DQL no contexto da criação de um banco de dados relacional para uma biblioteca universitária.

Gabarito: A

3. CESGRANRIO/IPEA/2024/Infraestrutura

Considere que um banco de dados foi criado para dar apoio à avaliação de instrumentos e políticas de gestão de trânsito no Brasil, nos últimos cinco anos. Os dados foram organizados e persistidos nas três seguintes tabelas, definidas de acordo com modelo relacional de dados:
SINISTRO, com dados dos acidentes de trânsito; MUNICIPIO, com dados de municípios; e RODOVIA, com dados de rodovias estaduais e federais.
SINISTRO (cod-sinistro, data-e-hora, localizacao, cod-rodovia, cod-municipio, quantidade-de-vitimas) RODOVIA (cod-rodovia, nome, estadual-ou-federal)
MUNICIPIO (cod-municipio, uf, quantidade-de-habitantes)

Os atributos que formam as chaves primárias de cada tabela estão sublinhados.

Na tabela SINISTRO, há duas chaves estrangeiras: cod-rodovia, que indica onde ocorreu o sinistro, caso ele tenha ocorrido em uma rodovia, e cod-municipio, que indica em que município ocorreu o sinistro.

Nesse contexto, considere o seguinte comando SQL:

```
SELECT S.cod-rodovia, S.data-e-hora, quantidade-de-vitimas
FROM SINISTRO S
WHERE S.cod-rodovia IN (
SELECT R cod-rodovia
FROM RODOVIA R
WHERE R estadual-ou-federal = federal)
AND EXISTS (
SELECT *
FROM MUNICIPIO M
```

```
WHERE M.cod-municipio = S cod-municipio
AND M.quantidade-de-habitantes < 50000)
```

Os resultados produzidos pela execução desse comando apresentam o código da rodovia, a data e hora e a quantidade de vítimas de sinistros ocorridos em

a) rodovias federais que passam por municípios com menos de 50.000 habitantes.

b) rodovias federais, em municípios com menos de 50.000 habitantes.

b) rodovias federais que têm como origem ou destino municípios com menos de 50.000 habitantes.

d) município com menos de 50.000 habitantes ou em rodovias federais.

e) município com menos de 50.000 habitantes com duas ou mais rodovias federais.

Comentário: O comando SQL realiza uma seleção de dados da tabela SINISTRO, buscando o código da rodovia, a data e hora e a quantidade de vítimas dos sinistros. Ele utiliza uma subconsulta para verificar se a rodovia do acidente é federal e outra subconsulta para verificar se o município tem menos de 50.000 habitantes. Agora vamos procurar a explicação correta entre as alternativas: (B) rodovias federais, em municípios com menos de 50.000 habitantes. O comando SQL seleciona sinistros que ocorreram em rodovias federais e em municípios com menos de 50.000 habitantes, conforme a condição especificada nas subconsultas. Dessa forma, a alternativa correta é a letra (B) rodovias federais, em municípios com menos de 50.000 habitantes.

Gabarito: B

4. CESGRANRIO/IPEA/2024/Ciência de Dados

Para a avaliação de políticas públicas na área de Segurança Alimentar e Nutricional, um município brasileiro utilizou dados persistidos em três relações (tabelas) organizadas de acordo com o seguinte modelo relacional:

PRODUTO (cod-produto, nome-produto, grupo-alimentar)

FORNECEDOR (CNPJ, nome-empresa, tipo)

COMPRADO (CNPJ, cod-produto, data, quantidade, valor)

Os atributos que formam as chaves primárias de cada tabela estão sublinhados.

Nesse contexto, considere o comando SQL apresentado a seguir.

```
SELECT P.cod-produto, SUM (quantidade)
FROM PRODUTO P, FORNECEDOR F, COMPRADO C
WHERE P.cod-produto = C.cod-produto
AND C.CNPJ = F.CNPJ
AND F.tipo = 'agricultura familiar'
GROUP BY P.cod-produto
HAVING SUM (quantidade) > 10000
```

Os resultados produzidos pela execução desse comando apresentam o código do produto e a soma das quantidades compradas dos produtos de

a) fornecedores com mais de 10.000 produtos distintos.

b) fornecedores do tipo 'agricultura familiar' que tiveram mais de 10.000 unidades compradas.

c) fornecedores do tipo 'agricultura familiar' que fornecem mais de 10.000 produtos distintos.

d) todos os fornecedores do tipo 'agricultura familiar'.

e) produtos que tiveram mais de 10.000 unidades compradas.

Comentário:

O comando SQL apresentado realiza uma consulta que envolve as tabelas PRODUTO, FORNECEDOR e COMPRADO, onde:

- É selecionado o código do produto (P.cod-produto) e a soma das quantidades compradas (SUM(quantidade)).
- A junção é feita considerando as relações entre as tabelas, onde o código do produto em PRODUTO deve ser igual ao código do produto em COMPRADO (P.cod-produto = C.cod-produto), e o CNPJ em COMPRADO deve ser igual ao CNPJ em FORNECEDOR (C.CNPJ = F.CNPJ).
- Além disso, é filtrado apenas os fornecedores do tipo 'agricultura familiar' (F.tipo = 'agricultura familiar').

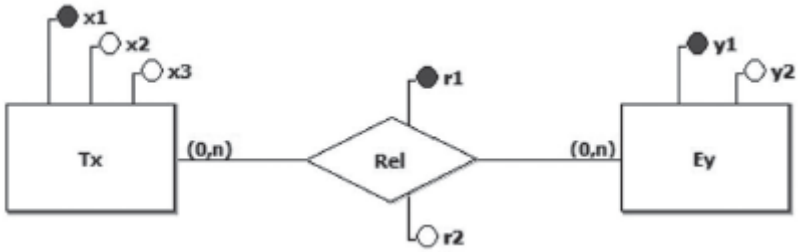
- Por fim, é agrupado o resultado pelo código do produto (GROUP BY P.cod-produto) e, através da cláusula HAVING, é selecionado apenas os grupos em que a soma das quantidades compradas é maior que 10.000 (HAVING SUM(quantidade) > 10000).

Portanto, os resultados produzidos pela execução desse comando apresentam o código do produto e a soma das quantidades compradas dos produtos que foram comprados em quantidades superiores a 10.000 unidades, apenas dos fornecedores do tipo 'agricultura familiar'. Assim, a opção correta é a letra (B).

Gabarito: B

5. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023

Considere o seguinte diagrama E-R:



Foi criado um conjunto de tabelas relacionais, a partir do modelo E-R acima. Uma vez que as regras de transformações de entidades e relações para tabelas relacionais independem dos tipos de dados dos atributos, todos os atributos do modelo E-R acima foram tratados como itens de dados do tipo cadeia de caracteres (TEXT).

As tabelas resultantes são as seguintes:

```
CREATE TABLE TX (
        X1              TEXT            NOT NULL,
        X2              TEXT            NOT NULL,
        X3              TEXT            NOT NULL,
        PRIMARY KEY (X1));
CREATE TABLE EY (
         Y1             TEXT            NOT NULL,
         Y2             TEXT            NOT NULL,
         PRIMARY KEY (Y1));
```

Qual transformação da relação Rel irá preservar a semântica do diagrama E-R apresentado?

a) CREATE TABLE REL (

X1 TEXT NOT NULL,

Y1 TEXT NOT NULL,

```
    R1              TEXT        NOT NULL,
    R2              TEXT        NOT NULL,
    PRIMARY KEY (X1, Y1),
    FOREIGN KEY (X1)
            REFERENCES TX (X1),
    FOREIGN KEY (Y1)
            REFERENCES EY (Y1));
```

b)
```
CREATE TABLE REL (
    X1              TEXT        NOT NULL,
    Y1              TEXT        NOT NULL,
    R1              TEXT        NOT NULL,
    R2              TEXT        NOT NULL,
    PRIMARY KEY (X1, R1),
    FOREIGN KEY (X1)
            REFERENCES TX (X1),
    FOREIGN KEY (Y1)
            REFERENCES EY (Y1));
```

c)
```
CREATE TABLE REL (
    X1              TEXT        NOT NULL,
    Y1              TEXT        NOT NULL,
    R1              TEXT        NOT NULL,
    R2              TEXT        NOT NULL,
    PRIMARY KEY (Y1, R1),
    FOREIGN KEY (X1)
            REFERENCES TX (X1),
```

```
    FOREIGN KEY (Y1)
            REFERENCES EY (Y1));
```

d)
```
CREATE TABLE REL (
    X1          TEXT        NOT NULL,
    Y1          TEXT        NOT NULL,
    R1          TEXT        NOT NULL,
    R2          TEXT        NOT NULL,
    PRIMARY KEY (X1, Y1, R1),
    FOREIGN KEY (X1)
            REFERENCES TX (X1),
    FOREIGN KEY (Y1)
            REFERENCES EY (Y1));
```

e)
```
CREATE TABLE REL (
    X1          TEXT        NOT NULL,
    Y1          TEXT        NOT NULL,
    R1          TEXT        NOT NULL,
    R2          TEXT        NOT NULL,
    PRIMARY KEY (R1),
    FOREIGN KEY (X1)
            REFERENCES TX (X1),
   FOREIGN KEY (Y1)
            REFERENCES EY (Y1));
```

Comentário:
Dado que existem duas entidades conectadas por um relacionamento muitos-para-muitos (N:N), onde Tx pode ter várias instâncias de Ey, e Ey pode conter várias instâncias de Tx, torna-se necessário criar uma nova tabela para armazenar informações de relacionamento, conforme

explicado anteriormente. Nesse contexto, é crucial incluir as chaves primárias das entidades Tx e Ey, juntamente com os atributos r1 e r2 do tipo de relacionamento presente no diagrama, como atributos de chave estrangeira da entidade Rel. Até esse ponto, as linhas de comando seriam as seguintes:

```
CREATE TABLE REL (
    X1 TEXT NOT NULL,
    Y1 TEXT NOT NULL,
    R1 TEXT NOT NULL,
    R2 TEXT NOT NULL,
    FOREIGN KEY (X1) REFERENCES TX (X1),
    FOREIGN KEY (Y1) REFERENCES EY (Y1)
);
```

É essencial observar que, geralmente, a combinação das chaves estrangeiras formará a chave primária da tabela Rel, permitindo a definição do relacionamento muitos-para-muitos. O diagrama indica também que o relacionamento Rel possui um atributo identificador próprio, que faz parte da chave primária, chamado r1. Por sua vez, o atributo r2 é um atributo comum. Para atender a essas condições, é necessário modificar o comando apresentado, incluindo os três valores (x1, y1, r1) como chaves primárias de Rel, resultando no comando final:

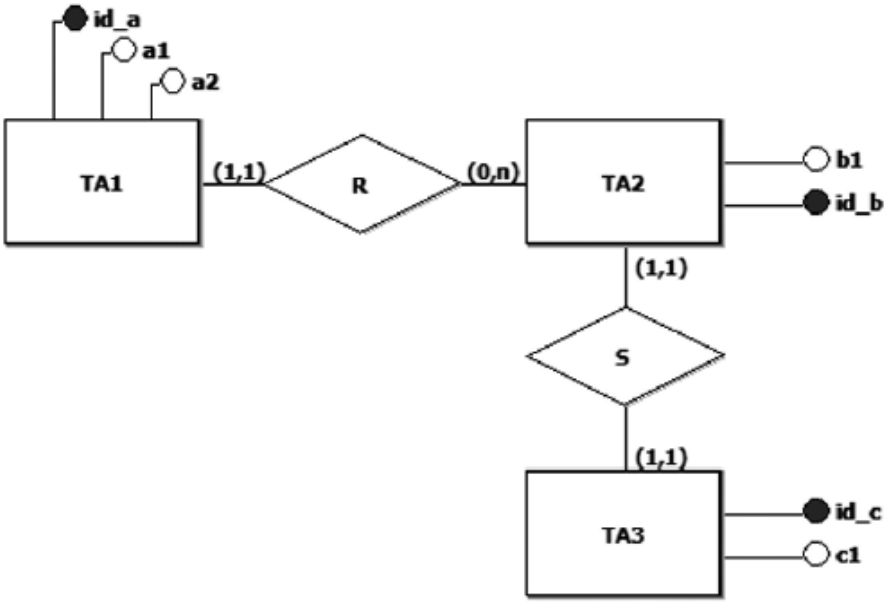
```
CREATE TABLE REL (
    X1 TEXT NOT NULL,
    Y1 TEXT NOT NULL,
    R1 TEXT NOT NULL,
    R2 TEXT NOT NULL,
    PRIMARY KEY (X1, Y1, R1),
    FOREIGN KEY (X1) REFERENCES TX (X1),
    FOREIGN KEY (Y1) REFERENCES EY (Y1)
);
```

Portanto, a resposta correta é representada pela alternativa D.

Gabarito: D

6. CESGRANRIO - Ana Desenv (AgeRIO)/AgeRIO/Tecnologia da Informação/2023

Considere o diagrama E-R a seguir.



Para simplificar, todos os atributos desse modelo E-R devem ser considerados itens de dados do tipo cadeia de caracteres (TEXT).

A partir desse diagrama, foi produzido um conjunto de tabelas relacionais por meio da aplicação de regras de transformação.

Essas regras preservaram a semântica do modelo E-R, além de propiciarem mais eficiência nas operações de junção sobre as tabelas obtidas.

Qual conjunto de tabelas atende às transformações aplicadas?

a)

```
CREATE TABLE TA1 (
ID_A        TEXT        NOT NULL,
A1          TEXT        NOT NULL,
A2          TEXT        NOT NULL,
ID_B        TEXT        NOT NULL,
PRIMARY KEY (ID_A),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B));

CREATE TABLE TA2 (
ID_B        TEXT        NOT NULL,
B1          TEXT        NOT NULL,
ID_C        TEXT        NOT NULL UNIQUE,
C1          TEXT        NOT NULL,
PRIMARY KEY (ID_B));
```

```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
PRIMARY KEY (ID_A));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
C1          TEXT            NOT NULL,
PRIMARY KEY (ID_B));

CREATE TABLE R (
ID_A        TEXT            NOT NULL,
ID_B        TEXT            NOT NULL,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_A)
    REFERENCES TA1(ID_A),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B));
```

b)

```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
PRIMARY KEY (ID_A));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_A        TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
C1          TEXT            NOT NULL,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_A)
    REFERENCES TA1(ID_A));
```

c)

d)

```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
ID_B        TEXT            NOT NULL,
PRIMARY KEY (ID_A),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_C)
    REFERENCES TA3(ID_C));

CREATE TABLE TA3 (
ID_C        TEXT            NOT NULL,
C1          TEXT            NOT NULL,
ID_B        TEXT            NOT NULL UNIQUE,
PRIMARY KEY (ID_C),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B));
```

```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
PRIMARY KEY (ID_A));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_A        TEXT            NOT NULL,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_A)
    REFERENCES TA1(ID_A));

CREATE TABLE TA3 (
ID_C        TEXT            NOT NULL,
C1          TEXT            NOT NULL,
PRIMARY KEY (ID_C));

CREATE TABLE S (
ID_B        TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B),
FOREIGN KEY (ID_C)
    REFERENCES TA3(ID_C));
```

e)

Comentário:

Notamos a presença de dois relacionamentos, R e S. Desconsiderando as chaves estrangeiras e tipos de dados, temos as seguintes definições para cada tabela:

```
CREATE TABLE TA1 (
   ID_A,
   A1,
   A2,
   PRIMARY KEY (ID_A)
);

CREATE TABLE TA2 (
   ID_B,
   B1,
   ID_A,
   PRIMARY KEY (ID_B),
   FOREIGN KEY (ID_A) REFERENCES TA1 (ID_A)
);

CREATE TABLE TA3 (
   ID_C,
```

```
    C1,
    PRIMARY KEY (ID_C)
);
```

Ao definir os relacionamentos, utilizamos as regras de mapeamento E-R. Para o relacionamento R entre TA1 e TA2 (1:N), adicionamos uma coluna no lado N do relacionamento (Tabela TA2):

```
CREATE TABLE TA2 (
    ID_B,
    B1,
    ID_A,
    PRIMARY KEY (ID_B),
    FOREIGN KEY (ID_A) REFERENCES TA1 (ID_A)
);
```

Para o relacionamento S entre TA2 e TA3 (1:1), realizamos a fusão das tabelas:

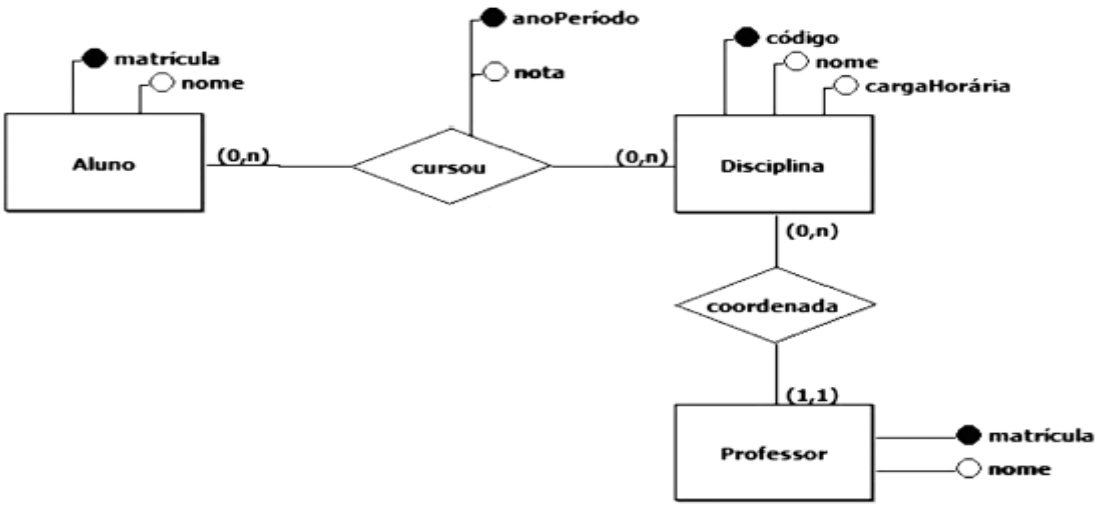
```
CREATE TABLE TA2 (
    ID_B,
    B1,
    ID_A,
    ID_C,
    C1,
    PRIMARY KEY (ID_B),
    FOREIGN KEY (ID_A) REFERENCES TA1 (ID_A)
);
```

Com base nessas definições, concluímos que o gabarito correto é a LETRA C.

Gabarito: C

7. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Infraestrutura/2023

Seja o seguinte modelo E-R:



Deverão ser criadas tabelas relacionais que preservem a semântica do modelo E-R apresentado e que atendam à 2a forma normal.

Qual esquema atende a esses requisitos?

a)
```
CREATE TABLE ALUNO (
        MATRICULA INTEGER NOT NULL,
        NOME TEXT NOT NULL,
        PRIMARY KEY (MATRICULA));

CREATE TABLE DISCIPLINA (
      CODIGO TEXT NOT NULL,
       NOME TEXT NOT NULL,
       CARGA_HOR INTEGER NOT NULL,
       MAT_PROF INTEGER NOT NULL,
       NOME_PROF TEXT NOT NULL,
       PRIMARY KEY (CODIGO));

CREATE TABLE CURSOU (
       MAT_ALUNO INTEGER NOT NULL,
       COD_DISC TEXT NOT NULL,
       ANO_PER TEXT NOT NULL,
       NOTA NUMERIC (3,1) NOT NULL,
       PRIMARY KEY (MAT_ALUNO, COD_DISC, ANO_PER),
       FOREIGN KEY (MAT_ALUNO)
              REFERENCES ALUNO (MATRICULA),
      FOREIGN KEY (COD_DISC)
             REFERENCES DISCIPLINA (CODIGO));
```

b)
```
CREATE TABLE ALUNO (
        MATRICULA INTEGER NOT NULL,
        NOME TEXT NOT NULL,
        PRIMARY KEY (MATRICULA));
CREATE TABLE DISCIPLINA (
      CODIGO TEXT NOT NULL,
      NOME TEXT NOT NULL,
       CARGA_HOR INTEGER NOT NULL,
       MAT_PROF INTEGER NOT NULL,
       NOME_PROF TEXT NOT NULL,
       PRIMARY KEY (CODIGO));

CREATE TABLE CURSOU (
       MAT_ALUNO INTEGER NOT NULL,
       COD_DISC TEXT NOT NULL,
       ANO_PER TEXT NOT NULL,
       NOTA NUMERIC (3,1) NOT NULL,
       NOME_DISCIPLINA TEXT NOT NULL,
       PRIMARY KEY (MAT_ALUNO, COD_DISC, ANO_PER),
      FOREIGN KEY (MAT_ALUNO)
             REFERENCES ALUNO (MATRICULA),
      FOREIGN KEY (COD_DISC)
             REFERENCES DISCIPLINA (CODIGO));
```

c) CREATE TABLE ALUNO (
MATRICULA INTEGER NOT NULL,
NOME TEXT NOT NULL,
PRIMARY KEY (MATRICULA));

CREATE TABLE PROFESSOR (
MATRICULA INTEGER NOT NULL,
NOME TEXT NOT NULL,
PRIMARY KEY (MATRICULA));

CREATE TABLE DISCIPLINA (
CODIGO TEXT NOT NULL,
NOME TEXT NOT NULL,
CARGA_HOR INTEGER NOT NULL,
MAT_PROF INTEGER NOT NULL,
PRIMARY KEY (CODIGO),
FOREIGN KEY (MAT_PROF)

REFERENCES PROFESSOR (MATRICULA));

CREATE TABLE CURSOU (
MAT_ALUNO INTEGER NOT NULL,
COD_DISC TEXT NOT NULL,
ANO_PER TEXT NOT NULL,
NOTA NUMERIC (3,1) NOT NULL,
PRIMARY KEY (MAT_ALUNO, COD_DISC),
FOREIGN KEY (MAT_ALUNO)
REFERENCES ALUNO (MATRICULA),
FOREIGN KEY (COD_DISC)
REFERENCES DISCIPLINA (CODIGO));

d) CREATE TABLE ALUNO (
MATRICULA INTEGER NOT NULL,
NOME TEXT NOT NULL,
PRIMARY KEY (MATRICULA));

CREATE TABLE PROFESSOR (
MATRICULA INTEGER NOT NULL,
NOME TEXT NOT NULL,
PRIMARY KEY (MATRICULA));
CREATE TABLE DISCIPLINA (
CODIGO TEXT NOT NULL,
NOME TEXT NOT NULL,
CARGA_HOR INTEGER NOT NULL,
MAT_PROF INTEGER NOT NULL,
PRIMARY KEY (CODIGO),
FOREIGN KEY (MAT_PROF)

```
            REFERENCES PROFESSOR (MATRICULA));
CREATE TABLE CURSOU (
      MAT_ALUNO INTEGER NOT NULL,
      COD_DISC TEXT NOT NULL,
      ANO_PER TEXT NOT NULL,
      NOTA NUMERIC (3,1) NOT NULL,
      NOME_DISCIPLINA TEXT NOT NULL,
      PRIMARY KEY (MAT_ALUNO, COD_DISC, ANO_PER),
      FOREIGN KEY (MAT_ALUNO)
            REFERENCES ALUNO (MATRICULA),
       FOREIGN KEY (COD_DISC)
             REFERENCES DISCIPLINA (CODIGO));

e)  CREATE TABLE ALUNO (
          MATRICULA INTEGER NOT NULL,
          NOME TEXT NOT NULL,
         PRIMARY KEY (MATRICULA));
CREATE TABLE PROFESSOR (
      MATRICULA INTEGER NOT NULL,
      NOME TEXT NOT NULL,
      PRIMARY KEY (MATRICULA));

CREATE TABLE DISCIPLINA (
     CODIGO TEXT NOT NULL,
     NOME TEXT NOT NULL,
     CARGA_HOR INTEGER NOT NULL,
     MAT_PROF INTEGER NOT NULL,
    PRIMARY KEY (CODIGO),
    FOREIGN KEY (MAT_PROF)
            REFERENCES PROFESSOR (MATRICULA));
CREATE TABLE CURSOU (
      MAT_ALUNO INTEGER NOT NULL,
      COD_DISC TEXT NOT NULL,
      ANO_PER TEXT NOT NULL,
      NOTA NUMERIC (3,1) NOT NULL,
      NOME_DISCIPLINA TEXT NOT NULL,
       PRIMARY KEY (MAT_ALUNO, COD_DISC, ANO_PER),
       FOREIGN KEY (MAT_ALUNO)
             REFERENCES ALUNO (MATRICULA),
       FOREIGN KEY (COD_DISC)
              REFERENCES DISCIPLINA (CODIGO));
```

Comentário:

A questão apresenta dois relacionamentos, o primeiro relacionamento do tipo N-N exige uma tabela de ligação, ou seja, temos uma tabela para Aluno, uma para Cursou e outra para disciplina. Já o segundo relacionamento possui uma peculiaridade, o professor tem uma cardinalidade mínima e máxima iguais a 1. Assim, podemos adicionar os atributos de professor à

tabela disciplina. Desta forma, o comando SQL que descreve os comandos corretos para criação das tabelas encontra-se na alternativa A.

Gabarito: A

8. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023

Uma empresa de investimentos financeiros busca identificar novas oportunidades de negócio para pessoas jurídicas, em especial dentre aquelas que têm como característica a adoção de governança ambiental, social e corporativa (conhecida como ESG, uma sigla em inglês).

Considere que existe um banco de dados nessa empresa com as seguintes tabelas (todas as chaves primárias são numéricas):

Empresa (CNPJ, razaoSocial, endereco)
Caracteristica (cod, sigla, nome)
Tem (CNPJ, cod)

Que comando SELECT do SQL retorna apenas o CNPJ e a razão social das empresas que não têm "ESG" como característica?

a) SELECT *

FROM Empresa

WHERE Caracteristica.Sigla <> 'ESG'

b) SELECT E.CNPJ, E.razaoSocial

FROM Empresa E

JOIN Tem T ON (E.CNPJ = T.CNPJ)

WHERE Tem.cod = 'ESG'

c) SELECT E.CNPJ, E.razaoSocial

FROM Empresa E

JOIN Tem T ON (E.CNPJ = T.CNPJ)

JOIN Caracteristica C ON (C.cod = T.cod)

WHERE C.nome <> 'ESG'

d) SELECT E.CNPJ, E.razaoSocial

FROM Empresa E

WHERE E.CNPJ NOT IN (

SELECT T.CNPJ

```
    FROM Tem T
    JOIN Caracteristica C ON (T.cod = C.cod)
    WHERE C.sigla = 'ESG'
  )
```

e)
```
SELECT Empresa.*
FROM Empresa, Tem
WHERE Empresa.CNPJ = Tem.cod
AND Tem.cod <> 'ESG'
```

Comentário:
A opção correta que retorna apenas o CNPJ e a razão social das empresas que não têm "ESG" como característica é:

```
d) SELECT E.CNPJ, E.razaoSocial
FROM Empresa E
WHERE E.CNPJ NOT IN (
    SELECT T.CNPJ
    FROM Tem T
    JOIN Caracteristica C ON (T.cod = C.cod)
    WHERE C.sigla = 'ESG'
);
```

Essa consulta utiliza a cláusula NOT IN para filtrar as empresas que não têm a característica "ESG".

Gabarito: D

9. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023

O banco de dados de uma empresa que comercializa seguros pessoais possui as seguintes tabelas:

Pessoa (email, nome, unidadeFederativaNascimento, faixaEtaria)
UF (sigla, nome)
Faixa (nome, menorIdade, maiorIdade)

A coluna "unidadeFederativaNascimento" da tabela Pessoa é uma chave estrangeira que referencia a coluna "sigla" da tabela UF; a coluna "faixaEtaria" da tabela Pessoa é uma chave estrangeira que aponta para a coluna "nome" da tabela Faixa.

A tabela Faixa possui os seguintes dados:

nome	menorIdade	maiorIdade
Jovens	- não informada -	19
Adultos	20	59
Idosos	60	- não informada -

Considere o seguinte comando:

```
SELECT COUNT(*)
FROM Pessoa P, Faixa F
WHERE P.faixaEtaria = F.nome
AND P.unidadeFederativaNascimento = 'RJ'
AND F.maiorIdade <= 19
```

Esse comando SQL
a) apresenta quantas são as pessoas que estão na tabela Pessoa, que são jovens e que nasceram no estado do Rio de Janeiro.
b) apresenta o nome e o email de jovens nascidos no Rio de Janeiro.
c) agrupa pessoas por faixa etária e mostra quantos são os grupos com pessoas nascidas no Rio de Janeiro.
d) realiza uma operação equivalente à união de dois outros comandos SQL.
e) agrupa pessoas por UF e mostra quantos são os grupos com jovens.

Comentário:
O comando SQL apresentado conta quantas são as pessoas que estão na tabela Pessoa, que são jovens (faixa etária correspondente à faixa "Jovens" na tabela Faixa) e que nasceram no estado do Rio de Janeiro (UF 'RJ'). Portanto, a opção correta é: a) Apresenta quantas são as pessoas que estão na tabela Pessoa, que são jovens e que nasceram no estado do Rio de Janeiro.

Gabarito: A

10. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023

O banco de dados de uma empresa de investimentos financeiros possui as seguintes tabelas:
Empresa (CNPJ, razaoSocial, endereco)
UF (sigla, nome)
O que o comando SQL "SELECT CNPJ, sigla FROM Empresa, UF" recupera desse banco de dados?
a) A sigla da UF das sedes das empresas cadastradas.
b) Alguns pares de CNPJ e sigla, onde o nome da UF é igual à razão social da empresa.
c) O CNPJ das empresas cadastradas cuja sigla de UF esteja na tabela UF.
d) Pares de CNPJ e sigla de todas as empresas cadastradas com as UFs de suas respectivas sedes.
e) Todos os pares de CNPJ e sigla possíveis, de todas as empresas e de todas as UFs cadastradas.

Comentário:
O comando SQL "SELECT CNPJ, sigla FROM Empresa, UF" recupera todos os pares de CNPJ e sigla de todas as empresas e de todas as UFs cadastradas. A sintaxe utilizada (sem uma condição explícita de junção ou filtragem) resulta em um produto cartesiano entre as tabelas Empresa e UF, retornando todas as combinações possíveis de CNPJ e sigla, sem considerar alguma relação específica entre elas. Portanto, a opção correta é:

e) Todos os pares de CNPJ e sigla possíveis, de todas as empresas e de todas as UFs cadastradas.

Gabarito: E

11. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023

A CNAE (Classificação Nacional de Atividades Econômicas), de responsabilidade do IBGE, possui códigos que são utilizados para caracterizar as atividades econômicas das empresas no Brasil. Por exemplo: empresas da área de construção de edifícios utilizam o CNAE de código 4120-4/00 para caracterizar a sua atividade econômica principal.

Considere que existe um banco de dados em uma empresa, que concede empréstimos a outras empresas, com as seguintes tabelas:

Empresa (CNPJ, razaoSocial, endereco, atividade)
CNAE (codigo, descricao)

A coluna "atividade" da tabela Empresa é uma chave estrangeira que referencia a coluna "codigo" da tabela CNAE.

Que comando SELECT do SQL retorna apenas o CNPJ e a razão social das empresas cuja atividade econômica principal é a construção de edifícios (código 4120-4/00)?

a) SELECT *

FROM Empresa E, CNAE C

WHERE E.atividade = C.codigo

b) SELECT *

FROM Empresa

WHERE atividade = 'construção de edifícios'

c) SELECT CNPJ, razaoSocial

FROM CNAE

WHERE codigo = '4120-4/00'

d) SELECT CNPJ, razaoSocial

```
FROM Empresa E, CNAE C

WHERE E.atividade = C.codigo

AND C.codigo = 'construção de edifícios'
```

e)
```
SELECT CNPJ, razaoSocial

FROM Empresa

WHERE atividade = '4120-4/00'
```

Comentário:
Esta questão aborda a utilização de comandos SQL para realizar consultas em um banco de dados que relaciona empresas e suas atividades econômicas por meio da Classificação Nacional de Atividades Econômicas (CNAE). A correta compreensão dos comandos é essencial para selecionar os dados desejados de forma eficaz.

A opção correta (letra e) demonstra o conhecimento necessário para filtrar as empresas cuja atividade econômica principal é a construção de edifícios, usando corretamente o código da CNAE. A resposta destaca a importância de entender a estrutura do banco de dados, incluindo as relações entre as tabelas e como realizar consultas que atendam aos requisitos específicos da questão.

As demais opções apresentam equívocos, tais como o uso inadequado de junções, erros de filtragem, e confusões entre códigos e descrições.

Gabarito: E

12. CESGRANRIO - PTNM (TRANSPETRO)/TRANSPETRO/Informática/2023

Uma empresa aérea possui um sistema de informação para registrar as reservas de passagens de seus passageiros nos voos que oferece. O esquema desse banco de dados foi criado com os seguintes comandos SQL:

```
CREATE TABLE PASSAGEIRO (
        CODIGO                  NUMERIC(5) NOT NULL,
        NOME                    TEXT NOT NULL,
        PRIMARY KEY (CODIGO));

CREATE TABLE VOO (
        NUMERO                  NUMERIC(3) NOT NULL,
        ORIGEM                  TEXT NOT NULL,
        DESTINO                 TEXT NOT NULL,
        NUM_LUGARES             NUMERIC(3) NOT NULL,
        PRIMARY KEY (NUMERO));

CREATE TABLE RESERVA (
        NUM_VOO                 NUMERIC(3) NOT NULL,
        CD_PASS                 NUMERIC(5) NOT NULL,
```

```
PRIMARY KEY (NUM_VOO, CD_PASS),
FOREIGN KEY (NUM_VOO)
        REFERENCES VOO(NUMERO),
FOREIGN KEY (CD_PASS)
        REFERENCES PASSAGEIRO(CODIGO));
```

Na Figura a seguir, são exibidos os estados atuais dessas tabelas.

--------TABELA PASSAGEIRO----------		----------TABELA VOO----------				--------TABELA RESERVA-----	
CODIGO	NOME	NUMERO	ORIGEM	DESTINO	NUM_LUGARES	NUM_VOO	CD_PASS
11232	FABIO CARNEIRO	357	RIO DE JANEIRO	FORTALEZA	30	357	65789
13121	CARLOS AMARAL	635	SALVADOR	RIO DE JANEIRO	20	357	13121
22578	ANA MARIA RIOS	784	FORTALEZA	BRASILIA	20	357	56390
22667	CECILIA LOPES	998	PORTO ALEGRE	CURITIBA	20	357	22578
23799	KAREN RIBEIRO					357	22667
44532	VICENTE DE CARVALHO					357	56123
44677	JULIA VASCONCELOS					357	44909
44909	ALEX MEDEIROS					635	45131
45131	FLAVIA NASCIMENTO					635	89567
51734	EDSON RIBEIRO					635	11232
56123	PEDRO COIMBRA					635	44532
56390	FELIPE DE SOUZA					635	78980

65123	POLIANA PEDROSA					635	44677
65789	JOANA RODRIGUES					784	23799
79980	ROSANA DA SILVA					784	51734
89567	IARA DE MELO					784	65123

Qual comando SQL será executado sem produzir erro?
a) DELETE FROM PASSAGEIRO WHERE NOME = 'VICENTE DE CARVALHO';
b) DELETE FROM VOO WHERE DESTINO = 'CURITIBA';
c) INSERT INTO RESERVA VALUES(998, 56000);
d) INSERT INTO PASSAGEIRO VALUES(56390, 'RICARDO GONÇALVES');
e) UPDATE VOO SET NUMERO = 532 WHERE DESTINO = 'RIO DE JANEIRO';
Comentário:
Vamos comentar algo sobre cada um dos comandos acima:

A) ERRADO - A alternativa A é considerada incorreta, pois menciona que a exclusão de um passageiro da tabela PASSAGEIRO com o nome 'VICENTE DE CARVALHO' não produzirá erro. No entanto, isso gerará um erro, pois o código associado ao passageiro está sendo usado como chave estrangeira na tabela RESERVA. A tentativa de exclusão violaria a integridade referencial.

B) CORRETO - A alternativa B está correta, pois a exclusão de um destino 'CURITIBA' da tabela VOO não afeta diretamente outras tabelas que possam depender dessa informação. Não há violação de integridade referencial ao excluir essa entrada.

C) ERRADO - 56000 não pertence a nenhum passageiro - A alternativa C é correta ao apontar que a tentativa de inserir uma reserva com o número de voo 998 e o código de passageiro 56000 não é válida, pois o código 56000 não existe na tabela PASSAGEIRO.

D) ERRADO - 56390 é uma chave primária e já está associada ao FELIPE DE SOUZA - A alternativa D destaca corretamente que a tentativa de inserir um novo passageiro com o código 56390 e nome 'RICARDO GONÇALVES' resultará em erro, pois o código 56390 já é uma chave primária existente associada ao passageiro 'FELIPE DE SOUZA'.

E) ERRADO - O voo está associado à tabela RESERVA e deve ser atualizado lá também. A alternativa E destaca a necessidade de atualizar a tabela RESERVA ao alterar o número do voo na tabela VOO. Essa preocupação com a consistência dos dados é válida, indicando que as alterações devem ser refletidas em todas as tabelas que referenciam a informação modificada.

Gabarito: B

13. CESGRANRIO - PTNM (TRANSPETRO)/TRANSPETRO/Informática/2023

Na Figura abaixo, é exibido o estado atual de uma tabela que registra as disciplinas de um curso e seus pré-requisitos.

--- TABELA CURSO----	
DISCIPLINA	PRE_REQUISITO
PROJETOS DE SISTEMAS	BANCO DE DADOS I
PROJETO FINAL	PROJETO DE SISTEMAS
BANCO DE DADOS I	MODELAGEM DE DADOS
PROGRAMAÇÃO	ESTRUTURAS DE DADOS

Uma consulta SQL nessa tabela foi executada e produziu como resposta uma relação com duas linhas, cujos valores são exibidos abaixo.

PROJETO DE SISTEMAS
BANCO DE DADOS I

Qual consulta SQL foi executada?

a) SELECT DISCIPLINA FROM CURSO
UNION
SELECT PRE_REQUISITO FROM CURSO;

b) SELECT DISCIPLINA, PRE_REQUISITO FROM CURSO
WHERE DISCIPLINA = 'PROJETO DE SISTEMAS' AND
PRE_REQUISITO = 'BANCO DE DADOS I' OR
DISCIPLINA = 'BANCO DE DADOS I' AND PRE_REQUISITO = 'PROJETO DE SISTEMAS';

c) SELECT DISCIPLINA FROM CURSO
INTERSECT
SELECT PRE_REQUISITO FROM CURSO;

d) SELECT C1.DISCIPLINA, C2.PRE_REQUISITO
FROM CURSO C1
RIGHT JOIN CURSO C2
ON C1.DISCIPLINA = C2.PRE_REQUISITO;

e) SELECT C1.DISCIPLINA, C2.PRE_REQUISITO
FROM CURSO C1
LEFT JOIN CURSO C2
ON C1.DISCIPLINA = C2.PRE_REQUISITO;

Comentário:
A operação INTERSECT retorna apenas as linhas que são comuns às duas consultas. Portanto, para obter as disciplinas que são também pré-requisitos, a consulta correta seria:

c) SELECT DISCIPLINA FROM CURSO INTERSECT SELECT PRE_REQUISITO FROM CURSO;

Essa consulta retornará as disciplinas que são pré-requisitos, e a resposta correta para a questão seria a opção (c).

Gabarito: C

14. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Infraestrutura/2023

Considere um banco de dados de uma empresa contendo a tabela HIERARQUIA, que possui as colunas Chefe e Subordinado definindo uma hierarquia de cargos. Considere, também, uma instância que permite ilustrar as informações contidas nesta tabela.

CHEFE	SUBORDINADO
----------	----------
Pedro	Renata
Roberto	Beatriz
Gabriela	Fernando
Lucas	Daniela
João	Rafaela
Carlos	Juliana
Beatriz	Carlos
Rafaela	Marcelo
Pedro	Roberto
Marcelo	Gabriela
Renata	Felipe
João	Pedro
Daniela	Antônio
Felipe	Clara
Juliana	Lucas

Uma consulta feita para determinar os subordinados, diretos ou indiretos, do Carlos retorna os seguintes nomes: {Juliana, Lucas, Daniela e Antônio}.

A expressão, na linguagem de consultas SQL, para esta consulta é

a)
```
WITH RECURSIVE subordinados AS (
        SELECT SUBORDINADO
        FROM HIERARQUIA
        WHERE CHEFE = 'Carlos'

        UNION

      SELECT t.SUBORDINADO
      FROM HIERARQUIA t
      JOIN subordinados s ON t.CHEFE = s.SUBORDINADO
)
SELECT SUBORDINADO FROM subordinados;
```

b)
```
WITH RECURSIVE subordinados AS (
        SELECT CHEFE
```

```
        FROM HIERARQUIA
        WHERE SUBORDINADO = 'Carlos'

        UNION ALL
        SELECT t.CHEFE
        FROM HIERARQUIA t
        JOIN subordinados s ON t.SUBORDINADO = s.CHEFE
)
SELECT CHEFE FROM subordinados;

c) WITH RECURSIVE subordinados AS (
         SELECT SUBORDINADO
         FROM HIERARQUIA
         WHERE CHEFE = 'Carlos'

         UNION ALL

        SELECT t.SUBORDINADO
         FROM HIERARQUIA t
         JOIN subordinados s ON t.CHEFE = s.SUBORDINADO
)
SELECT SUBORDINADO FROM subordinados;

d) WITH subordinados AS (
         SELECT SUBORDINADO
          FROM HIERARQUIA
          WHERE CHEFE = 'Carlos'

           UNION ALL

           SELECT t.SUBORDINADO
           FROM HIERARQUIA t
           JOIN subordinados s ON t.SUBORDINADO = s.CHEFE
)
SELECT SUBORDINADO FROM subordinados;

e) WITH RECURSIVE subordinados AS (
         SELECT SUBORDINADO
         FROM HIERARQUIA
         WHERE CHEFE = 'Carlos'

         UNION ALL

         SELECT t.CHEFE
         FROM HIERARQUIA t
        JOIN subordinados s ON t.SUBORDINADO = s.CHEFE
)
SELECT SUBORDINADO FROM subordinados;
```

Comentário:
A alternativa correta encontra-se na alternativa C. Esta consulta está corretamente procurando por subordinados de Carlos e usando a UNION ALL para incluir todos os subordinados diretos e indiretos. A consulta recursiva está juntando as linhas onde o chefe é igual ao subordinado na tabela recursiva, o que retornará todos os subordinados de Carlos.

Nesta consulta, a cláusula WITH RECURSIVE é usada para criar uma consulta recursiva. Inicialmente, ela seleciona os subordinados diretos do Carlos (primeira parte do UNION). Em seguida, a parte recursiva da consulta (UNION ALL com JOIN subordinados) é usada para selecionar os subordinados indiretos, continuando a busca na tabela HIERARQUIA até não haver mais subordinados a serem encontrados.

A questão acabou sendo anulada pela banca, pois a alternativa A também se encontra correta.

Gabarito: C (Anulada)

15. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Processos de Negócios/2023

Considere o esquema relacional simplificado de um banco de dados de uma Universidade, contendo, ao menos, as seguintes tabelas:

Departamentos (ID_Departamento, Nome, Localizacao)
Professores (ID_Professor, Nome, ID_Departamento)
Alunos (ID_Aluno, Nome, ID_Professor_Orientador)

Entre outras restrições que devem ser consideradas, sabe-se que alunos podem ser orientados por professores e que todo professor pertence a um departamento.

Considerando-se o esquema relacional apresentado, o trecho de código em linguagem SQL DDL que respeita todas as restrições de integridade citadas é

a)
```
CREATE TABLE Departamentos (
        ID_Departamento INT PRIMARY KEY,
        Nome VARCHAR(100),
        Localizacao VARCHAR(100)
   );
CREATE TABLE Professores (
       ID_Professor INT PRIMARY KEY,
       Nome VARCHAR(100),
       ID_Departamento INT
);
CREATE TABLE Alunos (
      ID_Aluno INT PRIMARY KEY,
      Nome VARCHAR(100),
      ID_Professor_Orientador INT
);
```

b)
```
CREATE TABLE Departamentos (
        ID_Departamento INT PRIMARY KEY,
```

```
        Nome VARCHAR(100),
        Localizacao VARCHAR(100)
);
CREATE TABLE Professores (
    ID_Professor INT PRIMARY KEY,
    Nome VARCHAR(100),
    ID_Departamento INT,
    FOREIGN KEY (ID_Departamento) REFERENCES Departamentos(ID_Departamento)
);
CREATE TABLE Alunos (
     ID_Aluno INT PRIMARY KEY,
     Nome VARCHAR(100),
     ID_Professor_Orientador INT,
    FOREIGN KEY (ID_Professor_Orientador) REFERENCES Professores(ID_Professor)
);
c) CREATE TABLE Departamentos (
        ID_Departamento INT PRIMARY KEY,
        Nome VARCHAR(100),
        Localizacao VARCHAR(100)
);
CREATE TABLE Professores (
    ID_Professor INT PRIMARY KEY,
    Nome VARCHAR(100),
    ID_Departamento INT
);
CREATE TABLE Alunos (
    ID_Aluno INT PRIMARY KEY,
    Nome VARCHAR(100),
    ID_Professor_Orientador INT,
    FOREIGN KEY (ID_Professor_Orientador) REFERENCES Professores(ID_Professor)
);
d) CREATE TABLE Departamentos (
        ID_Departamento INT PRIMARY KEY,
        Nome VARCHAR(100),
        Localizacao VARCHAR(100)
);
CREATE TABLE Professores (
     ID_Professor INT PRIMARY KEY,
     Nome VARCHAR(100),
     ID_Departamento INT
);
CREATE TABLE Alunos (
    ID_Aluno INT PRIMARY KEY,
    Nome VARCHAR(100),
    ID_Professor_Orientador INT
);
e) CREATE TABLE Departamentos (
        ID_Departamento INT PRIMARY KEY,
```

```
        Nome VARCHAR(100),
        Localizacao VARCHAR(100)
);
CREATE TABLE Professores (
      ID_Professor INT,
      Nome VARCHAR(100),
      ID_Departamento INT,
      PRIMARY KEY (ID_Professor),
      FOREIGN KEY (ID_Departamento) REFERENCES Departamentos(ID_Departamento)
);
CREATE TABLE Alunos (
       ID_Aluno INT PRIMARY KEY,
       Nome VARCHAR(100),
       ID_Professor_Orientador INT
);
```

Comentário:
A opção que respeita todas as restrições de integridade citadas é:

b) CREATE TABLE Departamentos (ID_Departamento INT PRIMARY KEY, Nome VARCHAR(100), Localizacao VARCHAR(100)); CREATE TABLE Professores (ID_Professor INT PRIMARY KEY, Nome VARCHAR(100), ID_Departamento INT, FOREIGN KEY (ID_Departamento) REFERENCES Departamentos(ID_Departamento)); CREATE TABLE Alunos (ID_Aluno INT PRIMARY KEY, Nome VARCHAR(100), ID_Professor_Orientador INT, FOREIGN KEY (ID_Professor_Orientador) REFERENCES Professores(ID_Professor));

Nesta opção, as tabelas **Departamentos**, **Professores**, e **Alunos** são criadas com as chaves primárias, e as referências entre as tabelas são estabelecidas por meio de chaves estrangeiras. Isso garante que um professor pertence a um departamento e que um aluno é orientado por um professor.

Gabarito: B

16. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Processos de Negócios/2023

Considere a tabela Familia, apresentada abaixo, que possui as colunas Pai e Filho e uma instância exemplo que permite ilustrar as informações contidas nessa tabela.

Pai	Filho
Andre	Carlos
Andre	Claudio
Bruno	Diego
Bruno	Daniel
Carlos	Eriko
Carlos	Fabio
Daniel	Gustavo

Diego	Heleno
Eriko	Isidoro
Fabio	João

Uma consulta SQL feita para exibir os nomes dos netos de Carlos retorna os nomes Isidoro e João. A expressão, em linguagem SQL, dessa consulta é

a) SELECT f2.filho

FROM Familia f1 INNER JOIN Familia f2 ON f1.filho = f2.pai

WHERE f1.pai = 'Carlos';

b) SELECT f1.pai

FROM Familia f1 INNER JOIN Familia f2 ON f1.pai = f2.filho

WHERE f1.filho = 'Carlos';

c) SELECT f1.pai

FROM Familia f1 INNER JOIN Familia f2 ON f1.filho = f2.pai

WHERE f1.pai = 'Carlos';

d) SELECT

f2.filho FROM Familia f1 INNER JOIN Familia f2 ON f1.pai = f2.pai

WHERE f1.pai = 'Carlos';

e) SELECT f2.filho

FROM Familia f1 INNER JOIN Familia f2 ON f1.filho = f2.filho

WHERE f1.pai = 'Carlos';

Comentário:
A expressão SQL que retorna os nomes dos netos de Carlos é:

SELECT f2.filho FROM Familia f1 INNER JOIN Familia f2 ON f1.filho = f2.pai WHERE f1.pai = 'Carlos';

Nesta consulta, a tabela **Familia** é usada duas vezes, sendo renomeada como **f1** e **f2**. A junção (**INNER JOIN**) é feita considerando que o filho da primeira instância (**f1.filho**) é igual ao pai da segunda instância (**f2.pai**). Em seguida, é aplicado um filtro para selecionar os netos de Carlos, onde o pai (**f1.pai**) é igual a 'Carlos'. Logo, a resposta está na alternativa A.

Gabarito: A

17. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Infraestrutura/2023

Gatilhos (*triggers*) e procedimentos armazenados (*stored procedures*) são componentes fundamentais em sistemas de gerenciamento de banco de dados (SGBDs) relacionais. Tanto os gatilhos quanto os procedimentos armazenados desempenham papéis vitais e, muitas vezes, complementares em aplicações baseadas em banco de dados relacionais, sendo escolhidos de acordo com as necessidades específicas de uma aplicação ou de um sistema. A respeito de gatilhos e de procedimentos armazenados, tem-se que

a) tanto os gatilhos quanto os procedimentos armazenados são executados manual e explicitamente por um usuário ou por um programa de aplicação, e não podem ser disparados automaticamente por eventos de atualização em um banco de dados.
b) tanto os gatilhos quanto os procedimentos armazenados são disparados automaticamente por eventos de atualização em um banco de dados, e não podem ser ativados manual e explicitamente por um usuário ou por um programa de aplicação.
c) gatilhos, ao contrário dos procedimentos armazenados, não podem expressar a lógica de regras de negócios, e são utilizados exclusivamente para validação de dados.
d) gatilhos são disparados automaticamente em resposta a eventos de atualização em um banco de dados, enquanto procedimentos armazenados precisam ser chamados explicitamente.
e) gatilhos e procedimentos armazenados não podem coexistir em um mesmo sistema de banco de dados, pois possuem funcionalidades idênticas e redundantes.

Comentário:
A opção correta é: d) Gatilhos são disparados automaticamente em resposta a eventos de atualização em um banco de dados, enquanto procedimentos armazenados precisam ser chamados explicitamente.

Gatilhos são ativados automaticamente por eventos específicos no banco de dados, como inserções, atualizações ou exclusões, enquanto procedimentos armazenados precisam ser chamados explicitamente por um usuário ou aplicação. Gatilhos são comumente utilizados para impor regras de integridade, validar dados ou realizar ações automáticas em resposta a certos eventos. Procedimentos armazenados são blocos de código SQL que podem ser reutilizados e chamados quando necessário, mas não são automaticamente acionados por eventos de banco de dados.

Gabarito: D

18. CESGRANRIO - Tec Cien (BASA)/BASA/Tecnologia da Informação/2022

Em banco de dados, os gatilhos são utilizados, entre outros objetivos, para implementar restrições de integridade. Considere que em um banco de dados de um banco comercial há duas tabelas, CLIENTE (chave primária CPF) e SERVICO (chave primária composta por CPF e identificação do serviço), e há a restrição de que um cliente não pode estar associado a mais de cinco serviços.

Que definição deve ser utilizada para implementar essa restrição por meio de um gatilho?
a) BEFORE INSERT ON CLIENTE
b) AFTER INSERT ON CLIENTE
c) BEFORE INSERT ON SERVICO

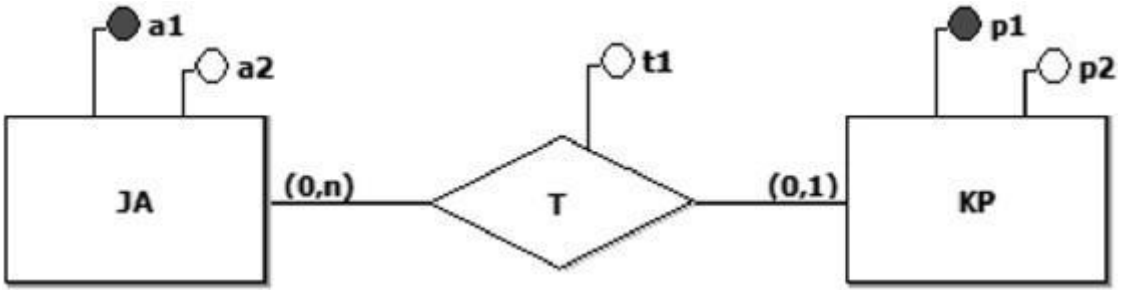
d) AFTER INSERT ON SERVICO
e) WHEN CLIENTE INSERT ON SERVICO

Comentário:
A restrição que você descreveu, onde um cliente não pode estar associado a mais de cinco serviços, envolve a tabela SERVICO e, portanto, o gatilho deve ser definido na tabela SERVICO. Dessa forma, a opção correta seria: c) BEFORE INSERT ON SERVICO.

Gabarito: C

19. CESGRANRIO - PNS (ELETRONUCLEAR)/ELETRONUCLEAR/Analista de Sistemas/Aplicações e Segurança de TIC/2022

Considere o diagrama E-R abaixo.



Qual esquema relacional preserva a semântica desse modelo E-R?
a)

CREATE TABLE JA (

A1 TEXT NOT NULL,

A2 TEXT NOT NULL,

PRIMARY KEY (A1));

CREATE TABLE KP (

P1 TEXT NOT NULL,

P2 TEXT NOT NULL,

A1 TEXT NULL,

T1 TEXT NULL,

PRIMARY KEY (P1),

FOREIGN KEY (A1)

```
REFERENCES JA (A1));
```

b)

```
CREATE TABLE JA (
A1 TEXT NOT NULL,
A2 TEXT NOT NULL,
PRIMARY KEY (A1));
 CREATE TABLE KP (
P1 TEXT NOT NULL,
P2 TEXT NOT NULL,
PRIMARY KEY (P1));
 CREATE TABLE T (
P1 TEXT NOT NULL,
A1 TEXT NOT NULL,
T1 TEXT NOT NULL,
PRIMARY KEY (P1, A1),
FOREIGN KEY (P1)
REFERENCES KP (P1),
FOREIGN KEY (A1)
REFERENCES JA (A1));
```

c)

```
CREATE TABLE JA (
A1 TEXT NOT NULL,
A2 TEXT NOT NULL,
PRIMARY KEY (A1));
```

```
CREATE TABLE KP (
P1 TEXT NOT NULL,
P2 TEXT NOT NULL,
PRIMARY KEY (P1));

CREATE TABLE T (
P1 TEXT NOT NULL,
A1 TEXT NOT NULL,
T1 TEXT NOT NULL,
PRIMARY KEY (P1),
FOREIGN KEY (P1)
REFERENCES KP (P1),
FOREIGN KEY (A1)
REFERENCES JA (A1));
```

d)

```
CREATE TABLE T (
A1 TEXT NOT NULL,
A2 TEXT NOT NULL,
P1 TEXT NOT NULL,
P2 TEXT NOT NULL,
T1 TEXT NOT NULL,
PRIMARY KEY (P1, A1));
```

e)

```
CREATE TABLE JA (
```

A1 TEXT NOT NULL,

A2 TEXT NOT NULL,

P1 TEXT NULL,

T1 TEXT NULL,

PRIMARY KEY (A1)

FOREIGN KEY (P1)

REFERENCES KP (P1));

CREATE TABLE KP (

P1 TEXT NOT NULL,

P2 TEXT NOT NULL,

PRIMARY KEY (P1));

Comentário:
Neste caso, o examinador demanda conhecimento de mapeamento conceitual para o Modelo Entidade-Relacionamento (E-R). Ele explicitou que ambas as entidades estão usando um relacionamento um-para-muitos (1:N).

O principal método a ser empregado em um relacionamento 1:N é a adição de coluna (do lado N do relacionamento). Portanto, teríamos:

Tabela JA (lado N da relação): conterá todos os seus atributos + atributos do relacionamento + chave estrangeira apontando para a chave primária de KP.
Tabela KP: permanece exatamente como está.

Traduzindo para o SQL:

```
CREATE TABLE JA
(
  A1 TEXT NOT NULL, -- sua própria coluna
  A2 TEXT NOT NULL, -- sua própria coluna
  P1 TEXT NULL, -- coluna chave estrangeira
  T1 TEXT NULL, -- coluna do relacionamento
  PRIMARY KEY (A1), -- definição da sua própria chave primária
  FOREIGN KEY (P1) REFERENCES KP (P1) -- definição da chave estrangeira apontando para a tabela KP
);
```

```
CREATE TABLE KP
(
  P1 TEXT NOT NULL, -- sua própria coluna
  P2 TEXT NOT NULL, -- sua própria coluna
  PRIMARY KEY (P1) -- definição da sua própria chave primária
);
```

Portanto, a resposta correta para a questão é a LETRA E.

LETRA A - Adicionou a coluna no lado errado do relacionamento (o que compromete a cardinalidade do modelo).

LETRA B - Criou uma tabela associativa (preferencialmente usado em casos de N:N, e como temos a LETRA E, nem precisamos analisar essa tabela).

LETRA C - Semelhante à LETRA B.

LETRA D - Realizou a fusão, o que é proibido em relacionamentos 1:N.

Gabarito: E

20. CESGRANRIO - Tec Cien (BASA)/BASA/Tecnologia da Informação/2022

Considere que em um banco de dados de um banco comercial há duas tabelas:

PESSOA_FISICA (CPF, nome, email, telefone)
CLIENTE (CPF, nome, email, telefone).

Um funcionário de TI recebeu a tarefa de identificar corretamente quais pessoas físicas, cadastradas na tabela PESSOA_FISICA, ainda não eram clientes, pois não estavam cadastradas na tabela CLIENTE. Para isso, ele utilizou um comando SELECT em SQL. Que trecho, em SQL, faz parte de uma das possíveis soluções para essa tarefa?

a) … WHERE PESSOA_FISICA.CPF NOT IN (SELECT CPF FROM CLIENTE…
b) … HAVING PESSOA_FISICA.CPF != CLIENTE.CPF…
c) … WHERE PESSOA_FISICA.CPF <> CLIENTE.CPF...
d) … DISTINCT PESSOA_FISICA.CPF FROM CLIENTE WHERE ...
e) … IN PESSOA_FISICA BUT NOT IN CLIENTE...

Comentário:
A opção correta para identificar as pessoas físicas que não são clientes na tabela CLIENTE é a opção a):

... WHERE PESSOA_FISICA.CPF NOT IN (SELECT CPF FROM CLIENTE…

Essa cláusula NOT IN permite selecionar registros da tabela PESSOA_FISICA onde o CPF não está presente na lista de CPFs da tabela CLIENTE, ou seja, aqueles que não são clientes. As demais opções apresentam sintaxes incorretas ou inadequadas para realizar essa tarefa específica.

Gabarito: A

21. CESGRANRIO - Tec Cien (BASA)/BASA/Tecnologia da Informação/2022

As tabelas a seguir fazem parte do banco de dados da área de recursos humanos de uma empresa. Elas registram os dados referentes aos empregados e aos seus dependentes.

```
CREATE TABLE EMPREGADO (
   MATRICULA INTEGER NOT NULL,
    NOME TEXT NOT NULL,
    DATA_NASC TEXT NOT NULL,
    CERT_RESRV INTEGER UNIQUE NULL,
    PRIMARY KEY (MATRICULA));

CREATE TABLE DEPENDENTE (
    MAT_EMP INTEGER NOT NULL,
    NUM_SEQ INTEGER NOT NULL,
    NOME TEXT NOT NULL,
    DATA_NASC TEXT NOT NULL,
    PRIMARY KEY (MAT_EMP, NUM_SEQ)
    FOREIGN KEY (MAT_EMP)
REFERENCES EMPREGADO (MATRICULA));
```

O estado corrente desse banco de dados é exibido nas figuras abaixo.

EMPREGADO

MATRICULA	NOME	DATA_NASC	CERT_RESRV
11111	Paulo Menezes	24/05/1991 00:00	234811
22222	Ana Maria Carvalho	25/07/1983 00:00	null
33333	Alexandre Cardoso	11/08/1989 00:00	101678

DEPENDENTE

MAT_EMP	NUM_SEQ	NOME	DATA_NAS
22222	1	Valéria	31/12/2017 00:00
22222	2	Pedro	09/06/2015 00:00
33333	1	Joana	22/03/2019 00:00
33333	2	Mariana	25/07/2020 00:00

Qual comando INSERT irá falhar, na tentativa de incluir uma nova linha em uma das tabelas desse banco de dados?

a) INSERT INTO DEPENDENTE(NUM_SEQ,MAT_EMP,DATA_NASC,NOME)

VALUES(00,11111,datetime('2015-01-06'),'Luiz Paulo');

b) INSERT INTO DEPENDENTE(NOME,DATA_NASC,NUM_SEQ,MAT_EMP)

VALUES('Maria Paula',datetime('2017-08-11'),3,11111);

c) INSERT INTO EMPREGADO VALUES(55555,'Antônia Pinto',datetime('1994-04-01'),NULL);
d) INSERT INTO EMPREGADO VALUES(66666,'Adriana Andrade',datetime('1985-06-04'));
e) INSERT INTO EMPREGADO VALUES(44444,'Nilce Peçanha',datetime('1999-09-06'),'');

Comentário:
Vejamos as alternativas:

a) O comando está correto, apresenta os dados e o nome das colunas passados na mesma ordem, embora estejam diferentes da ordem salva no banco.

b) O comando está correto, apresenta os dados e o nome das colunas passados na mesma ordem, embora estejam diferentes da ordem salva no banco.

c) O comando não passa de forma explícita os nomes das colunas, dessa forma os dados são informados na ordem correta. A coluna CERT_RESRV da tabela EMPREGADO recebe um dado *NULL* através desse comando, o que não gera qualquer problema, pois a coluna pode receber dados NULL, como é possível ver no exemplo da tabela passada pela questão.

d) O comando não passa de forma explícita os nomes das colunas, por isso os dados são informados na ordem correta, entretanto a coluna CERT_RESRV não recebe dados o que irá gerar um erro na execução do comando. Ainda que a coluna aceite dados nulos, alguma informação precisa ser indicada. Logo, essa é a nossa respsta.

e) - O comando não passa de forma explícita os nomes das colunas, por isso os dados são informados na ordem correta. Outro detalhe importante nessa alternativa é a maneira usada para inserção de dados NULL. É possível inserir dados nulos (*NULL*) em uma tabela de forma não explícita através de duas aspas ('') sem valores entre elas no espaço onde seria inserido o dado (mesmo que o valor não seja uma String, Varchar, etc.). O interpretador de comandos do SGBD irá inserir um dado nulo quando reconhecer padrão. Como é possível inserir dados nulos na coluna CERT_RESRV, o comando dessa alternativa não irá falhar.

Gabarito: D

22. CESGRANRIO - PNS(ELETRONUCLEAR)/ELETRONUCLEAR/Analista de Sistemas/Aplicações e Segurança de TIC/2022

O controle diário da utilização de passes de metrô em uma cidade é feito por programas que utilizam um banco de dados composto pelas seguintes tabelas:

```
CREATE TABLE PASSE (
NUM INTEGER NOT NULL,
DATA_EXP DATE NOT NULL,
NUM_VIAGENS INTEGER NOT NULL,
PRIMARY KEY (NUM))

CREATE TABLE REG_VIAGEM (
NUM INTEGER NOT NULL,
NUM_ROLETA INTEGER NOT NULL,
DATA_VIAGEM DATE NOT NULL,
FOREIGN KEY (NUM)
REFERENCES PASSE (NUM))
```

A tabela PASSE contém uma linha para cada passe vendido pela empresa que administra o metrô. A coluna DATA_EXP informa a data de emissão do passe, e a coluna NUM_VIAGENS informa o número de viagens em que o passe poderá ser usado (número máximo de viagens). Este número não sofre alteração ao longo do tempo.

A tabela REG_VIAGEM contém uma linha para cada viagem em que o passe foi usado. A coluna NUM_ROLETA informa a roleta na qual o passe foi inserido, e a coluna DATA_VIAGEM informa a data em que o usuário inseriu o passe na roleta.

Qual consulta SQL permite obter os números dos passes que nunca foram usados, juntamente com os números dos passes que já esgotaram o número de viagens realizadas?

a)

```
SELECT A.NUM

FROM PASSE A

WHERE A.NUM_VIAGENS = (SELECT COUNT(*) FROM REG_VIAGEM

WHERE NUM = A.NUM) OR A.NUM NOT IN (SELECT NUM FROM REG_VIAGEM)
```

b)

```
SELECT DISTINCT (A.NUM)

FROM PASSE A

LEFT JOIN REG_VIAGEM B

ON A.NUM=B.NUM OR A.NUM NOT IN (SELECT NUM FROM REG_VIAGEM)
```

c)

```
SELECT A.NUM
```

```
FROM PASSE A, REG_VIAGEM B

WHERE A.NUM_VIAGENS = (SELECT COUNT(*) FROM REG_VIAGEM

WHERE NUM = A.NUM) OR A.NUM != B.NUM
```

d)

```
SELECT A.NUM

FROM PASSE A

INTERSECT

SELECT A.NUM

FROM PASSE A

WHERE A.NUM NOT IN (SELECT NUM FROM REG_VIAGEM)
```

e)

```
SELECT NUM

FROM PASSE

EXCEPT

SELECT NUM

FROM REG_VIAGEM
```

Comentário:
A consulta SQL que permite obter os números dos passes que nunca foram usados, juntamente com os números dos passes que já esgotaram o número de viagens realizadas é:

```
SELECT NUM
FROM PASSE
WHERE NUM_VIAGENS = 0 OR NUM NOT IN (SELECT NUM FROM REG_VIAGEM);
```

WHERE NUM_VIAGENS = 0: Isso inclui os passes que nunca foram usados (com número de viagens igual a zero).

OR NUM NOT IN (SELECT NUM FROM REG_VIAGEM): Isso inclui os passes que não estão presentes na tabela REG_VIAGEM, ou seja, aqueles que nunca foram usados, pois não têm correspondência na tabela de registro de viagens.

Portanto, a opção correta é aquela que usa OR para considerar ambos os casos, e a consulta acima reflete essa lógica. Assim, a resposta correta é a opção:

a) SELECT A.NUM
FROM PASSE A
WHERE A.NUM_VIAGENS = (SELECT COUNT(*) FROM REG_VIAGEM
WHERE NUM = A.NUM) OR A.NUM NOT IN (SELECT NUM FROM REG_VIAGEM)

Gabarito: A

23. CESGRANRIO - PNS (ELETRONUCLEAR)/ELETRONUCLEAR/Analista de Sistemas/Aplicações e Segurança de TIC/2022

O controle diário da utilização de passes de metrô em uma cidade é feito por programas que utilizam um banco de dados composto pelas seguintes tabelas:

CREATE TABLE PASSE (
NUM INTEGER NOT NULL,
DATA_EXP DATE NOT NULL,
NUM_VIAGENS INTEGER NOT NULL,
PRIMARY KEY (NUM))

CREATE TABLE REG_VIAGEM (
NUM INTEGER NOT NULL,
NUM_ROLETA INTEGER NOT NULL,
DATA_VIAGEM DATE NOT NULL,
FOREIGN KEY (NUM)
REFERENCES PASSE (NUM))

A tabela PASSE contém uma linha para cada passe vendido pela empresa que administra o metrô. A coluna DATA_EXP informa a data de emissão do passe, e a coluna NUM_VIAGENS informa o número de viagens em que o passe poderá ser usado (número máximo de viagens). Este número não sofre alteração ao longo do tempo.

A tabela REG_VIAGEM contém uma linha para cada viagem em que o passe foi usado. A coluna NUM_ROLETA informa a roleta na qual o passe foi inserido, e a coluna DATA_VIAGEM informa a data em que o usuário inseriu o passe na roleta.

Qual comando SQL NÃO produzirá um erro de tempo de execução independentemente dos dados existentes nas duas tabelas que fazem parte do banco de dados?
a) INSERT INTO REG_VIAGEM VALUES (7777, 720, '2022-04-21')
b) UPDATE PASSE SET NUM = 8888 WHERE NUM = 4444
c) DELETE FROM REG_VIAGEM WHERE DATA_VIAGEM = '2021-09-02';
d) INSERT INTO PASSE (NUM_VIAGENS, NUM) VALUES (10, 9999)
e) DELETE FROM PASSE WHERE DATA_EXPIR = '2021-08-11'

Comentário:
Vejamos os erros de de cada alternativa:

a) Para adicionar valor em NUM na tabela onde ele é uma chave estrangeira(foreign key), antes você precisa adicionar o valor na tabela Passe onde NUM é chave primária(primary key). A verdade é que existem cenários possíveis em que esse comando não irá executar corretamente.
b) Se você alterar o valor da PK, isso afetará a integridade referencial entre as tabelas, pois outras tabelas utilizam essa chave como FK. Além disso, se o valor for referenciado pela outra tabela e não existir uma função referencial engatilhada a alteração pode gerar um erro.
c) Deleta se existir o valor, mas se não existir não causa nenhum erro. Logo, essa é a nossa resposta.
d) DATA_EXP não pode ser nula.

Gabarito: C

24. CESGRANRIO - Tec Cien (BASA)/BASA/Tecnologia da Informação/2021

Um programador experiente estava revisando o código de um estagiário e detectou a instrução SQL abaixo, complicada demais para seu objetivo.

SELECT * FROM Compras where codProduto not in (select codProduto from Produtos where codProduto<3 or valor<4000)

Para simplificar o código, sem alterar a resposta, a instrução apresentada acima pode ser substituída por
a) SELECT * FROM Compras where codProduto<3 or valor<4000
b) SELECT * FROM Compras where codProduto<3 and valor<4000
c) SELECT * FROM Compras where codProduto<=3 and valor<=4000
d) SELECT * FROM Compras where codProduto>=3 and valor>=4000
e) SELECT * FROM Compras where codProduto>=3 or valor>=4000

Comentário:
Para resolver essa questão basta usar a teoria dos conjuntos. Os elementos que não estão (not in) nas restrições codPodruto<3 ou valor <4000 são aqueles que tem codPoduto >= 3 e valor>=4000. Desta forma, a nossa resposta encontra-se na alternativa D.

Gabarito: D

25. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2021
Para gerar um gráfico de dispersão, um programador precisava consultar duas tabelas, T1 e T2. Ele decidiu, então, usar um LEFT JOIN, como em

SELECT * FROM T1 LEFT JOIN T2 USING (CHAVE);

Essa consulta resultou em 214 linhas.
Por motivos de segurança, ele fez outra consulta semelhante, apenas trocando o LEFT JOIN por um JOIN, e essa segunda consulta resultou em 190 linhas.

O que pode explicar corretamente a quantidade diferente de linhas nas consultas realizadas?

a) CHAVE é a chave primária de T1, mas apenas um campo da chave primária de T2.
b) CHAVE é a chave primária de T2, mas apenas um campo da chave primária de T1.
c) T1 possui linhas cujo valor de CHAVE não está presente na T2.
d) T2 possui linhas cujo valor de CHAVE não está presente na T1.
e) T2 possui linhas com todas as chaves presentes em T1, mas com campos nulos.

Comentário:
A consulta original usa um LEFT JOIN com a condição USING (CHAVE). Nesse contexto:

SELECT * FROM T1 LEFT JOIN T2 USING (CHAVE);

Isso significa que a junção será feita com base no campo CHAVE, e as linhas de T1 serão incluídas independentemente da existência de uma correspondência em T2. Se não houver correspondência em T2, os campos de T2 para essa linha terão valores nulos. Portanto, o resultado pode incluir todas as linhas de T1, mesmo que não haja correspondência em T2.

Já na segunda consulta:

SELECT * FROM T1 JOIN T2 USING (CHAVE);

Aqui, um JOIN padrão (INNER JOIN) é usado, o que significa que apenas as linhas que têm correspondência nas duas tabelas serão incluídas. Se não houver uma correspondência para uma linha em T1 ou T2, essa linha será excluída do resultado.
Portanto, a explicação mais provável para a diferença de linhas entre as duas consultas é que há linhas em T1 que não têm correspondência em T2, resultando em um maior número de linhas no LEFT JOIN em comparação com o JOIN. A opção correta é:

c) T1 possui linhas cujo valor de CHAVE não está presente na T2.

Gabarito: C

26. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2021
Ao coletar dados em um sistema compatível com SQL 2008 para fazer uma análise de dados, um programador percebeu que havia dois campos, data_de_nascimento e data_de_emissão_RG, em que o valor de data_de_emissão_RG sempre deve ser mais recente que data_de_nascimento. Percebeu, porém, que em 10% das linhas acontecia o inverso, isto é, data_de_nascimento era mais recente que data_de_emissão_RG. Ele corrigiu os dados nessas linhas, verificando que estavam consistentemente trocados, mas, preocupado que tal problema voltasse a acontecer, resolveu solicitar ao DBA uma alteração da tabela, de forma que data_de_emissão_RG sempre tivesse que ser mais recente que data_de_nascimento.

O DBA atendeu adequadamente a esse pedido do programador por meio de uma restrição em SQL 2008 do tipo
a) CHECK
b) INSPECT
c) TEST
d) VALIDATE
e) VERIFY

Comentário:
O DBA atendeu ao pedido do programador utilizando uma restrição do tipo CHECK. Portanto, a resposta correta é: a) CHECK.

O comando CHECK no SQL é uma restrição que permite especificar uma condição para os valores em uma coluna. Essa condição é avaliada quando novos dados são inseridos ou atualizados na tabela, garantindo que apenas dados que satisfaçam a condição especificada sejam aceitos.

No contexto do exemplo fornecido, o DBA poderia ter adicionado uma restrição CHECK à tabela para garantir que a data de emissão do RG (data_de_emissao_RG) sempre seja mais recente que a data de nascimento (data_de_nascimento). A condição na restrição CHECK seria formulada para verificar essa relação entre as duas colunas e garantir que ela seja mantida.

A sintaxe básica para adicionar uma restrição CHECK durante a criação de uma tabela seria semelhante a esta:

CREATE TABLE Exemplo (data_de_nascimento DATE, data_de_emissao_RG DATE, -- Outras colunas... CONSTRAINT data_emissao_recente CHECK (data_de_emissao_RG > data_de_nascimento));

Essa restrição garantirá que a condição especificada seja verdadeira para cada linha na tabela. Se, durante uma operação de inserção ou atualização, a condição não for satisfeita, a operação será recusada, mantendo a consistência dos dados.

Gabarito: A

27. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2021
Após um treinamento em SQL padrão 2008, compatível com ambiente MS SQL Server 2008, um escriturário do Banco Z precisou utilizar os conhecimentos adquiridos para criar uma tabela no sistema de banco de dados desse Banco. A tabela a ser criada é de fornecedores, e tem os seguintes campos: CNPJ, nome do fornecedor e país de origem. As características gerais da tabela são:

- o campo CNPJ é chave primária e contém 14 caracteres, sendo que os caracteres devem se ater aos numéricos [“0” a “9”], e o caractere zero “0” não pode ser ignorado, seja qual for a posição dele (início, meio ou fim da chave);
- o campo NOME contém 20 caracteres e aceita valor nulo;
- o campo PAIS contém 15 caracteres e não aceita valor nulo.

Nesse contexto, o comando SQL2008 que cria uma tabela com as características descritas acima é
a) CREATE TABLE Fornecedores

(CNPJ INTEGER PRIMARY KEY,

NOME VARCHAR(20) ACCEPT NULL,

PAIS VARCHAR(15) NOT NULL)

b) CREATE TABLE Fornecedores

(CNPJ CHAR(14) PRIMARY KEY,

NOME VARCHAR(20),

PAIS VARCHAR(15) NOT NULL)

c) CREATE TABLE Fornecedores

(CNPJ CHAR(14) NOT NULL,

NOME VARCHAR(20) NOT NULL,

PAIS VARCHAR(15))

d) CREATE TABLE Fornecedores

(CNPJ CHAR(14),

NOME VARCHAR(20) NOT NULL,

PAIS VARCHAR(15) NOT NULL),

PRIMARY KEY (CNPJ)

e) CREATE TABLE Fornecedores

(CNPJ INTEGER(14) NOT NULL,

NOME VARCHAR(20),

PAIS VARCHAR(15) NOT NULL),

PRIMARY KEY (CNPJ)

Comentário:
A opção correta que atende às características descritas para criar a tabela de fornecedores no MS SQL Server 2008 é:

b) CREATE TABLE Fornecedores (CNPJ CHAR(14) PRIMARY KEY, NOME VARCHAR(20), PAIS VARCHAR(15) NOT NULL)

O comando SQL apresentado é utilizado para criar uma tabela chamada "Fornecedores" no ambiente MS SQL Server 2008, seguindo as especificações fornecidas. Vamos analisar cada parte do comando:

- CREATE TABLE: Indica que estamos criando uma nova tabela.
- Fornecedores: É o nome da tabela que está sendo criada.

- CNPJ CHAR(14): Define o campo "CNPJ" como do tipo CHAR, com um comprimento fixo de 14 caracteres. Isso atende à especificação de que o CNPJ deve ter 14 caracteres.

- PRIMARY KEY: Declara que o campo "CNPJ" será a chave primária da tabela, garantindo unicidade e identificação única para cada registro na tabela.

- NOME VARCHAR(20): Define o campo "NOME" como do tipo VARCHAR, permitindo armazenar strings de até 20 caracteres. O VARCHAR é uma boa escolha para campos de texto variável.

- PAIS VARCHAR(15) NOT NULL: Define o campo "PAIS" como do tipo VARCHAR, permitindo armazenar strings de até 15 caracteres. A cláusula "NOT NULL" indica que este campo não pode ter valores nulos, atendendo à especificação.

Portanto, esse comando cria uma tabela chamada "Fornecedores" com os campos CNPJ, NOME e PAIS, conforme as características especificadas.

Gabarito: B

28. CESGRANRIO - TBN (CEF)/CEF/Tecnologia da Informação/2021

Um sistema gerenciador de banco de dados utiliza metadados, persistidos em bancos de dados, para

a) avaliar o seu próprio desempenho, considerando metas pré-estabelecidas pelo administrador do banco de dados.
b) decidir que páginas de dados sujas na sua cache precisam ser persistidas em disco.
c) implementar o isolamento entre transações concorrentes.
d) permitir a restauração de um estado íntegro do banco de dados, em caso de falha.
e) validar comandos SQL informados por um usuário.

Comentário:

A opção correta é: e) Validar comandos SQL informados por um usuário. Os metadados também são utilizados para validar comandos SQL, garantindo que as consultas e operações realizadas pelos usuários estejam de acordo com as restrições e a estrutura do banco de dados.

Gabarito: E

29. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação

Considere as Tabelas a seguir para responder a questão.

Essas Tabelas fazem parte do esquema de um banco de dados usado por uma associação de criadores de cães para organizar informações sobre os torneios que ela promove.

```
CREATE TABLE CAO (
   COD           NUMBER(5)     NOT NULL,
   NOME          VARCHAR2(50)  NOT NULL,
   RACA          VARCHAR2(50)  NOT NULL,
   NOME_PAI      VARCHAR2(50),
   NOME_PROPR    VARCHAR2(50)  NOT NULL,
   CONSTRAINT CAO_PK PRIMARY KEY (COD)
)
CREATE TABLE COMPETICAO (
   COD           NUMBER(5)     NOT NULL,
   DESCR         VARCHAR2(50)  NOT NULL,
   CONSTRAINT COMPETICAO_PK PRIMARY KEY (COD)
)
CREATE TABLE ARBITRO (
   COD           NUMBER(5)     NOT NULL,
   NOME          VARCHAR2(50)  NOT NULL,
   CONSTRAINT ARBITRO_PK PRIMARY KEY (COD)
)
CREATE TABLE PARTICIPACAO (
   COD_CAO       NUMBER(5)      NOT NULL,
   COD_COMP      NUMBER(5)      NOT NULL,
   COLOCACAO     NUMBER(4)      NOT NULL,
   CONSTRAINT PARTICIPACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP),
   CONSTRAINT PARTICIPACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT PARTICIPACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD)
)
CREATE TABLE AVALIACAO (
   COD_CAO       NUMBER(5)      NOT NULL,
   COD_COMP      NUMBER(5)      NOT NULL,
   COD_ARBTR     NUMBER(5)      NOT NULL,
   NOTA_ARBTR    NUMBER(3,1)    NOT NULL,
   CONSTRAINT AVALIACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP,COD_ARBTR),
   CONSTRAINT AVALIACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT AVALIACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD),
   CONSTRAINT AVALIACAO_FK3 FOREIGN KEY (COD_ARBTR)
      REFERENCES ARBITRO (COD)
)
```

Observações:

- A Tabela CAO contém os dados dos cães inscritos na referida associação. A coluna NOME_PAI indica o nome do pai de um cão, a coluna RACA indica a raça do mesmo, e a coluna NOME_PROPR indica o nome do seu proprietário. As demais colunas são autoexplicativas.
- A Tabela COMPETICAO contém informações sobre as competições patrocinadas pela associação.

Suas colunas são autoexplicativas.

- A Tabela PARTICIPACAO informa as competições das quais participaram os cães registrados na associação. Cada linha dessa tabela indica a colocação obtida por um cão em uma determinada competição. Suas colunas são autoexplicativas.
- A Tabela ARBITRO contém os dados dos árbitros que julgam os cães que participam de competições. Suas colunas são autoexplicativas.
- Cada linha da Tabela AVALIACAO representa a nota atribuída a um cão, por um determinado árbitro em uma determinada competição. Suas colunas são autoexplicativas.

As Figuras a seguir exibem os dados presentes, em determinado instante, nas Tabelas CAO, COMPETICAO e PARTICIPACAO.

TABELA CAO

COD	NOME	RACA	NOME_PAI	NOME_PROPR
1111	GINGER	SETTER		LUANA ALBUQUERQUE
3333	BIBA	POINTER	RALPH	JOAO MARTINS
6666	BETINA	SETTER	BETINO	TELMA AGUIAR
4444	QUIM	PASTOR ALEMAO		MARIA IDALINA
5555	TAINA	PASTOR ALEMAO		PEDRO ALMEIDA
7777	JANIS	COCKER	TED	ANA PAULA PINTO
2222	JUDY	POINTER		JOSE MARTINS
8888	VIVI	SHITZU		FERNANDA MATHIAS

TABELA COMPETICAO

COD	DESCR
2222	TERESÓPOLIS OPEN
3333	TORNEIO DE FRIBURGO
1111	ABERTO DO RIO

TABELA PARTICIPACAO

COD_CAO	COD_COMP	COLOCACAO
4444	1111	4
8888	2222	1
3333	2222	2
2222	2222	3
1111	1111	1
2222	1111	2
3333	1111	3
5555	1111	5
1111	2222	5
4444	2222	3

Qual comando irá inserir uma nova linha no banco de dados em questão?

a) INSERT INTO COMPETICAO VALUES ('MOSTRA COMPETITIVA DE BRASILIA',9595)

b) INSERT INTO PARTICIPACAO VALUES(8877,1111,6)

c) INSERT INTO PARTICIPACAO (COLOCACAO, COD_COMP, COD_CAO) VALUES (1,1111,8888)

d) INSERT INTO CAO (COD, NOME, RACA, NOME_PAI) VALUES (1130,'TUTU','BULDOG FRANCES','PEPEU')

e) INSERT INTO PARTICIPACAO VALUES (2222,1111,7)

Comentário:

Vamos analisar as alternativas acima. A primeira, letra a), gera um erro na inserção pois o tamanho do campo DESCR da tabela competição é de no máximo 20 caracteres (VARCHAR (20)) e o valor passado como parâmetro possui 30 caracteres. Já a alterativa b) comete um equívoco na integridade referencial. O código do cão 8877 não existe na tabela CAO.

A alternativa c é a nossa resposta. Veja que a ordem dos parâmetros da clausula VALUES só devem seguir a mesma ordem da declaração do comando CREATE se não for definida a lista de atributos após o nome da tabela no comando INSERT. Vejam que, neste caso, o examinador usou (COLOCACAO, COD_COMP, COD_CAO) e seguiu essa mesma ordem na clausula VALUES. Logo, a alternativa está perfeita do ponto de vista sintático e semântico.

A alternativa d) tem um erro, pois o nome do proprietário não pode ser nulo. Da forma como está definido o comando, sem a associação de um valor para tal atributo e ainda sem um valor DEFAULT definido no comando CREATE TABLE essa inserção não será executada.

Já a alternativa e) está errada pois já existe um valor de chave primária igual ao que estamos tentando inserir na tabela PARTICIPACAO. Logo, pela restrição de integridade de chave esse comando vai gerar um erro e não será executado a contento. Desta forma, podemos marcar nossa resposta na alternativa c.

Gabarito: C

30. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Exploração de Petróleo Júnior - Informática

ALUNO (cpf : string , nome : string , endereco : string, telefone : string) MATRICULA (cpf : string , cod-cad : string)

CADEIRA (cod-cad : string , nome : string , creditos : number) Que comando SQL cria tabela ALUNO?

a) CREATE TABLE ALUNO (cpf string , nome string , endereco string , telefone string) ;

b) CREATE TABLE ALUNO (cpf : string , nome : string , endereco : string , telefone : string) ;

c) CREATE TABLE ALUNO (cpf string PK, nome string , endereco string , telefone string) ;

d) CREATE ALUNO AS TABLE (cpf : string PK, nome : string , endereco : string , telefone : string) ;

e) CREATE ALUNO AS TABLE (cpf string PK , nome string , endereco string , telefone string) ;

Comentário:

Essa é uma questão de sintaxe. O comando CREATE TABLE descreve primeiramente o nome da tabela. Em seguida, entre parênteses e separados por vírgulas, apresenta o nome e o tipo dos atributos. Depois, ainda dentro dos parênteses são incluídas as restrições de integridade. Estas podem aparecer ao lado da definição do atributo ou ao final. Neste caso se estivermos falando de uma chave primária composta temos que declara a CONSTRAINT após a declaração dos atributos. Vejam que o erro das alternativas (a, b, d, e) está em incluir elementos que não fazem parte da sintaxe do comando. Na letra "a" falta a PK, nas letras "b" e "d" temos um ":" desnecessário, e na "e" temos ":" e um "AS". Logo, podemos marcar a resposta correta na alternativa C.

Gabarito: C

31. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Exploração de Petróleo Júnior - Informática

ALUNO (cpf : string , nome : string , endereco : string, telefone : string) MATRICULA (cpf : string , cod-cad : string)

CADEIRA (cod-cad : string , nome : string , creditos : number)

Qual o comando SQL para alterar o nome do aluno com CPF 512.859.850-01 para "Jose da Silva"?

a) ALTER RECORD ALUNO SET nome='Jose da Silva' WHERE cpf='512.859.850-01'

b) INSERT INTO ALUNO nome='Jose da Silva' AND cpf='512.859.850-01'

c) UPDATE ALUNO WHERE nome='Jose da Silva' AND cpf='512.859.850-01'

d) UPDATE ALUNO SET nome='Jose da Silva' WHERE cpf='512.859.850-01'

e) INSERT INTO ALUNO nome='Jose da Silva' WHERE cpf='512.859.850-01'

Comentário:

Para alterara valores de uma tupla ou linha da tabela usamos o comando UPDATE. A sintaxe do comando é a seguinte:

UPDATE nome_tabela SET nome_atributo = "novo_valor" WHERE CONDIÇÃO;.

Assim, ao analisarmos as alternativas, podemos concluir que a nossa resposta se encontra letra d. Vejas que as demais alternativas pecam por usar a sintaxe errada a linguagem para alteração de registros.

Gabarito: D

32. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Exploração de Petróleo Júnior - Informática

ALUNO (cpf : string , nome : string , endereco : string, telefone : string) MATRICULA (cpf : string , cod-cad : string)

CADEIRA (cod-cad : string , nome : string , creditos : number)

Qual o comando SQL que obtém apenas os nomes de todos os alunos?

a) SELECT * FROM ALUNO WHERE nome IS STRING
b) SELECT nome FROM ALUNO
c) LIST * FROM ALUNO
d) SELECT nome WHERE ALUNO
e) LIST nome FROM ALUNO

Comentário:

Essa questão também é relativamente tranquila. Veja que ela pede o nome de todos os alunos. Assim, precisamos apenas construir um SELECT sobre a tabela ALUNO sem restrição na clausula WHERE, e listando apenas a coluna nome na clausula SELECT. Assim, podemos observar nossa resposta na alternativa B.

Gabarito: B

33. Ano: 2014 Banca: CESGRANRIO Órgão: PetrobrasProva: Engenheiro(a) de Equipamentos Júnior

- Eletrônica

Observe a Tabela FERRAMENTAS abaixo, relativa a um banco de dados relacional.

	CODIGO	ITEM	PRECO	DESCRICAO
tupla 1 →	F-1542	alicate de pressao	23,99	medio
tupla 2 →	F-1543	alicate comum	14,11	ponta fina
tupla 3 →	F-1544	alicate de corte	15,70	pequeno
tupla 4 →	F-2376	chave de fenda	5,76	comum
tupla 5 →	F-2378	chave de fenda	8,20	phillips

A execução do comando SQL SELECT * FROM FERRAMENTAS WHERE ((ITEM = 'chave de fenda' OR ITEM = 'chave de teste') AND PRECO < 9) produzirá como resposta, respectivamente, as tuplas de números

a) 2, 6 e 7

b) 4, 5 e 8

c) 3, 7 e 9

d) 1, 2, 3 e 9

e) 4, 5, 6 e 8

Comentário:

Veja que temos algumas restrições feitas na clausula WHERE que por conta do conectivo AND devem ser aplicadas em conjunto sobre as tuplas da relação. Se aplicarmos a primeira restrição, selecionando apenas os itens chave de fenda e chave inglesa, ficamos com as tuplas 4, 5, 6, 7 e 8. Agora vamos restringir esse subconjunto as tuplas cujo valos do preço é menor do que 9. Ficamos, portanto, com as tuplas 4, 5 e 8. Assim, podemos marcar nossa resposta na alternativa B.

Gabarito: B

34. Ano: 2014 Banca: CESGRANRIO Órgão: CEFET-RJ Prova: Técnico de Tecnologia da Informação

Que comando SQL insere uma linha na Tabela Aluno, com idAluno=1 e nomeAluno="Aline"?

a) INSERT INTO Aluno SET nomeAluno="Aline" WHERE idAluno=1

b) INSERT INTO Aluno (idAluno, nomeAluno) VALUES (1,"Aline")

c) INSERT INTO Aluno SET nomeAluno="Aline" AND idAluno=1

d) UPDATE Aluno SET nomeAluno="Aline" WHERE idAluno=1

e) UPDATE Aluno(idAluno, nomeAluno) SET VALUES (1,"Aline")

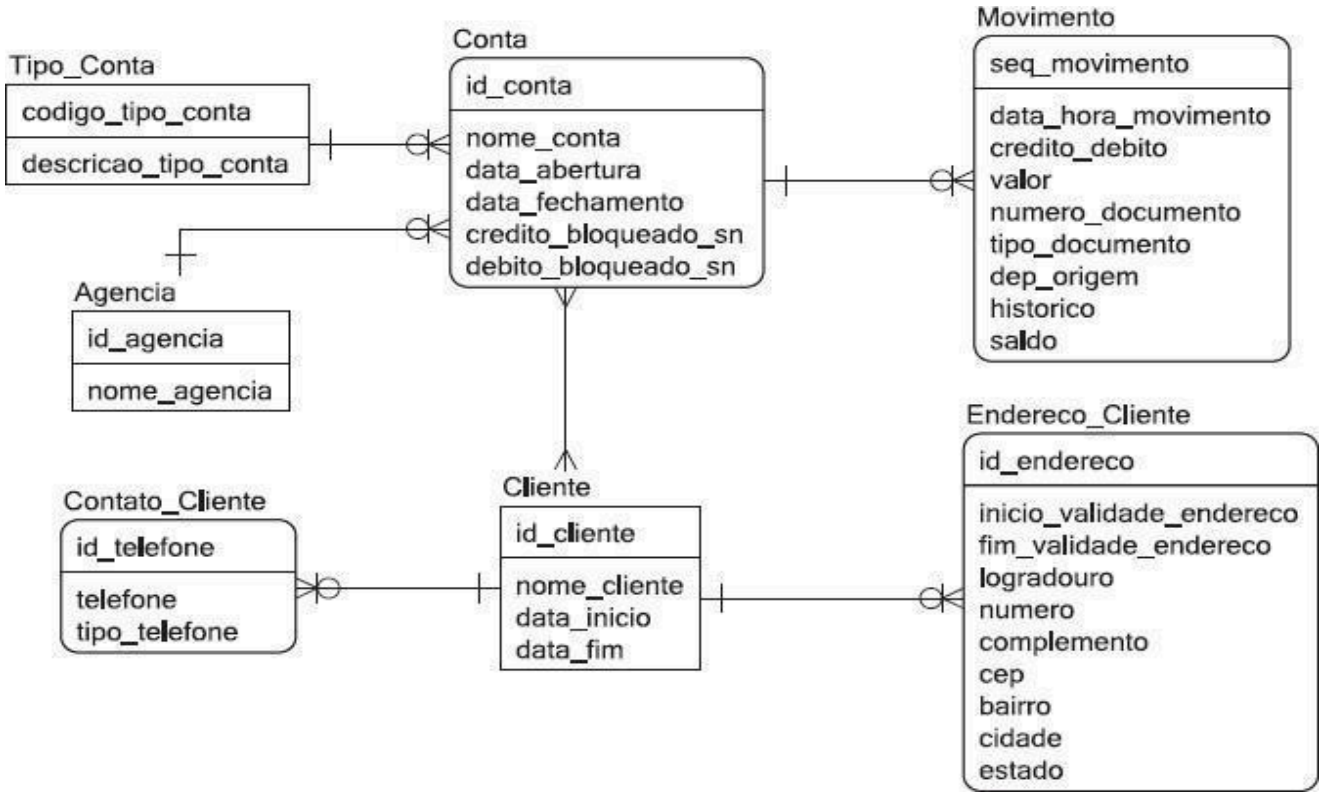
Comentário:

Mas uma questão que trata do comando INSERT. Lembra-se que a sintaxe correta é INSERT INTO nome_tabela [(nome_coluna, ...)] VALUES (parâmetro1,...). Assim, podemos encontrar a sintaxe correta apenas na alternativa B.

Gabarito: B

35. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Banco de Dados

Para responder à questão, tenha como referência o diagrama de entidades e relacionamentos, apresentado abaixo, que representa parte do modelo de dados de uma instituição financeira.



Que comando SQL deve ser dado para criar a Tabela Tipo_Conta?

a) CREATE TABLE Tipo_Conta (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

b) CREATE TABLE Tipo_Conta (codigo_tipo_conta:NUMERIC PRIMARY KEY, descricao_tipo_conta:VARCHAR(256))

c) CREATE TABLE Tipo_Conta WITH COLUMNS (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

d) CREATE Tipo_Conta AS TABLE (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

e) CREATE Tipo_Conta AS TABLE WITH COLUMNS (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

Comentário:

Ao analisar a notação da figura podemos observar alguns pontos importante para a tabela TIPO_CONTA. Primeiramente cada conta está relacionada com um tipo conta, mais um tipo conta pode ser associado a várias contas. Logo, a chave de conta será chave estrangeira na tabela CONTA. Segundo, o CÓDIGO_TIPO_CONTA será a chave primária da tabela por estar localizado na parte superior do diagrama e será do tipo numérico. Assim, podemos construir

nosso comando DDL de criação de tabela com o seguinte comando:

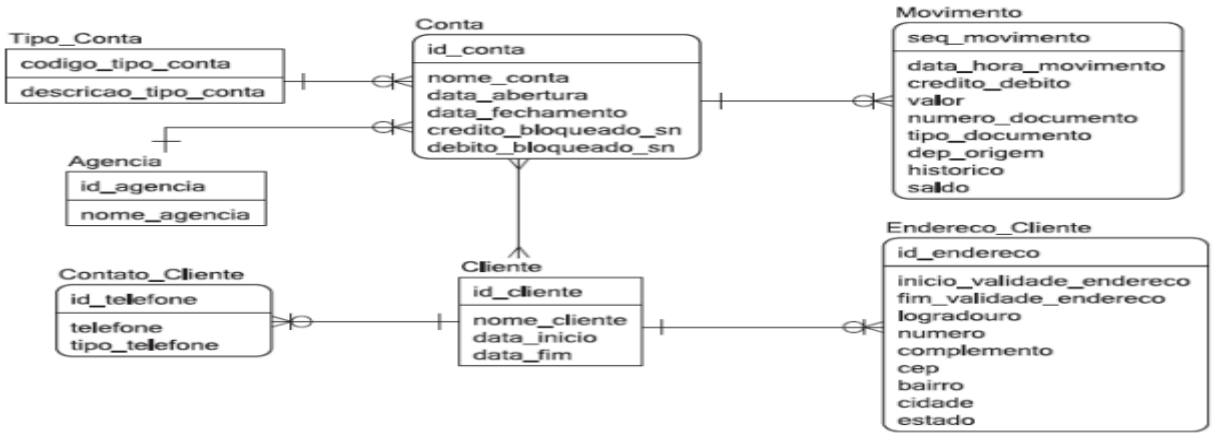
CREATE TABLE Tipo_Conta (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR (256));

Tal comando está presente na alternativa a) que é a nossa resposta.

Gabarito: A

36. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Banco de Dados

Para responder à questão, tenha como referência o diagrama de entidades e relacionamentos,



apresentado abaixo, que representa parte do modelo de dados de uma instituição financeira.

Que comando SQL deve ser dado para bloquear o crédito da conta 123456, colocando "S" no campo credito_bloqueado_sn?

a) UPDATE Conta SET credito_bloqueado_sn="S" SELECT * FROM Conta WHERE id_conta=123456

b) UPDATE Conta SET credito_bloqueado_sn="S" WHERE id_conta=123456

c) UPDATE Conta SET VALUES (id_conta,"S") WHERE id_conta=123456

d) UPDATE credito_bloqueado_sn="S" From Conta WHERE id_conta=123456

e) UPDATE INTO Conta VALUES (123456,"S")

Comentário:

Mas uma vez, assim que existirem dados na tabela, podemos chegar à conclusão que é necessário modificar os dados. Par tal, podemos utilizar o comando UPDATE. A sintaxe para tal é:

UPDATE "nome_tabela"

SET "coluna 1" = [novo valor]

WHERE "condição";

Gabarito: B

37. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Banco de Dados

Considere um banco de dados relacional com as duas tabelas a seguir. Empregado

(emp_id, emp_nome, dno, salario)

Departamento (dep_id, dep_mome)

O campo Empregado.dno indica o dep_id do departamento onde o empregado trabalha, e os campos sublinhados são chave primária.

Nesse contexto, analise o seguinte comando SQL:

```
SELECT  d.dep_nome, COUNT(*) AS x
FROM    Departamento d, Empregado e
WHERE  d.dep_id = e.dno AND e.salario > 5000 AND
e.dno IN (SELECT f.dno FROM  Empregado f GROUP BY
f.dno HAVING COUNT(*) > 2)
GROUP BY  d.dep_nome;
```

O que calcula o comando SQL apresentado acima?

a) Quantos empregados ganham mais de R$ 5.000,00, listados por departamento.

b) Quantos empregados existem, listados por departamento, em departamentos com mais de duas pessoas que ganham mais de R$ 5.000,00.

c) Quantos empregados existem, listados por departamento, em departamentos que possuem duas pessoas que ganham mais de R$ 5.000,00.

d) Quantos empregados ganham mais de R$ 5.000,00, listados por departamento, em departamentos com mais de duas pessoas.

e) Quantos departamentos existem com mais de duas pessoas que ganham R$ 5.000,00.

Comentário:

Essa questão é bem interessante precisa ser analisada com calma. Vejam que na clausula SELECT temos uma função agregada que vai contar alguma coisa por departamento. Outro ponto é que a consulta interna retorna os números de departamento com mais de 2 funcionários. Após o produto cartesiano entre departamento e empregado e aplicada a clausula de junção (d.dep_id=e.dno), vamos selecionar as linhas onde o salário do empregado é maior que 5000 em departamentos com mais de duas pessoas. Desta forma a nossa resposta está na alternativa D.

Gabarito: D.

38. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Analise de Sistemas

As Tabelas pertencem ao esquema de um banco de dados acadêmico de uma universidade.

```
CREATE TABLE ALUNO (
    MATRIC       NUMBER(5)      NOT NULL,
    NOME         VARCHAR2(30)   NOT NULL,
    CPF          NUMBER(11),
    CONSTRAINT   ALUNO_UK1 UNIQUE (CPF),
    CONSTRAINT   ALUNO_PK PRIMARY KEY (MATRIC))

CREATE TABLE HISTORICO (
    MATRIC      NUMBER(5)     NOT NULL,
    COD_DISC    CHAR(7)       NOT NULL,
    NOME_DISC   VARCHAR2(30)  NOT NULL,
    ANO         NUMBER(4)     NOT NULL,
    SEMESTRE    NUMBER(1)     NOT NULL,
    NOTA        NUMBER(3,1)   NOT NULL,
    CONSTRAINT  HIST_PK
        PRIMARY KEY (MATRIC,COD_DISC,ANO,SEMESTRE),
    CONSTRAINT  HIST_FK  FOREIGN KEY (MATRIC)
        REFERENCES ALUNO (MATRIC))
```

Observações:

- Um aluno pode cursar uma disciplina várias vezes, desde que em diferentes ANO/SEMESTRE.
- Um aluno será reprovado em uma disciplina caso sua nota seja inferior a 5,0.
- O estado corrente do banco de dados dessa universidade é mostrado a seguir.

Tabela ALUNO

MATRIC	NOME	CPF
33333	FERNANDA MARTINS	null
44444	GABRIEL LOPES	null
11111	ANA MARIA	12344456789
22222	FLÁVIA FERNANDES	23565677789
55555	LIVIA LEVY	23576712145

Tabela HISTORICO

MATRIC	COD_DISC	NOME_DISC	ANO	SEMESTRE	NOTA
11111	INF1009	LÓGICA	2012	1	6
33333	INF1009	LÓGICA	2013	1	8
33333	INF1010	ESTR DE DADOS	2012	2	4
33333	INF1010	ESTR DE DADOS	2013	1	4
33333	INF1010	ESTR DE DADOS	2013	2	6
55555	INF1009	LÓGICA	2013	1	4
55555	INF1009	LÓGICA	2013	2	6
55555	INF1010	ESTR DE DADOS	2013	2	8
11111	INF1010	ESTR DE DADOS	2013	2	7
33333	INF1001	PROGRAMAÇÃO II	2012	2	4
55555	INF1001	PROGRAMAÇÃO II	2012	2	6
33333	INF1001	PROGRAMAÇÃO II	2013	1	7

Qual comando irá modificar o estado corrente da Tabela ALUNO?

a) INSERT INTO ALUNO (MATRIC, CPF, NOME) VALUES (66666,'TIAGO MENEZES')

b) DELETE FROM ALUNO WHERE CPF IS NULL

c) UPDATE ALUNO SET CPF=23565677789 WHERE NOME ='GABRIEL LOPES'

d) INSERT INTO ALUNO VALUES (66666,'TIAGO MENEZES')

e) DELETE FROM ALUNO A WHERE NOT EXISTS (SELECT * FROM HISTORICO WHERE MATRIC=A.MATRIC)

Comentário:

Esse tipo de questão já apareceu outras vezes nesta lista. Lembre-se, não é porque o comando tem uma estrutura sintática correta que ele vai ser executado com sucesso. É possível que ele viole alguma restrição de integridade, portanto, muito cuidado. Vejamos agora cada uma das alternativas.

Na alternativa A temos um erro de sintaxe, o atributo CPF é passado entre parênteses e não aparece na clausula VALUES. Logo, a linha não será inserida. Já na letra B, temos um problema que a deleção de linhas nas quais o valor de CPF é nulo. Quando você remover a linha, a integridade referencial será quebrada, pois não existiram mais as matrículas 33333 e 4444.

A alternativa C está errada porque CPF é um atributo UNIQUE e já existe uma pessoa com o CPF que iria ser atualizado ao Gabriel Lopes.

A letra E vai remover o aluno que não está matriculado em nenhuma matéria. No caso os alunos FLÁVIA FERNANDES e GABRIEL LOPES, sendo está a nossa resposta.

Gabarito: E.

39. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Analise de Sistemas

As Tabelas pertencem ao esquema de um banco de dados acadêmico de uma universidade.

```
CREATE TABLE ALUNO (
    MATRIC      NUMBER(5)     NOT NULL,
    NOME        VARCHAR2(30)  NOT NULL,
    CPF         NUMBER(11),
    CONSTRAINT  ALUNO_UK1 UNIQUE (CPF),
    CONSTRAINT  ALUNO_PK PRIMARY KEY (MATRIC))

CREATE TABLE HISTORICO (
    MATRIC     NUMBER(5)     NOT NULL,
    COD_DISC   CHAR(7)       NOT NULL,
    NOME_DISC  VARCHAR2(30)  NOT NULL,
    ANO        NUMBER(4)     NOT NULL,
    SEMESTRE   NUMBER(1)     NOT NULL,
    NOTA       NUMBER(3,1)   NOT NULL,
    CONSTRAINT  HIST_PK
        PRIMARY KEY (MATRIC,COD_DISC,ANO,SEMESTRE),
    CONSTRAINT  HIST_FK  FOREIGN KEY (MATRIC)
        REFERENCES ALUNO (MATRIC))
```

Observações:

- Um aluno pode cursar uma disciplina várias vezes, desde que em diferentes ANO/SEMESTRE.
- Um aluno será reprovado em uma disciplina caso sua nota seja inferior a 5,0.
- O estado corrente do banco de dados dessa universidade é mostrado a seguir.

Tabela ALUNO

MATRIC	NOME	CPF
33333	FERNANDA MARTINS	null
44444	GABRIEL LOPES	null
11111	ANA MARIA	12344456789
22222	FLÁVIA FERNANDES	23565677789
55555	LIVIA LEVY	23576712145

Tabela HISTORICO

MATRIC	COD_DISC	NOME_DISC	ANO	SEMESTRE	NOTA
11111	INF1009	LÓGICA	2012	1	6
33333	INF1009	LÓGICA	2013	1	8
33333	INF1010	ESTR DE DADOS	2012	2	4
33333	INF1010	ESTR DE DADOS	2013	1	4
33333	INF1010	ESTR DE DADOS	2013	2	6
55555	INF1009	LÓGICA	2013	1	4
55555	INF1009	LÓGICA	2013	2	6
55555	INF1010	ESTR DE DADOS	2013	2	8
11111	INF1010	ESTR DE DADOS	2013	2	7
33333	INF1001	PROGRAMAÇÃO II	2012	2	4
55555	INF1001	PROGRAMAÇÃO II	2012	2	6
33333	INF1001	PROGRAMAÇÃO II	2013	1	7

A execução de uma consulta SQL sobre o banco de dados dessa universidade produziu o seguinte resultado:

NOME	AVG(NOTA)
LIVIA LEVY	6
FLÁVIA FERNANDES	null
ANA MARIA	7
FERNANDA MARTINS	5,5
GABRIEL LOPES	null

Que consulta é essa?

a) SELECT A.NOME, AVG(H.NOTA) FROM ALUNO A
RIGHT OUTER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.NOME

b) SELECT A.NOME, AVG(H.NOTA)
FROM ALUNO A
INNER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.NOME

c) SELECT A.NOME, AVG(H.NOTA) FROM ALUNO A, HISTORICO H WHERE A.MATRIC=H.MATRIC GROUP BY A.NOME

d) SELECT A.NOME, AVG(H.NOTA) FROM ALUNO A
LEFT OUTER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.NOME

e) SELECT A.NOME, AVG(H.NOTA) FROM ALUNO A
RIGHT OUTER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.MATRIC

Comentário:
Primeiramente precisamos observar que para unir o nome as notas precisamos fazer uma junção. Como vamos ficar com todos os valores da tabela da esquerda vamos fazer um LEFT OUTER JOIN entre as tabelas aluno e histórico. A função agregada calcula a média das notas. Por fim, não podemos esquecer da clausula de agrupamento (GROUP BY) e do atributo de junção. Desta forma, nossa resposta encontra-se na alternativa D.

Gabarito: D

40. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Analise de Sistemas

As Tabelas pertencem ao esquema de um banco de dados acadêmico de uma universidade.

```
CREATE TABLE ALUNO (
    MATRIC      NUMBER(5)      NOT NULL,
    NOME        VARCHAR2(30)   NOT NULL,
    CPF         NUMBER(11),
    CONSTRAINT  ALUNO_UK1 UNIQUE (CPF),
    CONSTRAINT  ALUNO_PK PRIMARY KEY (MATRIC))

CREATE TABLE HISTORICO (
    MATRIC      NUMBER(5)     NOT NULL,
    COD_DISC    CHAR(7)       NOT NULL,
    NOME_DISC   VARCHAR2(30)  NOT NULL,
    ANO         NUMBER(4)     NOT NULL,
    SEMESTRE    NUMBER(1)     NOT NULL,
    NOTA        NUMBER(3,1)   NOT NULL,
    CONSTRAINT  HIST_PK
        PRIMARY KEY (MATRIC,COD_DISC,ANO,SEMESTRE),
    CONSTRAINT  HIST_FK  FOREIGN KEY (MATRIC)
        REFERENCES ALUNO (MATRIC))
```

Observações:

- Um aluno pode cursar uma disciplina várias vezes, desde que em diferentes ANO/SEMESTRE.
- Um aluno será reprovado em uma disciplina caso sua nota seja inferior a 5,0.
- O estado corrente do banco de dados dessa universidade é mostrado a seguir.

Tabela ALUNO

MATRIC	NOME	CPF
33333	FERNANDA MARTINS	null
44444	GABRIEL LOPES	null
11111	ANA MARIA	12344456789
22222	FLÁVIA FERNANDES	23565677789
55555	LIVIA LEVY	23576712145

Tabela HISTORICO

MATRIC	COD_DISC	NOME_DISC	ANO	SEMESTRE	NOTA
11111	INF1009	LÓGICA	2012	1	6
33333	INF1009	LÓGICA	2013	1	8
33333	INF1010	ESTR DE DADOS	2012	2	4
33333	INF1010	ESTR DE DADOS	2013	1	4
33333	INF1010	ESTR DE DADOS	2013	2	6
55555	INF1009	LÓGICA	2013	1	4
55555	INF1009	LÓGICA	2013	2	6
55555	INF1010	ESTR DE DADOS	2013	2	8
11111	INF1010	ESTR DE DADOS	2013	2	7
33333	INF1001	PROGRAMAÇÃO II	2012	2	4
55555	INF1001	PROGRAMAÇÃO II	2012	2	6
33333	INF1001	PROGRAMAÇÃO II	2013	1	7

Qual consulta exibe os nomes dos alunos que nunca foram reprovados?

a) SELECT DISTINCT A.NOME FROM ALUNO A, HISTORICO H

WHERE A.MATRIC=H.MATRIC AND H.NOTA >= 5.0

b) SELECT NOME FROM ALUNO MINUS

SELECT DISTINCT A.NOME FROM ALUNO A, HISTORICO H WHERE A.MATRIC=H.MATRIC

GROUP BY A.NOME, H.NOTA HAVING H.NOTA < 5.0

c) SELECT DISTINCT A.NOME FROM ALUNO A, HISTORICO H WHERE

A.MATRIC=H.MATRIC GROUP BY A.NOME, H.NOTA HAVING H.NOTA >=5.0

d) SELECT A.NOME

FROM ALUNO A WHERE A. MATRIC IN

(SELECT MATRIC FROM HISTORICO WHERE NOTA >= 5.0)

e) SELECT DISTINCT A.NOME FROM ALUNO A

LEFT OUTER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.NOME, H.NOTA

HAVING H.NOTA >=5.0

Comentário:
Para ser reprovado você precisa tirar uma nota abaixo de 5,00, isso pode ser observado pelos valores das tuplas. Assim, para saber que nunca foi reprovado, basta subtrair do grupo de alunos aqueles que já foram reprovados. Em SQL, tal comando está descrito na alternativa B.

Gabarito: B.

41. Ano: 2014 Banca: CESGRANRIO Órgão: EPE Prova: Analista de Gestão Corporativa - Tecnologia da Informação

As informações presentes abaixo devem ser usadas para responder às questões de nos 50 a 52.

Pessoa			
Id	Nome	Idade	Sexo
1	Ana	70	F
2	Beto	65	M
3	Carlos	45	M
4	Débora	40	F
5	Ênio	22	M
6	Fabiana	20	F
7	Guto	1	M
8	Hilda	52	F

Parentesco	
PaiMae	FilhoFilha
1	3
1	4
2	3
2	4

Que comando SQL inclui a informação de que Hilda é mãe de Fabiana?

a) INSERT INTO Parentesco SELECT F.Id,P.Id FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Hilda' AND F.Nome='Fabiana '

b) INSERT INTO Parentesco SELECT P.Id,F.Id FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Hilda' AND F.Nome='Fabiana'

c) INSERT INTO Parentesco SELECT P.Id,F.Id FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Fabiana' AND F.Nome='Hilda'

d) INSERT INTO Parentesco VALUES SELECT F.Id,P.FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Hilda' AND F.Nome='Fabiana '

e) INSERT INTO Parentesco VALUES SELECT P.Id,F.Id FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Hilda' AND F.Nome='Fabiana'

Comentário:

Veja que vamos repassar o resultado da consulta(SELECT) para que possa ser inserido. Desta forma, precisamos respeitar a ordem dos valores, conforme a tabela está estruturada. Veja que primeiro precisamos informar o nome da Mãe, depois o nome da filha.

Agora a questão resolveu usar a força bruta para resolver a questão. Primeiramente o comando faz um produto cartesiano da tabela com ela mesma, assim, todas as combinações de nomes serão construídas. Em segunda, o comando restringe o resultado apenas a tuplas cujo nome da Mãe é Hilda e o da Filha Fabiana. Por fim, seleciona o Id de cada uma delas e insere na tabela parentesco. Isso nos leva resposta na alternativa B.

Gabarito: B

42. Ano: 2014 Banca: CESGRANRIO Órgão: EPE Prova: Analista de Gestão Corporativa - Tecnologia da Informação

As informações presentes abaixo devem ser usadas para responder às questões de nos 50 a 52.

Pessoa			
Id	Nome	Idade	Sexo
1	Ana	70	F
2	Beto	65	M
3	Carlos	45	M
4	Débora	40	F
5	Ênio	22	M
6	Fabiana	20	F
7	Guto	1	M
8	Hilda	52	F

Parentesco	
PaiMae	FilhoFilha
1	3
1	4
2	3
2	4
5	7
6	7
3	5
4	5

Que comando SQL NÃO fornecerá apenas o nome de todos os filhos de Ana?

a) SELECT F.Nome AS FF FROM (Pessoa AS P INNER JOIN Parentesco ON P.Id=PaiMae) INNER JOIN Pessoa AS F ON F.Id=FilhoFilha WHERE P.Nome='Ana'

b) SELECT F.Nome FROM Pessoa AS F WHERE F.Id IN (SELECT FilhoFilha FROM Pessoa AS P, Parentesco AS PP WHERE P.Id=PP.PaiMae AND P.Nome='Ana')

c) SELECT F.Nome FROM Pessoa AS F WHERE F.Id IN (SELECT FilhoFilha FROM Pessoa AS P INNER JOIN Parentesco AS PP ON P.Id=PP.PaiMae WHERE P.Nome='Ana')

d) SELECT F.Nome FROM Pessoa AS F WHERE F.Id IN (SELECT FilhoFilha FROM Pessoa AS P LEFT JOIN Parentesco AS PP ON P.Id=PP.PaiMae WHERE P.Nome='Ana')

e) SELECT F.Nome FROM Pessoa AS P, Pessoa AS F, Parentesco AS R WHERE P.Nome='Ana' AND F.Id=R.PaiMae AND P.Id=R.FilhoFilha

Comentário:

Essa questão eu vou deixar as alternativas que não são a resposta para que você possa perceber que em todas elas os filhos de Ana (Beto e Carlos) aparecem no resultado. Agora, observe a alternativa E. Ela faz um produto cartesiano de pessoa com ela mesma. Veja que existe a restrição do nome ser igual a Ana. Assim, o resultado intermediário será uma associação de Ana com todas as pessoas, inclusive com ela mesma. Ai depois a questão adiciona dois predicados: F.Id=R.PaiMae (o ID da filha é igual ao IF da Mãe) e P.Id=R.FilhoFilha (o ID da mãe é igual ao ID da filha). Veja que isso está invertido. Logo não pode ser nossa resposta. Veja que o ID de Ana não aparece nenhuma vez na coluna FIlhoFIlha. Logo, nosso resultado será, provavelmente, vazio.

Gabarito: E.

43. Ano: 2014 Banca: CESGRANRIO Órgão: FINEP Prova: Analista - Desenvolvimento de Sistemas

As tabelas abaixo pertencem ao esquema de um banco de dados de um supermercado. CREATE TABLE PRODUTO (
COD NUMBER(5) NOT NULL, DESCRICAO VARCHAR2(100) NOT NULL, PRECO NUMBER(8,2) NOT NULL, QTD_ESTOQUE NUMBER(5) ,
TIPO NUMBER(1) NOT NULL,

CONSTRAINT PRODUTO_PK PRIMARY KEY (COD)) CREATE TABLE ITEM (
NUM_SERIE NUMBER(7) NOT NULL, COR VARCHAR2(20) NOT NULL, VOLTAGEM NUMBER(5) NOT NULL, COD_PROD NUMBER(5) NOT NULL,
CONSTRAINT ITEM_PK PRIMARY KEY (NUM_SERIE), CONSTRAINT ITEM_FK FOREIGN KEY (COD_PROD)
REFERENCES PRODUTO (COD))

Observações:

- A empresa comercializa produtos controlados por quantidade (por exemplo, caixa de sabão em pó, pacote de biscoito e lata de extrato de tomate) e produtos controlados por

unidade (por exemplo, televisor, máquina de lavar roupa e liquidificador).

- A quantidade em estoque de um produto controlado por quantidade (TIPO=1) é obtida diretamente da coluna QTD_ESTOQUE.
- A quantidade em estoque de um produto controlado por unidade (TIPO=2) NÃO pode ser obtida diretamente da coluna QTD_ESTOQUE, pois, para esse tipo de produto, esta coluna irá conter o valor NULL.
- Cada linha da tabela ITEM contém informações sobre um item existente no estoque da empresa relativo a um tipo de produto controlado por unidade.

Qual consulta SQL irá exibir o código, a descrição e a quantidade em estoque relativos a cada um dos produtos comercializados pelo supermercado?

a) SELECT COD, DESCRICAO, QTD_ESTOQUE FROM PRODUTO
WHERE TIPO = 1 UNION
SELECT P.COD, P.DESCRICAO, COUNT(I.COD_PROD) FROM PRODUTO P,ITEM I
WHERE TIPO = 2 AND P.COD=I.COD_PROD GROUP BY P.COD, P.DESCRICAO

b) SELECT COD, DESCRICAO, QTD_ESTOQUE FROM PRODUTO
WHERE TIPO = 1 UNION
SELECT P.COD, P.DESCRICAO, COUNT(I.COD_PROD) FROM PRODUTO P
LEFT JOIN ITEM I
ON P.COD=I.COD_PROD WHERE P.TIPO = 2
GROUP BY P.COD, P.DESCRICAO

c) SELECT P.COD, P.DESCRICAO, COUNT(DISTINCT P.TIPO) FROM PRODUTO P
LEFT OUTER JOIN ITEM I ON P.COD=I.COD_PROD
GROUP BY P.COD, P.DESCRICAO

d) SELECT P.COD, P.DESCRICAO, SUM (DISTINCT P.TIPO) FROM PRODUTO P
INNER JOIN ITEM I

ON P.COD=I.COD_PROD GROUP BY P.COD, P.DESCRICAO

e) SELECT COD, DESCRICAO, QTD_ESTOQUE FROM PRODUTO
WHERE TIPO = 1

UNION
SELECT P.COD, P.DESCRICAO, COUNT(I.COD_PROD) FROM PRODUTO P
RIGHT JOIN ITEM I
ON P.COD=I.COD_PROD WHERE P.TIPO = 2
GROUP BY P.COD,P.DESCRICAO

Comentário:

A primeira coisa que temos que ter em mente é que, por termos dois tipos distintos de produtos em estoque, precisamos calcular cada um deles isoladamente e, em seguida, aplicarmos uma

operação de UNION para fundir os dois grupos resultantes da subconsultas. Para o produto do tipo 1 basta efetuarmos a seguinte consulta:

SELECT COD, DESCRICAO, QTD_ESTOQUE FROM PRODUTO

WHERE TIPO = 1

Para o produto do tipo 2 precisamos usar a tabela item que contém a descrição de todos os produtos. Vamos operar um left join para ficarmos com todos os produtos, mas restringimos o tipo ao 2. Neste caso, como estamos usando uma função agregada, vamos fazer uso também da clausula GROUP BY. Assim temos:

SELECT P.COD, P.DESCRICAO, COUNT(I.COD_PROD) FROM PRODUTO P

LEFT JOIN ITEM I

ON P.COD=I.COD_PROD WHERE P.TIPO = 2

GROUP BY P.COD, P.DESCRICAO

Se unirmos os dois SQLs acima, podemos encontrar nossa resposta na alternativa B.

Gabarito: B

44. Ano: 2014 Banca: CESGRANRIO Órgão: FINEP Prova: Analista - Desenvolvimento de Sistemas

As tabelas abaixo pertencem ao esquema de um banco de dados de um supermercado. CREATE

TABLE PRODUTO (

COD NUMBER(5) NOT NULL,

DESCRICAO VARCHAR2(100) NOT NULL,

PRECO NUMBER(8,2) NOT NULL, QTD_ESTOQUE NUMBER(5) , TIPO NUMBER(1) NOT NULL,

CONSTRAINT PRODUTO_PK PRIMARY KEY (COD))

CREATE TABLE ITEM (

NUM_SERIE NUMBER(7) NOT NULL, COR VARCHAR2(20) NOT NULL, VOLTAGEM NUMBER(5) NOT NULL, COD_PROD NUMBER(5) NOT NULL,

CONSTRAINT ITEM_PK PRIMARY KEY (NUM_SERIE), CONSTRAINT ITEM_FK FOREIGN KEY (COD_PROD)

REFERENCES PRODUTO (COD))

Observações:

- A empresa comercializa produtos controlados por quantidade (por exemplo, caixa de sabão em pó, pacote de biscoito e lata de extrato de tomate) e produtos controlados por unidade (por exemplo, televisor, máquina de lavar roupa e liquidificador).
- A quantidade em estoque de um produto controlado por quantidade (TIPO=1) é obtida diretamente da coluna QTD_ESTOQUE.

- A quantidade em estoque de um produto controlado por unidade (TIPO=2) NÃO pode ser obtida diretamente da coluna QTD_ESTOQUE, pois, para esse tipo de produto, esta coluna irá conter o valor NULL.
- Cada linha da tabela ITEM contém informações sobre um item existente no estoque da empresa
relativo a um tipo de produto controlado por unidade.

Qual comando SQL irá inserir corretamente uma nova linha na tabela de produtos, além de não violar restrições semânticas relativas ao banco de dados do supermercado?

a) INSERT INTO PRODUTO (COD, DESCRICAO, PRECO, TIPO)
VALUES (7777, 'COMPUTADOR BLUEX', 1000.00, 2)

b) INSERT INTO PRODUTO VALUES (7777,'COMPUTADOR BLUEX',1000.00,2)

c) INSERT INTO PRODUTO VALUES (8888,'SARDINHA EM LATA BOM PEIXE',2.50,700,2)

d) INSERT INTO PRODUTO (COD,DESCRICAO,PRECO,QTD_ESTOQUE)
VALUES(7777,'COMPUTADOR BLUEX',1000.00, ,2)

e) INSERT INTO PRODUTO (COD,DESCRICAO,PRECO,TIPO,QTD_ESTOQUE)
VALUES(7777,'COMPUTADOR BLUEX',1000.00,2)

Comentário:
Veja que a nossa resposta se encontra na alternativa A. A letra B está errada pois existe um atributo que, embora possa vir a ser nulo, ele aparece na declaração do comando CREATE entre os PRECO e TIPO. Logo não temos como fazer a inserção desta forma. Já a letra c apresenta um valor no atributo quantidade de estoque quando o tipo é 2, isso foi definido como algo contra as regras do modelo. Por fim, as alterativa D e E não informam valor para o atributo tipo, que não pode ser nulo.

Gabarito: A

45. Ano: 2014 Banca: CESGRANRIO Órgão: FINEP Prova: Analista - Desenvolvimento de Sistemas Para responder essa questão use a mesma referência da questão anterior.

O analista de suporte de banco de dados do supermercado solicitou que a coluna QTD_ESTOQUE passasse a conter a quantidade de itens em estoque de produtos do tipo 2. Embora ele reconheça que isso resultará em redundância, os relatórios de performance mostram que existe um desperdício de recursos computacionais significativo com o cálculo recorrente do total de itens em estoque de produtos do tipo 2.

Qual comando SQL irá atualizar corretamente a coluna QTD_ESTOQUE com a quantidade de itens em estoque relativa a cada um dos produtos do tipo 2 comercializados pelo supermercado?

a) UPDATE PRODUTO P SET QTD_ESTOQUE = (SELECT COUNT(*) FROM ITEM WHERE COD_PROD=P.COD)

b) UPDATE PRODUTO P SET QTD_ESTOQUE = (SELECT COUNT(*) FROM ITEM WHERE COD_PROD=P.COD AND P.TIPO=2)

c) UPDATE PRODUTO P SET QTD_ESTOQUE = (SELECT COUNT(DISTINCT COD_PROD) FROM ITEM WHERE COD_PROD=P.COD AND P.TIPO=2)

d) UPDATE PRODUTO P SET QTD_ESTOQUE=(SELECT COUNT(COD_PROD) FROM ITEM WHERE COD_PROD=P.COD) WHERE TIPO = 2

e) UPDATE PRODUTO P SET QTD_ESTOQUE=(SELECT COUNT(DISTINCT COD_PROD) FROM ITEM WHERE COD_PROD=P.COD) WHERE TIPO=2

Comentário:

Nesta questão vamos precisar contar a quantidade de produtos em estoque, agrupada por código do produto. E, em seguida, atualizar o valor na tabela. Assim, podemos encontra nossa resposta na alternativa D.

Gabarito: D

46. Ano: 2014 Banca: CESGRANRIO Órgão: FINEP Prova: Analista - Desenvolvimento de Sistemas

Ao implementar um sistema de gerência de fornecedores, o desenvolvedor percebeu que não existia no banco de dados relacional da empresa qualquer representação da entidade PRODUTO que aparecia em seu modelo de dados. Para corrigir essa falha, preparou um comando SQL que alteraria o esquema do banco de dados.

Tal comando SQL deve ser iniciado com

a) ALTER SCHEMA ADD TABLE PRODUTO
b) ALTER TABLE PRODUTO

c) CREATE PRODUTO : TABLE

d) CREATE PRODUTO AS TABLE

e) CREATE TABLE PRODUTO

Comentário:

Essa entidade precisa ser criada no banco de dados. Isso é feito por meio do comando CREATE TABLE PRODUTO.

Gabarito: E

47. Ano: 2013 Banca: CESGRANRIO Órgão: IBGE Prova: Analista - Suporte Operacional

Em um banco de dados, a tabela Pessoa foi criada com a seguinte instrução: CREATE TABLE Pessoa (PessoaID int,

Nome varchar(255), Sobrenome varchar(255), Endereco varchar(255), Cidade varchar(255)

);

Que instrução SQL acrescenta um campo CEP do tipo varchar(9) a essa tabela?

a) ADD COLUMN CEP varchar(9) INTO TABLE

b) ALTER TABLE Pessoa ADD CEP varchar(9)

c) ALTER TABLE Pessoa INSERT COLUMN CEP varchar(9)

d) ALTER TABLE Pessoa ALTER COLUMN CEP varchar(9)

e) ALTER TABLE Pessoa MODIFY COLUMN ADD CEP varchar(9)

Comentário:

Para acrescentar uma coluna na tabela basta utilizar o comando ALTER TABLE em seguida a clausula ADD com o nome da coluna e o tipo. Vejamos, mais uma vez, a sintaxe do comando:

ALTER TABLE *table_name*

ADD *column_name datatype*;

Veja que o nome e o tipo são informações passadas pelo enunciado, desta forma, podemos marcar o gabarito na alternativa B.

Gabarito: B

48. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Informática Júnior

```
CREATE TABLE Inquilino(
      nome                  VARCHAR(20) NULL,
      cpf                   CHAR(11) NOT NULL,
      PRIMARY KEY (cpf));

CREATE TABLE Vaga(
      andar                 INTEGER NOT NULL,
      numero                INTEGER NOT NULL,
      PRIMARY KEY (andar,numero));

CREATE TABLE Vaga_Inquilino(
      andar                 INTEGER NOT NULL,
      numero                INTEGER NOT NULL,
      cpf                   CHAR(11) NOT NULL,
      PRIMARY KEY (andar,numero,cpf));
```

Desse modo, para incluir o campo telefone na Tabela Inquilino, o comando necessário é

a) add column telefone varchar(12) on table Inquilino;
b) alter table Inquilino add column telefone varchar(12);
c) alter table Inquilino insert column telefone varchar(12);
d) insert column telefone varchar(12) on table Inquilino;
e) modify table Inquilino add column telefone varchar(12);

Comentário:

Essa questão é semelhante a anterior. As únicas coisas que mudam são o nome da coluna que passa ser telefone, o parâmetro do tipo que agora é varchar (12) e o nome da tabela que passa a ser inquilino. Veja que até a alternativa que é considerada a resposta da questão é a mesma, a alternativa B.

Gabarito: B

49. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Informática Júnior

```
CREATE TABLE Inquilino(
      nome                  VARCHAR(20) NULL,
      cpf                   CHAR(11) NOT NULL,
      PRIMARY KEY (cpf));

CREATE TABLE Vaga(
      andar                 INTEGER NOT NULL,
      numero                INTEGER NOT NULL,
      PRIMARY KEY (andar,numero));

CREATE TABLE Vaga_Inquilino(
      andar                 INTEGER NOT NULL,
      numero                INTEGER NOT NULL,
      cpf                   CHAR(11) NOT NULL,
      PRIMARY KEY (andar,numero,cpf));
```

Nessa situação, para se obter um relatório com a quantidade de vagas por cada inquilino, listadas e agrupadas por cpf, deve ser feita a seguinte consulta:

a) select cpf,andar,numero from Vaga_Inquilino;

b) select cpf,count(*) from Vaga_Inquilino group by cpf;

c) select cpf,sum(cpf) from Vaga_Inquilino group by cpf;

d) select distinct cpf from Vaga_Inquilino;

e) select distinct count(Inquilino.cpf) from Inquilino, Vaga_Inquilino Where Inquilino.cpf = Vaga_Inquilino.cpf order by Inquilino.cpf;

Comentário:

Vejam que para construir a consulta precisamos apenas da tabela Vaga_Inquilino, pois a mesma já apresenta os dois atributos elencados no enunciado que devem aparecer no relatório. Neste sentido, vamos fazer um agrupamento por CPF e contar as diferentes vagas que cada um deles possui. Assim, podemos marcar nossa resposta na alternativa B.

Gabarito: B

50. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação Considere as Tabelas a seguir para responder a questão.

Essas Tabelas fazem parte do esquema de um banco de dados usado por uma associação de criadores de cães para organizar informações sobre os torneios que ela promove.

Observações:

- A Tabela CAO contém os dados dos cães inscritos na referida associação. A coluna NOME_PAI indica o nome do pai de um cão, a coluna RACA indica a raça do mesmo, e a coluna NOME_PROPR indica o nome do seu proprietário. As demais colunas são autoexplicativas.
- A Tabela COMPETICAO contém informações sobre as competições patrocinadas pela associação.

Suas colunas são autoexplicativas.

- A Tabela PARTICIPACAO informa as competições das quais participaram os cães

registrados na associação. Cada linha dessa tabela indica a colocação obtida por um cão em uma determinada competição. Suas colunas são autoexplicativas.

- A Tabela ARBITRO contém os dados dos árbitros que julgam os cães que participam de competições. Suas colunas são autoexplicativas.
- Cada linha da Tabela AVALIACAO representa a nota atribuída a um cão, por um determinado árbitro em uma determinada competição. Suas colunas são autoexplicativas.

No contexto do torneio, cuja descrição é ABERTO DO RIO, qual consulta exibe, em ordem decrescente de somatório de notas, os nomes dos cães participantes e o somatório das notas que cada um recebeu?

a) SELECT C.NOME,SUM(NOTA_ARBTR) FROM CAO C,COMPETICAO P,AVALIACAO N
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=N.COD_COMP AND N.COD_CAO=C.COD
GROUP BY C.NOME
ORDER BY SUM(NOTA_ARBTR) DESC

b) SELECT C.NOME,SUM(NOTA_ARBTR) FROM CAO C,AVALIACAO N
WHERE N.COD_COMP='ABERTO DO RIO' AND AND N.COD_CAO=C.COD GROUP BY C.NOME
ORDER BY SUM(NOTA_ARBTR) DESCENDING

c) SELECT C.NOME,SUM(NOTA_ARBTR) FROM CAO C,COMPETICAO P,AVALIACAO N
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=N.COD_COMP AND N.COD_CAO=C.COD
ORDER BY SUM(NOTA_ARBTR) DESCENDING

d) SELECT C.NOME,SUM(O.COLOCACAO) FROM CAO C,COMPETICAO P,PARTICIPACAO O
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=O.COD_COMP AND O.COD_CAO=C.COD
GROUP BY C.NOME
ORDER BY SUM(O.COLOCACAO)

e) SELECT C.NOME,SUM(N.NOTA_ARBTR) FROM CAO C,COMPETICAO P,AVALIACAO N
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=N.COD_COMP AND N.COD_CAO=C.COD
GROUP BY C.NOME
ORDER BY SUM(N.NOTA_ARBTR)

Comentário:
Vejam que nesta questão precisamos apresentar os nomes dos cães participantes e o somatório das notas que cada um deles. Vejam que são apenas 2 atributos. Agora temos as restrições que devem aparecer no predicado: basicamente, a descrição do torneio deve ser igual a ABERTO DO RIO. Assim ficamos com:

```
SELECT C.NOME,SUM(NOTA_ARBTR)
FROM CAO C,COMPETICAO P,AVALIACAO N
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=N.COD_COMP AND N.COD_CAO=C.COD
GROUP BY C.NOME
```

ORDER BY SUM(NOTA_ARBTR) DESC

Presente na alternativa A.

Gabarito: A.

51. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação

Considere as Tabelas a seguir para responder a questão.

Essas Tabelas fazem parte do esquema de um banco de dados usado por uma associação de criadores de cães para organizar informações sobre os torneios que ela promove.

```
CREATE TABLE CAO (
   COD            NUMBER(5)      NOT NULL,
   NOME           VARCHAR2(50)   NOT NULL,
   RACA           VARCHAR2(50)   NOT NULL,
   NOME_PAI       VARCHAR2(50),
   NOME_PROPR     VARCHAR2(50)   NOT NULL,
   CONSTRAINT CAO_PK PRIMARY KEY (COD)
)
CREATE TABLE COMPETICAO (
   COD            NUMBER(5)      NOT NULL,
   DESCR          VARCHAR2(50)   NOT NULL,
   CONSTRAINT COMPETICAO_PK PRIMARY KEY (COD)
)
CREATE TABLE ARBITRO (
   COD            NUMBER(5)      NOT NULL,
   NOME           VARCHAR2(50)   NOT NULL,
   CONSTRAINT ARBITRO_PK PRIMARY KEY (COD)
)
CREATE TABLE PARTICIPACAO (
   COD_CAO        NUMBER(5)       NOT NULL,
   COD_COMP       NUMBER(5)       NOT NULL,
   COLOCACAO      NUMBER(4)       NOT NULL,
   CONSTRAINT PARTICIPACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP),
   CONSTRAINT PARTICIPACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT PARTICIPACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD)
)
CREATE TABLE AVALIACAO (
   COD_CAO        NUMBER(5)       NOT NULL,
   COD_COMP       NUMBER(5)       NOT NULL,
   COD_ARBTR      NUMBER(5)       NOT NULL,
   NOTA_ARBTR    NUMBER(3,1)      NOT NULL,
   CONSTRAINT AVALIACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP,COD_ARBTR),
   CONSTRAINT AVALIACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT AVALIACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD),
   CONSTRAINT AVALIACAO_FK3 FOREIGN KEY (COD_ARBTR)
      REFERENCES ARBITRO (COD)
)
```

Observações:

- A Tabela CAO contém os dados dos cães inscritos na referida associação. A coluna NOME_PAI indica o nome do pai de um cão, a coluna RACA indica a raça do mesmo, e a coluna NOME_PROPR indica o nome do seu proprietário. As demais colunas são autoexplicativas.
- A Tabela COMPETICAO contém informações sobre as competições patrocinadas pela associação. Suas colunas são autoexplicativas.
- A Tabela PARTICIPACAO informa as competições das quais participaram os cães registrados na associação. Cada linha dessa tabela indica a colocação obtida por um cão em uma determinada competição. Suas colunas são autoexplicativas.
- A Tabela ARBITRO contém os dados dos árbitros que julgam os cães que participam de competições. Suas colunas são autoexplicativas.
- Cada linha da Tabela AVALIACAO representa a nota atribuída a um cão, por um determinado árbitro em uma determinada competição. Suas colunas são autoexplicativas.

Considerando-se o universo de todas as competições promovidas pela associação de criadores de cães, qual consulta exibe o nome do árbitro, cujo código é 1111, e a média das notas que ele atribuiu ao cão chamado GINGER?

a) SELECT A.NOME, AVG(N.NOTA_ARBTR) FROM CAO C,AVALIACAO N,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111
GROUP BY A.COD

b) SELECT A.NOME, AVG(N.NOTA_ARBTR)
FROM CAO C,AVALIACAO N,PARTICIPACAO P,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=P.COD_CAO AND P.COD_CAO=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111
GROUP BY N.NOTA_ARBTR

c) SELECT A.NOME, SUM(N.NOTA_ARBTR) / COUNT(*) FROM CAO C,AVALIACAO N,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111

d) SELECT A.NOME, AVG(N.NOTA_ARBTR)
FROM CAO C,AVALIACAO N,PARTICIPACAO P,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=P.COD_CAO AND P.COD_CAO=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111

e) SELECT A.NOME, SUM(N.NOTA_ARBTR) / COUNT(*) FROM CAO C,AVALIACAO N,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111

GROUP BY A.NOME

Comentário:

Para calcularmos a média, precisamos dividir a soma das notas atribuídas pela quantidade de notas (não sei porque o examinador não usou a média. É necessário fazer o produto cartesiano entre as tabelas CAO, AVALIACAO e ARBITRO, e restringir o nome do cão a GINGER e o número do árbitro é 1111. Ainda na clausula WHERE, precisamos fazer umas restrições para transformas o produto cartesiano em uma junção. Para finalizar, temos que agrupar pelo nome. Desta forma, temos nossa resposta na alternativa E.

Gabarito: E

52. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação

Considere as Tabelas a seguir para responder a questão.

Essas Tabelas fazem parte do esquema de um banco de dados usado por uma associação de criadores de cães para organizar informações sobre os torneios que ela promove.

```
CREATE TABLE CAO (
   COD            NUMBER(5)      NOT NULL,
   NOME           VARCHAR2(50)   NOT NULL,
   RACA           VARCHAR2(50)   NOT NULL,
   NOME_PAI       VARCHAR2(50),
   NOME_PROPR     VARCHAR2(50)   NOT NULL,
   CONSTRAINT CAO_PK PRIMARY KEY (COD)
)
CREATE TABLE COMPETICAO (
   COD            NUMBER(5)      NOT NULL,
   DESCR          VARCHAR2(50)   NOT NULL,
   CONSTRAINT COMPETICAO_PK PRIMARY KEY (COD)
)
CREATE TABLE ARBITRO (
   COD            NUMBER(5)      NOT NULL,
   NOME           VARCHAR2(50)   NOT NULL,
   CONSTRAINT ARBITRO_PK PRIMARY KEY (COD)
)
CREATE TABLE PARTICIPACAO (
   COD_CAO        NUMBER(5)       NOT NULL,
   COD_COMP       NUMBER(5)       NOT NULL,
   COLOCACAO      NUMBER(4)       NOT NULL,
   CONSTRAINT PARTICIPACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP),
   CONSTRAINT PARTICIPACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT PARTICIPACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD)
)
CREATE TABLE AVALIACAO (
   COD_CAO        NUMBER(5)       NOT NULL,
   COD_COMP       NUMBER(5)       NOT NULL,
   COD_ARBTR      NUMBER(5)       NOT NULL,
   NOTA_ARBTR    NUMBER(3,1)      NOT NULL,
   CONSTRAINT AVALIACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP,COD_ARBTR),
   CONSTRAINT AVALIACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT AVALIACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD),
   CONSTRAINT AVALIACAO_FK3 FOREIGN KEY (COD_ARBTR)
      REFERENCES ARBITRO (COD)
)
```

Observações:

- A Tabela CAO contém os dados dos cães inscritos na referida associação. A coluna NOME_PAI indica o nome do pai de um cão, a coluna RACA indica a raça do mesmo, e a coluna NOME_PROPR indica o nome do seu proprietário. As demais colunas são autoexplicativas.
- A Tabela COMPETICAO contém informações sobre as competições patrocinadas pela associação. Suas colunas são autoexplicativas.
- A Tabela PARTICIPACAO informa as competições das quais participaram os cães registrados na associação. Cada linha dessa tabela indica a colocação obtida por um cão em uma determinada competição. Suas colunas são autoexplicativas.
- A Tabela ARBITRO contém os dados dos árbitros que julgam os cães que participam de competições. Suas colunas são autoexplicativas.
- Cada linha da Tabela AVALIACAO representa a nota atribuída a um cão, por um determinado árbitro em uma determinada competição. Suas colunas são autoexplicativas.

Qual consulta exibe os nomes dos cães que participaram de, pelo menos, uma competição?

a) SELECT C.NOME FROM CAO C MINUS
SELECT DISTINCT C.NOME FROM CAO C,PARTICIPACAO P WHERE C.COD=P.COD_CAO
GROUP BY C.NOME
HAVING COUNT(*) > 0

b) SELECT C.NOME FROM CAO C
WHERE C.COD NOT IN (SELECT P.COD_CAO FROM PARTICIPACAO P WHERE
P.COD_CAO=C.COD)

c) SELECT C.NOME FROM CAO C
WHERE C.COD IN (SELECT COUNT(*) FROM PARTICIPACAO P WHERE P.COD_CAO=C.COD)

d) SELECT C.NOME
FROM CAO C, PARTICIPACAO P WHERE C.COD=P.COD_CAO GROUP BY C.NOME
HAVING COUNT(*) > 0

e) SELECT C.NOME
FROM CAO C,PARTICIPACAO P
WHERE C.COD=P.COD_CAO AND COUNT(*) > 0 GROUP BY C.NOME

Comentário:
Essa questão é relativamente simples, precisamos primeiramente fazer uma junção entre a tabela cao e participação pois queremos o nome do cão no resultado. Em seguida, basta procurarmos os cães que participaram de mais de uma competição (count (*) >0) após ter ordenado para agrupar por nome. Assim retornaremos os nomes dos cães que participaram de pelo menos uma competição. E a nossa resposta, encontra-se na alternativa D.

Gabarito: D

53. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação

O administrador de um banco de dados deseja remover do usuário RH5678 o privilégio de excluir linhas da tabela RH05_FUNCIONARIO.

Qual comando SQL executará o que esse administrador deseja?

a) REVOKE DELETE ON RH05_FUNCIONARIO FROM RH5678
b) PURGE DELETE FROM RH5678 ON RH05_FUNCIONARIO
c) DROP DELETE ON RH05_FUNCIONARIO FROM USER RH5678
d) DROP FUNCTION DELETE ON RH05_FUNCIONARIO FROM RH5678
e) DELETE FUNCTION DELETE FROM RH5678 ON RH05_FUNCIONARIO

Comentário:

Essa questão nos remete a parte da linguagem de SQL que trada do controle dos dados (DCL). Ela apresenta o comando que remove privilégios de acesso, no caso o REVOKE. Veja que somente com essa informação já conseguimos marcar a alternativa correta.

Gabarito: A

QUESTÕES COMENTADAS FGV

QUESTÕES COMENTADAS FGV

Vamos agora apresentar um conjunto de questões resolvida que servirão para fixação do conteúdo. Sempre que possível vamos inserir algum detalhamento teórico na explicação da questão. Esperamos que vocês gostem. As primeiras questões são da banca FGV todas comentadas. Em seguida, termos uma bateria de questões das mais diversas bancas.

1. **FGV - FTE (SEFAZ MT)/SEFAZ MT/2023 -TI - Banco de Dados - Sublinguagens SQL (DDL, DML, DQL, DCL e DTL)**

No contexto da concessão de privilégios/permissões na administração de bancos de dados relacionais, assinale a opção que indica dois dos comandos básicos disponíveis na maior parte dos SGDB.

a) ADD e DELETE.

b) ALLOW e CONSTAINT.

c) ATTACH e DETACH.

d) GRANT e REVOKE.

e) INSERT e REMOVE.

Comentário: A resposta correta é: **d) GRANT e REVOKE.** Esses são dois dos comandos básicos disponíveis na maioria dos Sistemas Gerenciadores de Banco de Dados (SGBD) para conceder (GRANT) e revogar (REVOKE) privilégios ou permissões a usuários e funções.

Gabarito: D

---

2. **FGV - AJ (TJ RN)/TJ RN/Apoio Especializado/Análise de Sistemas/2023 - TI - Banco de Dados - Conceitos e Fundamentos de Modelo Relacional**

Supondo que no Brasil

- todo brasileiro tem um, e somente um, CPF;

- alguns brasileiros têm um, mas somente um, passaporte válido;

- alguns brasileiros têm uma, mas somente uma, carteira de motorista (CNH) válida.

A definição **correta** desses atributos numa tabela relacional normalizada seria:

a) CPF varchar(11) not null

CNH varchar(9) UNIQUE

Passaporte varchar(9) UNIQUE

b) CPF varchar(11) not null UNIQUE

CNH varchar(9) UNIQUE

Passaporte varchar(9) UNIQUE

c) CPF varchar(11) UNIQUE

CNH varchar(9) not null

Passaporte varchar(9) not null

d) CPF varchar(11) not null

CNH varchar(9) null

Passaporte varchar(9) null

e) CPF varchar(11) null

CNH varchar(9) UNIQUE

Passaporte varchar(9) UNIQUE

Comentário: A definição correta dos atributos numa tabela relacional normalizada é:

b) CPF varchar(11) not null UNIQUE

CNH varchar(9) UNIQUE

Passaporte varchar(9) UNIQUE

Nesse caso, CPF é definido como not null (não pode ser nula) e único (chave única) para garantir que cada brasileiro tenha um, e somente um, CPF. CNH e Passaporte também são definidos como únicos (caso cada brasileiro tenha apenas um deles) e podem ser nulos para indicar que a pessoa não possui esses documentos.

Resumindo temos que:

- todo brasileiro tem um, e somente um, CPF (restrição not null e UNIQUE), garantindo que cada indivíduo tenha um CPF exclusivo registrado.
- alguns brasileiros têm um, mas somente um, passaporte válido (restrição UNIQUE), indicando que nem todos os brasileiros possuem um passaporte, mas aqueles que têm possuem apenas um registro válido.
- alguns brasileiros têm uma, mas somente uma, carteira de motorista (CNH) válida (restrição UNIQUE), o que significa que nem todos os brasileiros possuem CNH, mas aqueles que têm possuem apenas um registro válido.

Gabarito: B

---

3. **FGV - Ana (PGM Niterói)/Pref Niterói/Tecnologia da Informação/2023- TI - Banco de Dados - Álgebra Relacional**

Considere o comando SQL a seguir.

SELECT a.X, b.Y FROM T1 a, T2 b WHERE a.R = b.S

Dado que essa consulta pode ser expressa usando as operações primitivas da Álgebra Relacional, a lista que contém as operações necessárias e suficientes para essa conversão é, em ordem alfabética:

a) Diferença, Produto, Projeção;

b) Produto, Projeção, União;

c) Projeção, União;

d) Produto, Projeção, Seleção;

e) Seleção, União.

Comentário: A consulta SQL apresentada pode ser convertida em Álgebra Relacional usando as operações primitivas da seguinte forma:

Produto (Cartesiano) - Combina as tuplas de T1 e T2.

Seleção (Restrição) - Filtra as tuplas onde a condição a.R = b.S é verdadeira.

Projeção - Seleciona apenas as colunas a.X e b.Y.

Portanto, as operações necessárias e suficientes para essa conversão são Produto, Seleção e Projeção.

A resposta correta é: d) Produto, Projeção, Seleção.

Gabarito: D

---

4. **FGV - ACE (TCE ES)/TCE ES/Tecnologia da Informação/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Junior é o administrador do Banco de Dados da sua empresa e percebeu que um programador tinha acesso de alteração indevido a uma tabela.

Para cancelar a permissão previamente concedida ao programador, Junior deve usar o comando SQL:

a) GRANT;

b) REVOKE;

c) UPDATE;

d) DELETE;

e) TRUNCATE.

Comentário: Junior deve usar o comando SQL **b) REVOKE** para cancelar a permissão previamente concedida ao programador. O comando **REVOKE** é usado para retirar permissões que foram previamente concedidas usando o comando **GRANT**. Isso permite revogar o acesso ou privilégios de um usuário ou papel em relação a objetos do banco de dados. Portanto, é a opção correta para remover as permissões indevidas concedidas ao programador.

Gabarito: B

---

5. **FGV - ATRFB/SRFB/Geral/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Os principais Sistemas Gerenciadores de Bancos de Dados oferecem total suporte à linguagem SQL. Um aspecto importante da implementação do SQL é o tratamento para valores nulos quando esses são considerados como *unknown values*.

Nesse contexto, considere uma tabela T com colunas A e B, que podem conter valores nulos. T possui 100 registros e, em 50% das linhas, há pelo menos uma coluna preenchida com o valor NULL.

Considere a consulta a seguir:

```
SELECT * FROM T t1
WHERE t1.A = NULL or t1.B = NULL
```

O número máximo de linhas de resultados que seriam retornadas pela consulta é igual a

a) 0.

b) 25.

c) 50.

d) 75.

e) 100.

Comentário: A consulta apresentada na pergunta não retornará nenhum resultado.

O motivo é que, em SQL, o operador de igualdade para valores nulos é tratado de forma especial. Para verificar se uma coluna é nula, você deve usar o operador **IS NULL** em vez de **= NULL**. Portanto, a consulta correta seria:

```
SELECT * FROM T t1 WHERE t1.A IS NULL OR t1.B IS NULL
```

Nesse caso, a consulta retornaria todas as linhas em que a coluna A ou a coluna B (ou ambas) são nulas. Dado que em 50% das linhas pelo menos uma coluna é nula, o número máximo de linhas retornadas seria igual a 50% de 100, o que é igual a **50**.

Portanto, a resposta correta é **a) 0**.

Gabarito: A

---

6. **FGV - ATRFB/SRFB/Geral/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Num banco de dados relacional, considere uma tabela R, com duas colunas A e B, ambas do tipo *string* de caracteres, cuja instância é exibida a seguir.

A	B
Pedro	João
Maria	Ida
Maria	Ida
Pedro	João
Edson	Wilson
Edson	Maria

Nesse cenário analise os comandos a seguir.

I.

```
DELETE FROM R
WHERE EXISTS (SELECT * FROM R r1
              WHERE R.A = r1.A and R.B = r1.B)
```

II.

```
DELETE FROM R
WHERE EXISTS (SELECT * FROM R r1
              WHERE R.A + R.B > r1.A + r1.B)
```

III.

```
DELETE FROM R
WHERE R.A + R.B in (SELECT A + B FROM R)
```

Assinale a lista que contém o número de registros deletados em cada um dos comandos **I**, **II** e **III**, respectivamente, quando executados separadamente e usando a mesma instância inicial descrita.

a) 2, 2 e 0.

b) 2, 4 e 0.

c) 4, 4 e 4.

d) 6, 5 e 6.

e) 6, 6 e 6.

Comentário: A primeira coisa que é necessário saber para resolver a questão é que o sinal de "+" quando aplicado como operação entre duas strings (textos) vai fazer a concatenação dos termos. É importante salientar que essa operação está correta para o banco de dados SQL Server. Se fosse utilizado o Oracle, DB2 ou o PostgreSQL, era necessário usar || e caso a escolha seja pelo MySQL deveríamos usar a função concat(). Essa lógica será usada para responder as alternativas B e C.

Assim, o comando I vai remover a linha se existir na própria tabela uma linha onde os atributos A e B são iguais. Por óbvio, isso acontece em todas as linhas e iremos remover as 6 linhas da tabela.

Já no comando II temos que criar uma concatenação entre os valores A e B, em seguida ordenar lexicograficamente. Depois serão removidas todas as linhas em que na tabela tivermos um elemento menor do que ele. Para facilitar sua visualização vamos fazer a concatenação dos termos e apresentá-los a seguir em ordem alfabética:

EdsonMaria

EdsonWilson

MariaIda

MariaIda

PedroJoão

PedroJoão

Veja que a linha EdsonMaria não vai ter nenhum termo ou item que seja menor do que ela. Logo, essa tupla não será removida. As demais serão apagadas da tabela após a execução do comando. Logo, 5 linhas serão removidas.

Por fim, o comando III temos uma situação parecida com a do comando I. Neste item, entretanto, é feita a concatenação dos atributos antes de fazer a comparação de igualdade. Mais uma vez, sempre teremos elementos concatenados iguais e todas as linhas serão removidas. Logo, 6 linhas serão deletadas.

Desta forma temos que, 6, 5 e 6, linhas serão excluídas, respectivamente, pelos comandos I, II e III. Assim, a nossa resposta encontra-se na alternativa D.

Gabarito: D

---

7. **FGV - AFRFB/SRFB/Geral/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Num banco de dados relacional, considere a tabela Vencedores, cuja instância é exibida a seguir, com duas colunas, Tenista e Torneio, que representam alguns torneios que já foram vencidos por alguns tenistas.

Tenista	Torneio
Roger Federer	Australian Open
Roger Federer	Roland Garros
Roger Federer	Wimbledon

Roger Federer	US Open
Pete Sampras	US Open
Pete Sampras	Wimbledon
Pete Sampras	Australian Open
Bjorn Borg	Roland Garros
Bjorn Borg	Wimbledon

Maria precisa escrever um comando SQL que liste os tenistas que venceram todos os torneios mencionados na coluna Torneio. O comando deve valer para qualquer instância válida da tabela, que pode conter diferentes tenistas e diferentes torneios.

Assinale o comando que Maria deve usar.

a) select distinct Tenista from Vencedores v1

where v1.Torneio in (select Torneio from Vencedores)

b) select distinct Tenista from Vencedores v1

where exists(

select * from Vencedores v2

where v1.Torneio = v1.Torneio

and v1.Tenista = v2.Tenista

and v1 <> v2))

c) select distinct Tenista from Vencedores v1

where exists (

select * from Vencedores v2

where v1.Torneio = v1.Torneio

and v1.Tenista <> v2.Tenista)

d) select distinct Tenista from Vencedores v1

where for all (

select * from Vencedores v2

where exists (

select * from Vencedores v3

where v1.Tenista = v2.Tenista))

e) select distinct Tenista from Vencedores v1

where not exists(

select * from Vencedores v2

where not exists (

select * from Vencedores v3

where v2.Torneio = v3.Torneio

and v1.Tenista = v3.Tenista))

Comentário: Essa é uma questão de divisão em SQL. A ideia é pegar a lista de torneios e ver quem foram os tenistas que os venceram. Quem faz isso de forma correta é a alternativa E. A forma de construção da consulta está com 2 NOT EXISTS, dessa forma o que estamos procurando é se existe algum torneio que o tenista não venceu, se **não existir, o tenista venceu todos os torneios. Certo?**

Vejamos a explicação junto com o código SQL.

```
select distinct Tenista from Vencedores v1 -- pegue todos os tenistas distintos
where not exists( -- que não
   select Torneio from Vencedores v2 -- tenham torneios
   where not exists ( -- que não
      select Tenista, Torneio from Vencedores v3 -- tenham sido vencidos pelo v1.Tenista
         where v2.Torneio = v3.Torneio
            and v1.Tenista = v3.Tenista))
```

Desta forma, temos a nossa reposta correta na alternativa E

Gabarito: E

---

8. **FGV - AFRFB/SRFB/Geral/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Os principais Sistemas Gerenciadores de Bancos de Dados oferecem total suporte à linguagem SQL; um aspecto importante da implementação do SQL é o tratamento para valores nulos, quando a lógica admite três estados.

T – *true*

F – *false*

? – *unknown*

Nesse contexto, considere as expressões lógicas a seguir.

I. (T OR F) AND (? OR T)

II. T AND ((? OR F) OR ?)

III. NOT (? AND (? AND ?))

Com relação às expressões acima, está correto afirmar que o valor final é *unknown* (?) em

a) **I**, apenas.

b) **I** e **II**, apenas.

c) **I** e **III**, apenas.

d) **II** e **III**, apenas.

e) **I**, **II** e **III**.

Comentário: Vamos analisar cada uma das alternativas:

I. (T OR F) AND (? OR T)

(T OR F) é igual a T, porque é verdadeiro quando pelo menos um dos operandos é verdadeiro.

(? OR T) é igual a T, porque qualquer operação OR com um valor desconhecido resulta em verdadeiro.

Portanto, a expressão final é T AND T, que é igual a T.

II. T AND ((? OR F) OR ?)

(? OR F) é igual a ?, pois qualquer operação com um valor desconhecido resulta em desconhecido.

Então a expressão se torna T AND (? OR ?).

? OR ? é igual a ?, pois qualquer operação OR com pelo menos um valor desconhecido resulta em desconhecido.

Portanto, a expressão final é T AND ?, que é igual a ?.

III. NOT (? AND (? AND ?))

Primeiro, vamos resolver o interior da expressão: ? AND (? AND ?).

Qualquer operação AND que tenha pelo menos um valor desconhecido resulta em desconhecido, então ? AND (? AND ?) é igual a ?.

Agora, aplicamos o NOT a isso, o que transforma ? em ?.

Portanto, a resposta correta é **d) II e III, apenas**. As expressões II e III têm um valor final de **?**.

Gabarito: D

---

9. **FGV - Ana (PGM Niterói)/Pref Niterói/Tecnologia da Informação/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Figura 1 (a figura na prova é a composição do texto e da tabela)

Nas duas questões a seguir, considere a tabela relacional T cuja instância é exibida abaixo.

A instalação está configurada para o tratamento de valores NULL como valores desconhecidos *(unknown)*.

Tabela T

A	B	C
3	6	10
1	NULL	10
3	6	NULL
3	6	5

Considerando a tabela T da figura 1, analise a execução do comando SQL a seguir.

```
SELECT *
FROM T t1 LEFT JOIN T t2
        ON t1.A = t2.A and t1.B = t2.B
            and t1.C = t2.C
```

Além da linha de títulos, o número de linhas produzidas pela execução desse comando é:

a) 0;

b) 4;

c) 10;

d) 13;

e) 16.

Comentário: O comando SQL executa uma junção esquerda (LEFT JOIN) da tabela T com ela mesma, usando as colunas A, B e C como critérios de junção. A junção esquerda retorna todos os registros da tabela à esquerda (t1) e os registros correspondentes da tabela à direita (t2) que atendem aos critérios de junção. Como a tabela T possui 4 registros e a junção é com ela mesma, teremos uma combinação de todos os registros com todos os registros.

No entanto, a cláusula **ON t1.A = t2.A and t1.B = t2.B and t1.C = t2.C** especifica que todas as colunas devem corresponder, o que não é o caso em todos os registros da tabela T.

Vamos analisar os registros da tabela T:

O primeiro registro (3, 6, 10) corresponde a si mesmo (3, 6, 10).

O segundo registro (1, NULL, 10) não corresponde a nenhum outro registro porque a coluna B é NULL.

O terceiro registro (3, 6, NULL) não corresponde a nenhum outro registro porque a coluna C é NULL.

O quarto registro (3, 6, 5) corresponde a si mesmo (3, 6, 5).

Portanto, apenas dois registros da tabela T correspondem a outros registros com base nos critérios especificados. Como a junção retorna todos os registros da tabela à esquerda, teremos todos os 4 registros da tabela T mais os registros correspondentes, que são 2.

O número total de linhas produzidas pela execução do comando é 4 (registros originais da tabela T) + 2 (registros correspondentes) = 6.

A resposta correta é **b) 4**.

Gabarito: B

---

10. **FGV - Ana (PGM Niterói)/Pref Niterói/Tecnologia da Informação/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Figura 1

Nas duas questões a seguir, considere a tabela relacional T cuja instância é exibida abaixo.

A instalação está configurada para o tratamento de valores NULL como valores desconhecidos *(unknown)*.

Tabela T

A	B	C
3	6	10
1	NULL	10
3	6	NULL
3	6	5

Considerando a tabela T da figura 1, analise o comando a seguir.

```
DELETE FROM T
WHERE EXISTS ( SELECT * FROM T t2
                    WHERE T.A = t2.A
                    and T.B = t2.B
                    and T.C = t2.C )
```

O número de linhas removidas pela execução do comando acima é:

a) 0;

b) 1;

c) 2;

d) 3;

e) 4.

Comentário: O comando DELETE FROM T possui uma cláusula WHERE com uma subconsulta EXISTS. Esta subconsulta verifica se existe algum registro na tabela T (t2) que corresponda aos critérios especificados. Os critérios especificados são que os valores das colunas A, B e C em T devem corresponder aos valores correspondentes em t2.

Vamos analisar os registros da tabela T:

O primeiro registro (3, 6, 10) corresponde a si mesmo (3, 6, 10).

O segundo registro (1, NULL, 10) não corresponde a nenhum outro registro porque a coluna B é NULL.

O terceiro registro (3, 6, NULL) não corresponde a nenhum outro registro porque a coluna C é NULL.

O quarto registro (3, 6, 5) corresponde a si mesmo (3, 6, 5).

Portanto, apenas dois registros da tabela T correspondem a outros registros com base nos critérios especificados na subconsulta.

O comando DELETE FROM T removerá todos os registros da tabela T que atendem aos critérios da subconsulta. Como existem apenas dois registros que atendem a esses critérios, o número de linhas removidas pela execução do comando é **2**.

A resposta correta é **c) 2**. A banca, entretanto, deu o gabarito na alternativa A. Não tivemos recurso e a questão deve ter se mantida com o gabarito errado.

Gabarito: A

---

11. **FGV - FTE (SEFAZ MT)/SEFAZ MT/2023- TI - Banco de Dados - Consultas e Comandos em SQL**

No contexto das linguagens de manipulação de dados de SGBD relacionais, analise a instância da tabela T e o comando SQL a seguir.

pessoa	ancestral
Bruna	Joana
Joana	João
João	Maria

Maria	Gabriel
Paulo	Gabriel

insert into T

select t1.pessoa, t2.ancestral

from T t1, T t2

where t1.ancestral = t2.pessoa

and not exists

(select * from T tt

where tt.pessoa = t1.pessoa

and tt.ancestral = t2.ancestral)

Dado que o comando SQL acima foi executado por três vezes consecutivas, assinale o número de linhas inseridas na tabela **T** em cada execução, na ordem.

a) 0, 0, 0.

b) 3, 5, 0.

c) 5, 2, 1.

d) 5, 3, 0.

e) 8, 0, 0.

Comentário: Vou detalhar os resultados de cada execução do comando SQL:

Estado Inicial da Tabela T:

pessoa | ancestral

Bruna | Joana

Joana | João

João | Maria

Maria | Gabriel

Paulo | Gabriel

Primeira Execução:

Neste estágio, o comando **INSERT INTO** insere novas linhas na tabela T com base nas condições especificadas.

Para cada linha na tabela T, o comando procura outra linha onde a pessoa ancestral (**t1.ancestral**) seja igual à pessoa (**t2.pessoa**). Essa condição é representada por **WHERE t1.ancestral = t2.pessoa**.

Além disso, a cláusula **NOT EXISTS** verifica se não existe uma combinação idêntica na tabela T. Isso evita inserções duplicadas.

Na primeira execução, as linhas resultantes são as seguintes:

pessoa	ancestral
Bruna	Joana
Joana	João
João	Maria
Maria	Gabriel
Paulo	Gabriel
Bruna	João
Joana	Maria
João	Gabriel

Segunda Execução:

Novamente, para cada linha na tabela T, o comando procura outra linha onde a pessoa ancestral (**t1.ancestral**) seja igual à pessoa (**t2.pessoa**).

A cláusula **NOT EXISTS** verifica se não existe uma combinação idêntica na tabela T. Agora, como resultado da segunda execução, as seguintes linhas são inseridas:

pessoa	ancestral
Bruna	Joana
Joana	João
João	Maria
Maria	Gabriel
Paulo	Gabriel
Bruna	João
Joana	Maria
João	Gabriel
Bruna	Maria
Joana	Gabriel
Bruna	Maria
Joana	Gabriel
Bruna	Gabriel

Terceira Execução:

Novamente, para cada linha na tabela T, o comando procura outra linha onde a pessoa ancestral (**t1.ancestral**) seja igual à pessoa (**t2.pessoa**).

A cláusula **NOT EXISTS** verifica se não existe uma combinação idêntica na tabela T. Nesta terceira execução, nenhuma nova linha é inserida porque todas as combinações já existem na tabela.

pessoa	ancestral
Bruna	Joana
Joana	João
João	Maria
Maria	Gabriel
Paulo	Gabriel
Bruna	João
Joana	Maria
João	Gabriel
Bruna	Maria
Joana	Gabriel
Bruna	Maria
Joana	Gabriel
Bruna	Gabriel

Portanto, após a terceira execução, nenhuma nova linha é inserida, resultando em **0** novas linhas inseridas na tabela T.

Assim, é correto afirmar que, após a execução do comando SQL três vezes consecutivas:

Na primeira execução, **3** novas linhas foram inseridas na tabela T.

Na segunda execução, **5** novas linhas foram inseridas na tabela T.

Na terceira execução, **0** novas linhas foram inseridas na tabela T.

Gabarito: B

---

12. FGV - AJ (TJ RN)/TJ RN/Apoio Especializado/Análise de Sistemas/2023 - TI - Banco de Dados - Consultas e Comandos em SQL

Considere a estrutura e uma instância da tabela relacional FILIACAO exibida a seguir.

FILIACAO

Pessoa	Genitor
Alexandre	Francisco
Francisco	João
Joaquina	João
Carlota	Joaquina
João	Manuel
Paulo	Manuel
Maria	Paulo

Com relação à tabela FILIACAO, analise o comando SQL a seguir.

```
select distinct t3.Pessoa, t4.Pessoa
FROM FILIACAO t1, FILIACAO t2, FILIACAO t3, FILIACAO t4
WHERE t1.Pessoa < t2.Pessoa
and t1.Genitor = t2.Genitor
and t3.Genitor = t1.Pessoa
and t4.Genitor = t2.Pessoa
```

Afora a linha de títulos, o número de linhas produzidas pela execução do referido comando SQL é:

a) 3;

b) 4;

c) 5;

d) 6;

e) 7.

Comentário: Para entender o número de linhas produzidas pela execução do comando SQL fornecido, vamos analisar as condições da consulta. O comando SQL procura pares distintos de pessoas onde uma é a genitora da outra. Aqui estão as condições:

t1.Pessoa < t2.Pessoa garante que estamos considerando apenas pares únicos de pessoas.

t1.Genitor = t2.Genitor implica que ambas as pessoas têm o mesmo genitor.

t3.Genitor = t1.Pessoa e **t4.Genitor = t2.Pessoa** ligam as duas pessoas do par ao genitor de outra pessoa.

Dada a estrutura da tabela FILIACAO e as condições acima, podemos identificar os pares de pessoas que atendem a essas condições:

Alexandre e Joaquina (ambos filhos de Francisco).

Carlota e Maria (ambos filhos de Joaquina).

Francisco e Joaquina (ambos filhos de João).

Portanto, a consulta retorna 3 pares distintos de pessoas. Então, o número de linhas produzidas pela execução do comando SQL é A) 3.

Gabarito: A

---

13. **FGV - AJ (TJ RN)/TJ RN/Apoio Especializado/Análise de Sistemas/2023 -TI - Banco de Dados - Consultas e Comandos em SQL**

Considere a estrutura e uma instância da tabela relacional FILIACAO exibida a seguir.

FILIACAO

Pessoa	Genitor
Alexandre	Francisco
Francisco	João

Joaquina	João
Carlota	Joaquina
João	Manuel
Paulo	Manuel
Maria	Paulo

Com relação à tabela FILIACAO, definida anteriormente, o comando SQL que produz a lista dos nomes das pessoas que são citadas na instância da referida tabela, sem repetições, é:

a) select Pessoa FROM FILIACAO

UNION ALL select Genitor FROM FILIACAO

b) select Pessoa FROM FILIACAO

UNION select Genitor FROM FILIACAO

c) select distinct Pessoa, Genitor FROM FILIACAO

d) select distinct Pessoa UNION distinct Genitor

FROM FILIACAO

e) select Pessoa FROM FILIACAO and

select Genitor FROM FILIACAO

Comentário: Para obter a lista de nomes de pessoas sem repetições, você pode usar a cláusula **UNION**. A opção A está quase correta, mas a cláusula **UNION ALL** incluirá duplicatas, enquanto você deseja remover as duplicatas. Portanto, a opção correta é a B, que usa **UNION**, que remove automaticamente as duplicatas:

```
select Pessoa
FROM FILIACAO UNION
select Genitor
FROM FILIACAO
```

Essa consulta pegará os nomes da coluna "Pessoa" e os nomes da coluna "Genitor" da tabela FILIACAO e removerá automaticamente quaisquer duplicatas, resultando em uma lista de nomes de pessoas sem repetições. Portanto, a resposta correta é a opção B.

Gabarito: B

14. **FGV - AJ (TJ RN)/TJ RN/Apoio Especializado/Análise de Suporte/2023 -TI - Banco de Dados - Consultas e Comandos em SQL**

Luiz é o DBA do TJRN e atendeu um chamado da equipe de desenvolvimento que pedia para criar um banco de dados com 5 Gigabytes, mas, na hora de criar, Luiz digitou 50 Gigabytes.

Para apagar o banco de dados criado equivocadamente, Luiz deve utilizar o comando:

a) DROP;

b) ALTER;

c) DELETE;

d) UPDATE;

e) INJECTION.

Comentário: Para apagar um banco de dados em SQL, você deve usar o comando DROP. Portanto, a resposta correta é: a) DROP

Gabarito: A

---

15. **FGV - Ana (BBTS)/BBTS/Perfil Tecnológico/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Considere uma tabela T, com uma única coluna A, definida como uma chave primária, e o comando SQL a seguir.

```
delete from T
where not exists
(select * from T tt where T.A > tt.A)
```

Dado que a tabela tem 100 linhas preenchidas, assinale a opção que indica o número de linhas que será deletado pela execução do referido comando.

a) 0.

b) 1.

c) 50.

d) 99.

e) 100.

Comentário: O comando SQL DELETE está tentando deletar linhas da tabela T onde não existe outra linha na mesma tabela com um valor maior em A. Se a coluna A for uma chave primária, isso significa que os valores em A são únicos. Portanto, apenas uma linha será mantida, pois todas as outras terão valores menores que não atendem à condição.

Portanto, você está correto. A resposta correta é: **b) 1.**

Gabarito: B

---

16. **FGV - Ana (BBTS)/BBTS/Perfil Tecnológico/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Considere o comando de criação de uma tabela relacional num ambiente Oracle exibido a seguir.

```
create table T (
codigo int not null,
produto varchar (40) not null,
preço float not null default 0 check (preço >= 0),
imposto float not null default 0
check (imposto <=100))
```

Assinale o comando SQL de inserção de registro que provocaria um ***erro***.

a) insert into T(produto,código) values ('mesa',100)

b) insert into T values (100, 'mesa',100,10)

c) insert into T values (100, 'mesa',10,100)

d) insert into T(codigo) values (100)

e) insert into T(codigo,produto) values (100,'mesa')

Comentário: A resposta a questão está na alternativa **d) insert into T(codigo) values (100).** Neste comando, você está tentando inserir um valor na coluna "codigo" e deixando em branco as outras colunas, o que resultaria em um erro porque a coluna "produto" é declarada como "not null" e não possui um valor padrão. Portanto, isso causaria um erro de inserção.

Gabarito: D

---

17. **FGV - Analista Legislativo (ALERO)/Tecnologia da Informação/Banco de Dados/2018**

Pessoa	Descendente
Ana	Vitoria

João	Maria
João	Rafael
Maria	Tiago
Natalia	Ana
Rafael	Natalia
Ana	Vitoria

Analise o comando a seguir utilizando a tabela **arvore,** definida anteriormente.

```
delete from arvore
where exists
        (select * from arvore a
          where a.pessoa = 'João'
              and a.descendente = arvore.pessoa)
```

Assinale o número de registros que é removido na execução desse comando.

A Zero.

B Um.

C Dois.

D Três.

E Quatro.

Comentário: Essa questão já figurou em um exemplo da nossa aula. A questão envolve um DELETE que escolhe os elementos de forma correlacionada. Ou seja, você precisa pensar que existem duas tabelas com os mesmos dados e criar um ponteiro para percorrer cada uma delas. O primeiro ponteiro está associado ao termo arvore do comando externo. Já o segundo ponteiro está ligado ao "alias" a da consulta interna. O comando vai escolher uma linha para remover sempre que o atributo pessoa da tabela externa for igual a um dos descendentes que estão na mesma linha que o João. Neste caso, as linhas 4 e 6 seriam deletadas. Se quiser aprender mais sobre a lógica desta questão, volte na parte de consultas aninhada da nossa aula.

Gabarito: C

18. **FGV - Analista Especializado (IMBEL)/Analista de Sistemas/2021 (e mais 1 concurso)**

Considere o comando SQL a seguir, executado num banco de dados relacional com duas tabelas, R1 e R2, contendo 2.000 e 5.000 registros, respectivamente. R1 e R2 possuem chaves primárias definidas.

SELECT DISTINCT * FROM A, B

Assinale a opção que indica o número de linhas produzidas na execução.

A 1

B 2.000

C 5.000

D 7.000

E 10.000.000

Comentário: Questão clássica da FGV, pede para que o candidato calcule a quantidade de linhas geradas por um produto cartesiano. Esse produto pode ser verificado pelos valores A e B, separados por vírgulas, presentes no comando. Neste caso vamos multiplicar a quantidade de linhas da primeira tabela pela quantidade de linhas da segunda (2000x5000) = 10.000.000. Logo, temos a nossa resposta na alternativa E.

Gabarito: E.

19. **FGV - Auditor Fiscal da Receita Estadual (SEFAZ ES)/2021**

Considere um banco de dados relacional contendo as tabelas T, R e S, cujas instâncias são exibidas a seguir.

T

A	B	C
10	LPG Participações	S
20	Serviços & Gerenciamento Remoto	N
50	Academia Americana	S
70	Distribuidora São João de Artigos para Festas	S

R

D	E	F
12040	21/06/2021	200,00
12041	23/06/2021	548,00
1497	15/06/2021	147,10
1498	15/06/2021	85,00
214	18/06/2021	99,00
215	19/06/2021	997,45

S

G	H	I
10	12040	12
10	12041	12
50	1497	12
50	1498	10
70	214	20
50	215	12

Considere a tabela T e a execução dos dois comandos SQL a seguir.

```
SELECT T.*
FROM T LEFT JOIN S ON T.A = S.G
ORDER BY 2

SELECT T.*
FROM T RIGHT JOIN S ON T.A = S.G
ORDER BY 2
```

Sem considerar as linhas de títulos, assinale o número de linhas produzidas por cada comando, na ordem.

A Seis/zero.

B Seis/seis.

C Seis/sete.

D Sete/seis.

E Sete/sete.

Comentário: Para resolver essa questão precisamos lembrar que “outer join” considera todos as linhas da tabela à esquerda (left), à direita (right) e em ambas (full). Assim, podemos perceber que o resultado da primeira consulta apresenta como resultado 7 linhas, e a linha cujo valor de A é igual a 20, mesmo não tendo valor associado na tabela S, aparece no resultado. Já a segunda consulta teremos como resultado as seis linhas da tabela S. Vejamos esses resultados em forma de tabela:

Tabela 4 - Resultado da consulta 01

A	B	C	G	H	I
10	LPG Participações	S	10	12040	12
10	LPG Participações	S	10	12041	12
20	Serviços & Gerenciamento Remoto	N	NULL	NULL	NULL
50	Academia Americana	S	50	1497	12
50	Academia Americana	S	50	1498	10
50	Academia Americana	S	50	215	12
70	Distribuidora São João de Artigos para Festas	S	70	214	20

7

Tabela 5 - Resultado da consulta 02

A	...	G	H	I
10	...	10	12040	12
10	...	10	12041	12
50	...	50	1497	12
50	...	50	1498	10
70	...	70	214	20
50	...	50	215	12

6

Gabarito: D

20. **FGV - Auditor Fiscal da Receita Estadual (SEFAZ ES)/2021**

No contexto da instância da tabela S, considere a execução do comando SQL a seguir.

```
SELECT *
FROM S
WHERE (NOT G=10 OR I=12)
  AND NOT (H > 100 and H < 1000)
```

Assinale o conjunto de linhas que corresponde ao resultado produzido pelo referido comando.

A
```
10   12040   12
10   12041   12
```

B
```
10   12040   12
10   12041   12
50   1497    12
```

C
50 215 12
70 214 20

D
10 12040 12
10 12041 12
50 1497 12
50 1498 10

E
50 1497 12
50 1498 10

Comentário: Ao executar o comando temos que saber que o operador lógico NOT dentro do parêntese tem precedência sobre o OR. Logo primeiramente temos que analisar o NOT G = 10 e, de posse deste resultado (conjunto de linhas que satisfazem a essa condição), podemos executar o OR I = 12. Vejamos os resultados separadamente abaixo:

Tabela 6 - NOT G = 10

G	H	I
10	12040	12
10	12041	12
50	1497	12
50	1498	10
70	214	20
50	215	12

Tabela 7 - T = 12

G	H	I
10	12040	12
10	12041	12
50	1497	12
50	1498	10
70	214	20
50	215	12

Perceba que o resultado de (NOT G = 10 OR I =12) será todas as linhas da tabela. Do outro lado, avaliamos as linhas cujo atributo H não seja maior que 100 e menor que 1000. Neste caso temos 4 linhas.

Tabela 8 - NOT (H > 100 AND H < 1000)

G	H	I
10	12040	12
10	12041	12
50	1497	12
50	1498	10
70	214	20
50	215	12

Agora vamos operar um AND de todas as linhas com essas 4 linhas ... o resultado é a intersecção. A figura abaixo mostra o resultado:

G	H	I
10	12040	12
10	12041	12
50	1497	12
50	1498	10

Gabarito: D

21. **FGV - Auditor Técnico de Controle Externo (TCE-AM)/Auditoria de Tecnologia da Informação/2021**

Na questão, considere uma tabela de banco de dados T cuja instância é exibida a seguir.

A	B	C	D
12	2	3	1
14	3	8	2
18	2	9	3
21	5	4	4

Considerando-se a instância da tabela T (descrita anteriormente), analise o comando SQL abaixo.

select distinct * from T t1, T t2, T t3

A execução desse comando produz um resultado que, além da linha de títulos, contém:

A 8 linhas;

B 24 linhas;

C 32 linhas;

D 64 linhas;

E 128 linhas.

Comentário: Mais uma questão de produto cartesiano. Mais uma questão onde a banca coloca a cláusula DISTINCT para confundir o aluno. Vejam ... se eu tenho 4 linhas distintas e faço a concatenação dessas com 4 linhas distintas, o resultado são 16 linhas distintas. A questão aplica essa lógica duas vezes. Logo, temos 4 x 4 x 4 = 64 linhas distintas. O que leva a nossa resposta para a alternativa d.

Gabarito: D

22. **FGV - Auditor Técnico de Controle Externo (TCE-AM)/Auditoria de Tecnologia da Informação/2021**

Na questão, considere uma tabela de banco de dados T cuja instância é exibida a seguir.

A	B	C	D
12	2	3	1
14	3	8	2
18	2	9	3
21	5	4	4

Considerando-se a instância da tabela T (descrita anteriormente), analise o comando SQL abaixo.

```
update T
set a = a + 32
where
exists (select * from T t2 where T.c > t2.D)
```

O número de registros da tabela T afetados pela execução desse comando é:

A zero;

B um;

C dois;

D três;

E quatro.

Comentário: Essa questão usa a ideia de elementos correlacionados, só que, ao invés de usar a consulta para exibir os dados em tela, vamos atualizar algumas linhas específicas, somando 32 ao valor de A. E quando vamos atualizar? Toda vez que o valor de C for maior do que algum valor em D. Como assim? Lembra da ideia de construir ponteiros? Pois é, com a prática você já faz isso de cabeça. Mas vamos ao passo a passo:

Primeiro você aponta para a primeira instancia da tabela que é definida pelo T da cláusula UPDATE, depois você percorre todas as linhas da segunda instância da tabela comparando o valor de C com todos os valores de D. Se algum valor satisfizer esse predicado significa que a linha em questão será atualizada e ao valor presente na coluna A será adicionado 32. Perceba que isso vai acontecer para todas as linhas da tabela. Logo, quatro linhas serão atualizadas e a nossa resposta está presente na alternativa E.

Gabarito: E

23. **FGV - Auditor Técnico de Controle Externo (TCE-AM)/Auditoria de Tecnologia da Informação/2021**

Na questão, considere uma tabela de banco de dados T cuja instância é exibida a seguir.

A	B	C	D
12	2	3	1
14	3	8	2
18	2	9	3
21	5	4	4

Considerando-se a instância da tabela T (descrita anteriormente), analise o comando SQL abaixo.

```
delete from T where b + d = c
```

O número de registros da tabela T afetados pela execução desse comando é:

A zero;

B um;

C dois;

D três;

E quatro.

Comentário: Depois que você aprende a resolver questões com consultas correlacionadas, questões mais diretas como essa começam a parecer mais simples. Veja que será excluída da tabela todas as linhas onde o valor presente na coluna C seja igual a somas dos valores de B e D. A única linha que satisfaz essa condição é a primeira, onde 2 + 1 = 3. Logo, essa única linha será deletada da tabela. E a nossa resposta encontra-se na alternativa B.

Gabarito: B

---

24. **FGV - Técnico Superior Especializado (DPE RJ)/Tecnologia da Informação/2019**

FAMILIA

pessoa1	pessoa2	relação
João	Rafael	pai
Maria	Rafael	mãe
Rafael	Gabriela	pai
Gabriela	Rita	mãe
Rita	Bruna	mãe
Bruna	Ana	mãe
Rafael	Rita	avo

Em cada registro, a relação entre a primeira e a segunda pessoa é descrita. Por exemplo, João é pai de Rafael, Gabriela é mãe de Rita, e Rafael, por sua vez, é avô/avó de Rita. Nem todas as relações de avô/avó estão registradas na tabela.

Considere a tabela FAMILIA descrita anteriormente e o comando SQL a seguir.

```
select f1.pessoa1, f2.pessoa2
from família f1, família f2
where f1.pessoa2 = f2.pessoa1
and f1.relação in ('mãe','pai')
and f2.relação in ('mãe','pai')
union
select pessoa1, pessoa2
from familia
where relação = 'avo'
```

Além dos títulos, o número de linhas exibidas na execução desse comando é:

A 1;

B 2;

C 3;

D 4;

E 5.

Comentário: Essa questão irá fazer a união de duas consultas. Logo podemos avaliá-las separadamente. Começando pela primeira consulta:

```
select f1.pessoa1, f2.pessoa2
from família f1, família f2
where f1.pessoa2 = f2.pessoa1
and f1.relação in ('mãe','pai')
and f2.relação in ('mãe','pai')
```

pessoa1	pessoa2	relação
João	Rafael	pai
Maria	Rafael	mãe
Rafael	Gabriela	pai
Gabriela	Rita	mãe
Rita	Bruna	mãe
Bruna	Ana	mãe
Rafael	Rita	avo

pessoa1	pessoa2	relação
João	Rafael	pai
Maria	Rafael	mãe
Rafael	Gabriela	pai
Gabriela	Rita	mãe
Rita	Bruna	mãe
Bruna	Ana	mãe
Rafael	Rita	avo

Cada linha da tabela será associada a todas as linhas da outra tabela por conta do produto cartesiano presente na cláusula FROM. Depois vamos avaliar o predicado, perceba que só vamos retornar uma linha quando todas as condições forem verdadeiras por conta do operador lógico “and”. Para a primeira linha de f1 vamos retornar apenas a tupla:

f1.pessoa1	f2.pessoa2
João	Gabriela

Se aplicarmos a mesma lógica acima para todas as linhas de f1, ficaremos com o seguinte resultado:

f1.pessoa1	f2.pessoa2
João	Gabriela
Maria	Gabriela
Rafael	Rita
Gabriela	Bruna
Rita	Ana

Acho que você já entendeu que essa consulta retorna os pares de avô/avó, neto/neta. Mas ainda não acabou ... temos que executar a segunda consulta:

```
select pessoa1, pessoa2
from familia
where relação = 'avo'
```

Nesta consulta temos o seguinte resultado:

f1.pessoa1	f2.pessoa2
Rafael	Rita

Agora vamos aplicar a operação de união entre os dois resultados acima. Sabemos que as operações de conjunto eliminam as tuplas duplicadas, logo ficamos com 5 linha no resultado:

f1.pessoa1	f2.pessoa2
João	Gabriela
Maria	Gabriela
Rafael	Rita
Gabriela	Bruna
Rita	Ana

Desta forma temos o gabarito na alternativa E.

Gabarito: E

---

25. **FGV - Técnico Superior Especializado (DPE RJ)/Tecnologia da Informação/2019**

FAMILIA

pessoa1	pessoa2	relação
João	Rafael	pai
Maria	Rafael	mãe
Rafael	Gabriela	pai
Gabriela	Rita	mãe
Rita	Bruna	mãe
Bruna	Ana	mãe
Rafael	Rita	avo

Em cada registro, a relação entre a primeira e a segunda pessoa é descrita. Por exemplo, João é pai de Rafael, Gabriela é mãe de Rita, e Rafael, por sua vez, é avô/avó de Rita. Nem todas as relações de avô/avó estão registradas na tabela.

Considere a tabela FAMILIA descrita anteriormente e o comando SQL a seguir.

```
select relação, sum(1)
from familia
group by relação
having count(*) > 1
order by 2 desc, 1
```

Os valores exibidos pela execução desse comando, na ordem, são:

A mãe 4
pai 2
avo 1

B mãe 2
pai 4

C pai 2
mãe 4

D mãe 4
pai 2

E mãe 4
pai 2
avo 0

Comentário: A dificuldade da questão está em entender que a função sum(1) soma o valor 1 para cada linha, de forma que ela se comporta da mesma maneira que a função count(*).

A cláusula ORDER BY também é utilizada de maneira não muito usual: quando temos números na cláusula, eles se referem à posição da coluna que está no SELECT. Assim, "ORDER BY 2 desc, 1" faz com que os resultados sejam ordenados pela segunda coluna (sum) em ordem decrescente e pela primeira coluna (relação) em ordem crescente.

Dessa forma, o comando proposto está retornando a relação e a quantidade de ocorrências de cada relação na tabela família, apenas das relações que tiverem mais de 1 ocorrência, ordenadas em ordem decrescente de quantidade de ocorrências e, caso tenham a mesma quantidade, será ordenada pelo nome da relação em ordem alfabética crescente.

Ao olhamos para a tabela, percebemos que temos 2 relações pai, 4 relações mãe e 1 relação avo. Como avo só tem uma ocorrência, ela não aparecerá no resultado. O resultado é ordenado em ordem decrescente de quantidade, de forma que será exibido:

mãe 4

pai 2

Concluímos, assim, que o gabarito é letra D.

Gabarito: D

---

26. **FGV - Auditor Fiscal de Tributos Estaduais (SEFIN RO)/2018**

Considere as tabelas de bancos de dados T1, T2 e T3, que contêm, respectivamente, 10, 500 e 2.000 registros, e o comando SQL a seguir.

select count(*) FROM T1, T2, T3

Assinale a opção que apresenta o número exibido no resultado da execução desse comando.

A 10000000

B 1000000

C 2000

D 500

E 10

Comentário: Mais uma questão de produto cartesiano. Temos 3 relações com 10, 500 e 2000 linhas, o resultado da consulta vai contar a quantidade de linhas que é obtida pelo produto dos valores 10 x 500 x 2000 = 10000000. Resposta na letra A.

Gabarito: A

---

27. **FGV - Analista Legislativo Municipal (CM Salvador)/Tecnologia da Informação/2018**

Uma tabela de banco de dados denominada TTT, com atributos A, B e C, contém em sua instância sete registros, com os seguintes valores:

1,2,1

2,3,4

5,4,4

8,7,5

4,3,2

2,3,4

4,3,2

O número de registros removidos pela execução do comando

```
delete from TTT
where exists
    (select * FROM TTT t2
     where TTT.a = t2.a
      and TTT.b = t2.b)
```

seria:

A 3;

B 4;

C 5;

D 6;

E 7.

Comentário: Veja que essa questão tenta criar uma dificuldade, mas não é complexa. Se você já consegue enxergar a ideia de elementos correlacionados e a lógica de que sempre vai haver uma tupla em t2 onde TTT.a = t2.a e TTT.b=t2.b, porque as duas tabelas têm as mesmas linhas. Então, todas as linhas serão removidas. Como temos 7 linhas na tabela, a nossa resposta encontra-se na letra E.

Gabarito: E

---

28. **FGV - Analista Legislativo Municipal (CM Salvador)/Tecnologia da Informação/2018**

Uma tabela de banco de dados denominada TT, com atributos A e B, contém em sua instância seis registros, com os seguintes valores:

1,2

2,3

3,4

3,7

4,3

2,3

O número de registros alterados pela execução do comando

```
update TT set b = b + 1
where TT.a in (select b FROM TT)
```

seria:

A 1;

B 2;

C 3;

D 4;

E 5.

Comentário: Questão tranquila, vamos somar 1 ao valor da coluna A sempre que o valor analisado existir na coluna B. Começando pela primeira linha observamos que o valor 1 não existe na coluna B, logo, não será atualizado. As demais linhas, porém, possuem o valor de A presente na coluna B, o que faz com que a coluna A seja atualizada. Ao final da execução do comando teríamos o seguinte resultado.

1,2

3,3

4,4

5,7

5,3

3,3

Assim, 5 linhas foram atualizadas, o que nos leva a resposta na alternativa E.

Gabarito: E

29. **FGV - Analista de Políticas Públicas e Gestão Governamental (CGM Niterói)/Gestão de Tecnologia/2018**

A otimização de consultas em gerenciadores de bancos de dados é fundamental para o desempenho do sistema. Consultas escritas em SQL são particularmente propícias à otimização, porque essa linguagem

A não é procedural, permitindo a criação de diferentes planos de execução.

B tem uma sintaxe simples e é largamente utilizada.

C suporta todas as operações da Álgebra Relacional.

D permite o uso de subconsultas, facilitando os processos de busca.

E permite a criação de camadas de software de persistência.

Comentário: o fato de não ser uma linguagem procedural e sim uma linguagem declarativa permite que haja vários caminhos para chegar na mesma consulta, ou seja, vários planos de execução, podendo o SGBD escolher o que tem a melhor performance. Logo, temos a resposta na letra A.

Gabarito: A

30. FGV - Analista de Tecnologia da Informação (BANESTES)/Desenvolvimento de Sistemas/2018

Considere um banco de dados com duas tabelas, R e S, cujas instâncias são exibidas a seguir.

R

a	b
1	2
2	3
4	5

S

c	d
3	2
4	2
6	1

Na execução do comando SQL

select * from R left join S on a=b

UNION

select * from R right join S on b=a

o número de células contendo o valor nulo no resultado é:

A 0;

B 3;

C 6;

D 9;

E 12.

Comentário: Primeiramente, vamos executar as duas seleções de forma independente.

select * from R left join S on a=b;

A	B	C	D
1	2	–	–
2	3	–	–
4	5	–	–

Perceba que não haverá registro que satisfaça a condição de junção. Logo, os valores presentes nas colunas que vieram da tabela S ficam em branco, ou preenchidos com valores nulos.

select * from R right join S on b=a

A	B	C	D
–	–	3	2
–	–	4	2
–	–	6	1

Agora, vamos aplicar a operação de UNIÃO sobre os resultados acima, desta forma, temos que:

A	B	C	D
1	2	–	–
2	3	–	–
4	5	–	–
–	–	3	2
–	–	4	2
–	–	6	1

Observamos que, na tabela de resultado, 1**2 células possuem o valor nulo,.**

Gabarito: E.

31. **FGV - Analista de Tecnologia da Informação (BANESTES)/Desenvolvimento de Sistemas/2018**

Considere um banco de dados com duas tabelas. A primeira tabela, números, possui dez registros e apenas uma coluna, cujos valores são 1, 2, 3, 4, 5, 5, 9, 9, 9, 10. A segunda tabela, denominada teste, com cinco registros, também possui apenas uma coluna, cujos valores são 1, 3, 3, 4, 5.

Considere ainda o seguinte comando SQL

```
insert into teste
select numero from numeros n
where not exists
  (select * from teste t
   where t.numero = n.numero)
```

Quando da execução desse comando, o número de registros inseridos na tabela teste é:

A 2;

B 3;

C 5;

D 8;

E 10.

Comentário: O que o comando acima faz é inserir na tabela teste todos os valores da tabela números que não estão na tabela teste. Perceba que, seguindo a lógica dos ponteiros o select externo irá retornar uma linha toda vez que seu valor não for encontrado na consulta interna. Quando a consulta interna retorna vazio, o not exists avalia esse resultado como verdadeiro. Em seguida o valor é inserido na tabela de teste. Logo, os valores 2, 9, 9, 9 e 10 são inseridos, o que corresponde a 5 registro.

Gabarito: C

---

32. **FGV - Analista de Tecnologia da Informação (BANESTES)/Desenvolvimento de Sistemas/2018**

Considere uma tabela de bancos de dados com dez registros, e apenas uma coluna cujos valores são 1, 2, 2, 3, 3, 3, 4, 4, 4, 4.

Requisitado para remover os registros com valores repetidos dessa tabela, um programador produziu um script com dois comandos.

```
delete from exemplo
where exists (select * from exemplo e1
        where exemplo.x = e1.x)
select count(distinct x) from exemplo
```

Na execução desse script, o número produzido no segundo comando foi:

A 0;

B 1;

C 2;

D 3;

E 4.

Comentário: O comando delete acima vai apagar todas as linhas da tabela exemplo, e seguida o SELECT retornará zero

Gabarito: A

---

33. **Ano: 2016 Banca: FGV Órgão: IBGE Cargo: WEB MOBILE Questão: 45**

Os comandos SQL

```
create table R (a int, b int)
create table S (c int, d int)
insert into R values(1,2)
insert into R values(2,3)
insert into R values (2,3)
insert into R values (3,5)
insert into R values (4,1)
insert into S values (1,2)
insert into S values (2,1)
insert into S values (2,3)
insert into S values (3,5)
select r.a, r.b from R
where not exists
 (select * from S where s.c=r.a and s.d=r.b)
```

Produzem um resultado que, além da linha de títulos, contém:

(A) uma linha;

(B) duas linhas;

(C) três linhas;

(D) quatro linhas;

(E) cinco linhas.

Comentário: Após a execução dos comandos de insert acima temos os seguintes valores nas tabelas R e S.

R (1,2) (2,3) (3,5) (4,1)

S (1,2) (2,1) (2,3) (3,5)

A consulta pede que para cada elemento do conjunto R, verificarmos se o resultado da segunda consulta é vazio, caso seja, retorne no resultado da consulta. A consulta interna verifica de o valor do campo a de R é igual ao valor do campo c de S, e se o valor do campo b de R é igual ao valor do campo d de S. Vejam que se existir essa igualdade o valor será retornado na consulta interna, o que impede que a consulta externa seja verdadeira. Em outras palavras, estamos procurando os pares de R que não estão em S, ou seja, o par (4,1).

Gabarito: A.

---

34. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: WEB MOBILE Questão: 46

O comando SQL

select a, sum(b) x, COUNT(*) y

from T

group by a

produz como resultado as linhas abaixo.

a	x	y
1	6	1
3	6	2
4	4	1
5	1	1

Na tabela T, composta por duas colunas, a e b, nessa ordem, há um registro duplicado que contém os valores:

(A) 1 e 3

(B) 3 e 3

(C) 3 e 6

(D) 4 e 2

(E) 5 e 1

Comentário: Essa é uma questão interessante. Vejam que os valores mostrados na tabela é o resultado da consulta. Desta forma sabemos que a coluna x é o resultado de uma soma e que a coluna y é o resultado da contagem dos elementos com o mesmo valor, ou seja, das linhas duplicadas. Podemos perceber que o registro duplicado é o que tem valor de y maior do que 1. E que o para esse registro para a coluna "a" é 3, que é o mesmo valor que aparece na função de agrupamento; e b é também igual a 3 pois o valor da coluna x é a soma dos valores dos dois registros. Devemos portando dividir o valor por 2. Assim, chegamos a nossa resposta na alternativa B.

Gabarito: B.

---

35. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 31

Atenção: Algumas das questões seguintes fazem referência a um banco de dados relacional intitulado BOOKS, cujas tabelas e respectivas instâncias são exibidas a seguir. Essas questões referem-se às instâncias mostradas.

AUTOR

AutorID	AutorNome
1	Arthur Conan Doyle
2	Agatha Christie
3	Edgar Allan Poe

LIVRARIA

LivrariaID	LivrariaNome
1	Cultural
2	Travessia
3	Amazonas
4	Kremlin

LIVRO

LivroID	AutorID	Titulo	NumLivrarias
1	1	O Cão dos Baskervilles	NULL
2	1	As Aventuras de Sherlock Holmes	2
3	2	Assassinato no Expresso do Oriente	2
4	2	O Mistério dos Sete Relógios	3
5	3	Assassinatos na Rua Morgue	NULL

OFERTA

LivrariaID	LivroID	Preco
1	1	32
1	2	28
1	3	45
1	4	38
1	5	23
2	1	56
2	2	54
2	4	43
3	3	35
3	4	38

A tabela Livro representa livros. Cada livro tem um autor, representado na tabela Autor. A tabela Oferta representa os livros que são ofertados pelas livrarias, estas representadas pela tabela Livraria. NULL significa um campo não preenchido.

AutorID, LivrariaID e LivroID, respectivamente, constituem as chaves primárias das tabelas Autor, Livraria e Livro.

LivrariaID e LivroID constituem a chave primária da tabela Oferta.

Com relação ao banco de dados BOOKS, analise os comandos SQL exibidos a seguir:

I. select *

from oferta o, livro l, autor a, livraria ll

where o.livroid=l.livroid and

o.livrariaid=ll.livrariaid and l.autorid=a.autorid

II.select *

from oferta o inner join livro l on

o.livroid=l.livroid

inner join autor a on l.autorid=a.autorid

inner join livraria ll on

o.livrariaid=ll.livrariaid

III. select *

from oferta o left join livro l on

o.livroid=l.livroid

left join autor a on l.autorid=a.autorid

left join livraria ll on

o.livrariaid=ll.livrariaid

É correto afirmar que:

(A) somente I e II produzem resultados equivalentes;

(B) somente I e III produzem resultados diferentes;

(C) somente II e III produzem resultados diferentes;

(D) todos os resultados são diferentes;

(E) todos os resultados são equivalentes.

Comentário: A grande dificuldade dessa questão é observar que todas elas executam uma junção. Na primeira alternativa temos a utilização de um produto cartesiano seguido por uma seleção das colunas nas quais os valores da chave primárias são iguais. Na segunda alternativa temos a utilização do **inner join** que implicitamente já realiza as duas operações da alternativa I. Por fim, temos a alternativa II, nela temos que perceber que toda oferta tem um livro associado, que por sua vez tem um autor e uma livraria. Neste encadeamento, podemos observar que não temos a possibilidade de um registro na tabela oferta que não ter correspondência na tabela livro. Desta forma, podemos observar que as três alternativas produzem resultados equivalentes.

Gabarito: E.

---

36. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 33

No banco de dados BOOKS, o campo NumLivrarias, da tabela Livro, contém informação redundante, pois denota o número de livrarias que oferecem o livro e pode ser computado. O comando SQL que calcula e atualiza esse campo corretamente é:

(A) update livro

set numlivrarias=

(select count(*) from oferta o where

o.livroid=livro.livroid)

where numlivrarias=null

(B) update livro

set numlivrarias=

(select count(*) from oferta o where

o.livroid=livro.livroid)

(C) update numlivrarias

from livro as

(select count from oferta o where

o.livroid=livro.livroid)

where numlivrarias=null

(D) set livro.numlivrarias=

(select count from oferta o where

o.livroid=livro.livroid)

(E) set livro.numlivrarias=

(select count(livrariaid) from oferta o where

o.livroid=livro.livroid)

where numlivrarias=null

Comentário: Precisamos entender que a informação sobre a quantidade de livrarias nas quais cada livro é vendido está presente na tabela oferta. Precisamos então contar a quantidade de linhas da tabela oferta que tenha o id do livro igual ao id da oferta. Desta forma, percebemos que nossa resposta está na alternativa B.

Como exercício, sugiro que você procure os erros das demais alternativas.

Gabarito: B.

---

37. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 34

Quando executado no contexto do banco de dados BOOKS, o comando SQL

select numlivrarias from livro

where numlivrarias > 0

union

select numlivrarias from livro

where numlivrarias <= 0

produz um resultado cujo número de linhas, além da linha de título, é:

(A) 1

(B) 2

(C) 3

(D) 4

(E) 5

Comentário: Vejam que o atributo da tabela livro denominado numlivrarias possui valores iguais a null, 2, 2, 3 e null. As consultas acima vão retornar respectivamente o conjunto (2, 2, 3) e vazio. Agora vamos aplicar a operação de união sobre esses dois conjuntos. Esta operação vai fazer com que um dos elementos 2 que está duplicado seja removido da resposta. Logo teremos apenas **duas linhas** na resposta.

Gabarito: B.

---

38. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 41

João foi incumbido de rever um lote de consultas SQL. Como ainda é iniciante nesse assunto, João solicitou ajuda ao colega que lhe pareceu ser o mais experiente, e recebeu as seguintes recomendações gerais:

I. use a cláusula DISTINCT somente quando estritamente necessária;

II. dê preferência às junções externas (LEFT, RIGHT, OUTER) em relação às internas (INNER);

III. use subconsultas escalares no comando SELECT, tais como “SELECT x,y,(SELECT ...) z ..." sempre que possível.

Sobre essas recomendações, é correto afirmar que:

(A) nenhuma é adequada;

(B) somente I é adequada;

(C) somente I e II são adequadas;

(D) somente II e III são adequadas;

(E) todas são adequadas.

Comentário: Vejamos cada uma das sugestões:

I. Neste caso temos uma sugestão coerente. O uso do DISTINCT exige a execução de um algoritmo para remover registros duplicados. Para tal, é bastante possível que seja necessária alguma ordenação sobre o resultado, isso pode atrasar o processamento da consulta.

II. As junções internas são mais leves que as consultas externas, sendo processadas mais rapidamente. Portanto, não faz sentido dar preferências às junções externas.

III. Usar subconsultas com valores escalares prejudica o desempenho do SGBD pois obriga o armazenamento dos valores escalares em alguma memória temporária. Isso também deve ser evitado.

Analisando os comentários, apenas a **alternativa I está correta**.

Gabarito: B.

---

39. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: SUPORTE OPERACIONAL Questão: 42

João escreveu a consulta SQL a seguir, executou-a corretamente e obteve um resultado contendo 100 linhas, além da linha de títulos.

SELECT curso, nome

FROM aluno, curso

WHERE aluno.codcurso = curso.codcurso

ORDER BY curso, nome

As tabelas aluno e curso possuem, respectivamente, 120 e 12 linhas. No banco há ainda outras duas tabelas, pauta e disciplina, com 200 e 5 registros, respectivamente. Nessas condições, o número de linhas, além da linha de títulos, produzidas pelo comando

SELECT curso, nome

FROM aluno, curso, disciplina, pauta

WHERE aluno.codcurso = curso.codcurso

ORDER BY curso, nome

seria:

(A) 100;

(B) 305;

(C) 500;

(D) 20.000;

(E) 100.000.

Comentário: Vejam que a primeira informação relevante para resolvermos a questão é o fato da primeira consulta retornar 100 linhas. Em seguida, vamos aplicar a essas 100 linhas um produto cartesiano com a tabela pauta e, subsequentemente com a tabela disciplina. Sendo assim vamos ter no nosso resultado a seguinte equação:

100 (primeira query) x 200 (pauta) x 5 (disciplina) = 100.000

Assim o nosso resultado possui 100 mil linhas.

Gabarito: E

---

40. BANCA: FGV ANO: 2014 ÓRGÃO: CM-RECIFE PROVA: ANALISTA LEGISLATIVO - ANALISTA DE SISTEMAS

No SQL, o comando grant permite outorgar a um usuário (ou papel) privilégios sobre determinados recursos. Quando usado com a opção with grant option, o comando grant permite que:

A no caso de privilégios outorgados a papéis seja possível identificar usuários que excepcionalmente não devem receber esses privilégios;

B os privilégios outorgados não possam ser alvo de comandos revoke emitidos por usuários diferentes daquele que outorgou incialmente;

C os privilégios sejam outorgados em caráter temporário, sendo automaticamente removidos quando da expiração do prazo estabelecido;

D o usuário que recebe um privilégio possa concedê-lo a outros usuários;

E os privilégios sejam concedidos condicionalmente, e posteriormente confirmados mediante a execução automática de um procedimento de autorização.

Comentário: Utilizar a opção WITH GRANT OPTION indica que o usuário autorizado também poderá conceder a permissão especificada a outras entidades. Vimos isso quando tratamos de segurança em SQL.

Gabarito D.

41. BANCA: FGV ANO: 2015 ÓRGÃO: CM CARUARU PROVA: ANALISTA LEGISLATIVO - INFORMÁTICA

Em geral, a definição de chaves estrangeiras em bancos de dados relacionais pode vir acompanhada da especificação de procedimentos adicionais a serem adotados quando da exclusão/alteração de valores nos registros da tabela estrangeira.

```
create table R2 (
        a int not null primary key,
        b int null,
        x int not null,
        constraint FK foreign key (x) references
R1(x)
)
```

Considerando o script acima, as opções complementares compatíveis para a definição da chave estrangeira FK são:

A on delete cascade

on update set null

B on delete cascade

on update cascade

C on delete no action

on update set null

D on delete set null

on update restrict

E on delete restrict

on update set null

Comentário: A alternativa que faz mais sentido é a letra B. Não é possível utilizar a cláusula **on update set null,** pois o campo x da tabela R2 foi definido como "not null". Com os comandos da letra B, sempre que um registro de R1 tiver o valor de x alterado, essa alteração será refletida em R2. Se o registro for deletado em R1, também será excluído de R2.

Gabarito B.

42. ANO: 2015 BANCA: FGV ÓRGÃO: TCE-SE PROVA: ANALISTA DE TECNOLOGIA DA INFORMAÇÃO - SEGURANÇA DA INFORMAÇÃO

Considere duas tabelas X e Y, com as seguintes instâncias:

X	
a	b
1	2
3	3
4	5
5	7

Y	
c	d
1	2
3	4
5	6
7	8
9	1

O comando SQL que retorna

a	b	c	d
1	2	1	2
3	3	3	4
4	5	NULL	NULL
5	7	5	6
NULL	NULL	7	8
NULL	NULL	9	1

é:

A
```
select *
from X FULL JOIN Y on X.a=Y.c
```

B
```
select *
from X LEFT JOIN Y on X.a=Y.c
```

C
```
select *
from Y RIGHT JOIN X on X.a=Y.c
```

D
```
select *
from X CROSS JOIN Y on X.a=Y.c
```

E
```
select *
from X INNER JOIN Y on X.a=Y.c
```

Comentário: Vejam que esse é um exemplo de um FULL OUTER JOIN. Ele considera todas as colunas das tabelas X e Y. Os atributos de junção são a,b em X e c,d em Y. Desta forma, quando não existe correspondente na outra relação os valores são preenchidos com NULL.

Gabarito A.

---

43. Ano: 2013 Banca: FGV Órgão: SUDENE-PE Cargo: Analista Técnico Administrativo - Ciência da Computação

Com relação aos bancos de dados, os índices são uma das técnicas mais utilizadas na otimização de desempenho de consultas SQL.

A respeito dos índices, assinale V para a afirmativa verdadeira e F para a falsa.

(...) Os índices provavelmente serão utilizados quando uma coluna indexada aparecer nos critérios de busca de uma cláusula WHERE ou HAVING.

(...) Os índices provavelmente serão utilizados quando uma coluna indexada aparecer em uma cláusula GROUP BY e ORDER BY.

(...) Os índices provavelmente serão utilizados quando a seletividade dos dados de uma coluna indexada for baixa.

As afirmativas são, respectivamente,

A) F, V e F.

B) V, V e F.

C) V, F e F.

D) V, F e V.

E) F, F e V.

Comentário: Falaremos mais sobre essa questão de otimização na aula de tuning. Mas vamos comentar cada uma das alternativas acima.

I. Na primeira afirmativa temos uma demonstração consistente do uso de índices, quando eles aparecem nas cláusulas WHERE e HAVING, nestes casos eles vão agilizar o processamento da consulta.

II. Vejam que se o índice for usado numa clausula group by ou order by, por princípio, os valores estarão armazenados em posições próximas no disco rígido, isso vai permitir uma quantidade menor de operações de I/O, que consequentemente vai melhorar a performance das operações.

III. Primeiro precisamos entender a definição de seletividade. Trata do percentual de linhas em uma coluna que possuem o mesmo valor. Uma coluna com seletividade de 5% e uma coluna com boa seletividade, por apenas 5% dos valores são repetidos. Se um índice filtra muitos os dados, dizemos que sua seletividade é alta. Do outro lado, se um índice filtra pouco os dados, podemos concluir que sua seletividade é baixa. Geralmente estamos interessados em índices de alta seletividade, de forma que essa restrição agilize a nossa consulta.

Vejam que a afirmação III vai no sentido contrário do que se espera uma seletividade baixa não ajuda, logo a alternativa está incorreta.

Gabarito: B.

---

44. Ano: 2013 Banca: FGV Órgão: MPE-MS Cargo: Técnico - Informática

Observe o comando SQL a seguir:

SELECT nome, sobrenome, PIS, anos_de_servico FROMEmpregados

A cláusula que deve ser adicionada ao comando acima para ordenar os registros por anos de serviço, com os empregados que estão há mais tempo na empresa aparecendo primeiro na listagem, é

A) ORDER 'anos_de_servico' BY ASC

B) ORDER BY 'anos_de_servico'

C) ORDER BY anos_de_servico DESC

D) SORTED BY anos_de_servico DESC

E) ORDER BY anos_de_servico ASC

Comentário: Percebam que quanto maior o tempo de empresa maior será a quantidade de anos de serviço. Desta forma, como a questão pede que os empregados a mais tempo na empresa apareçam primeiro devemos ordenar de forma decrescente. Outro ponto importante para responder à questão é utilizar a sintaxe correta de ordenação em SQL, qual seja, ORDER BY nome_do_campo DESC. Desta forma, podemos marcar o gabarito correto na alternativa C.

Gabarito: C.

---

45. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas

A necessidade de construir consultas aplicadas a sistemas de apoio à decisão levou à introdução de algumas construções especiais na linguagem SQL, que facilitam e estendem a agregação de dados. Dentre essas estão:

A) oltp e olap;

B) intersection e minus;

C) rank e decode;

D) cube e rollup;

E) slicing e dicing.

Comentário: Em SQL, podemos utilizar operadores ROLLUP e CUBE na instrução SELECT para totalizações. O operador ROLLUP serve para mostrar que podemos obter um total geral, permitindo agregações sobre grupos de linhas. O ROLLUP não precisa que se especifique nenhum campo e ele age em conjunto com o GROUP BY e a função de agregação.

A cláusula CUBE é uma extensão de ROLLUP que permite as agregações totais por todas as colunas envolvidas. Na prática cria um cubo. O exemplo abaixo determina a soma de salários usando GROUP BY CUBE (JOB, DEPTNO):

```
select job, deptno, sum(sal)
from emp
group by cube(job, deptno)
order by job, deptno;
```

```
JOB       DEPTNO               SUM(SAL)
--------- -------------------- --------------------
ANALYST   20                   6000
ANALYST                        6000
CLERK     10                   1300
CLERK     20                   1900
CLERK     30                   950
CLERK                          4150
MANAGER   10                   2450
MANAGER   20                   2975
MANAGER   30                   2850
MANAGER                        8275
PRESIDENT 10                   5000
PRESIDENT                      5000
SALESMAN  30                   5600
SALESMAN                       5600
          10                   8750
          20                   10875
          30                   9400
                               29025

18 rows selected
```

Vejam que a opção que apresenta as duas instruções de agregação descritas por SQL é a presente na alternativa D.

Gabarito: D.

---

46. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas

Considere as seguintes tabelas relacionais e respectivas instâncias.

R

A	B
1	3
2	3
3	4
5	5
5	7
8	8

S

C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

SELECT A,

(**SELECT COUNT** (*)

FROM S **WHERE NOT** R.B = S.C **OR** R.A = S.D

) X **FROM** R

ORDER BY A;

Os números que aparecem na coluna X do resultado da execução desse comando, de cima para baixo, são:

A) 0, 0, 1, 0, 0, 0.

B) 0, 0, 0, 0, 0, 0.

C) 2, 2, 5, 5, 5, 5.

D) 2, 2, 4, 5, 5, 5.

E) 3, 3, 1, 0, 0, 0.

Comentário: A questão pede para que seja contabilizada a quantidade de registros que atendem a seguinte caraterísticas: **NOT** R.B = S.C **OR** R.A = S.D.

Vejamos o caso prático para a primeira linha da tabela R (1,3), para cada registro de S vamos verificar os dois predicados, e proceder uma operação lógica de OR entre os dois.

Na primeira linha de S temos (1, 9), vejam que R.B(3) != S.C(1), isso torna o primeiro predicado verdadeiro; e R.A(1) != S.D (9), que torna o segundo predicado falso. Então fazemos um OR entre os dois. Neste caso podemos contabilizar 1 no somatório.

Fazemos o mesmo para todas as linhas da tabela S e obtemos o resultado 2 para a primeira coluna X do resultado. Após aplicar essa lógica para todas as linhas podemos encontrar os seguintes valores para X (2, 2, 5, 5, 5, 5).

Gabarito: C.

---

47. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas

Considere as seguintes tabelas relacionais e respectivas instâncias.

R

A	B
1	3
2	3
3	4
5	5
5	7
8	8

S

C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

```
DELETE FROM S
WHERE NOT EXISTS
        (SELECT * FROM R
         WHERE R.A = S.C AND R.B = S.D)
```

O número de registros deletados por esse comando é:

A) 0;

B) 1;

C) 2;

D) 4;

E) 5.

Comentário: Vejam que essa questão trata de uma consulta correlacionada. Vamos excluir o registro de S toda vez que o conjunto de linhas da consulta interna for vazia. De forma simples, vamos apagar as tuplas de S que não tenham o valore em R. No caso concreto, a primeira linha de S será removida.

Gabarito: B.

---

48. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas

Considere as seguintes tabelas relacionais e respectivas instâncias.

R

A	B
1	3
2	3
3	4
5	5
5	7
8	8

S

C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

SELECT * FROM R UNION SELECT * FROM S

O número de linhas produzidas por esse comando, excetuada a linha de títulos de colunas, é:

A) 2;

B) 5;

C) 6;

D) 7;

E) 11.

Comentário: Questão interessante para lembrarmos que as operações com conjuntos elas vão eliminar as tuplas duplicadas. Desta forma, o nosso resultado terá apenas 7 linhas. Para a operação com conjuntos não eliminarem as tuplas duplicadas precisamos utilizar a cláusula ALL: UNION ALL, por exemplo.

Gabarito: D.

---

49. Ano: 2015 Banca: FGV Órgão: DPE-RO Cargo: Analista de redes e comunicação de dados

Observe o comando SQL a seguir.

UPDATE X SET Y = 'Z';

Para que esse comando esteja corretamente formulado, quando analisado isoladamente, pressupõe-se que:

A) Y seja uma coluna da tabela X;

B) X seja uma coluna da tabela Y;

C) X e Y sejam tabelas;

D) X seja um banco de dados e Y seja uma tabela;

E) Y seja uma coluna da tabela X e Z seja o nome de um tipo de dados válido.

Comentário: Veja que esta é uma questão relativamente simples. Ela procura medir nosso conhecimento a respeito da sintaxe do comando UPDATE. Observe que X deve ser o nome de uma tabela e Y um atributo desta tabela, pela forma como o argumento é passado podemos supor que é um atributo textual.

Gabarito: A.

---

50. Ano: 2015 Banca: FGV Órgão: DPE-RO Cargo: Analista de redes e comunicação de dados

Analise o comando SQL a seguir.

SELECT DISTINCT 1 FROM X

Sabendo-se que a instância da tabela X não é vazia, conclui-se que a execução desse comando produz um resultado com:

A) apenas uma linha;

B) um conjunto de linhas contendo o valor NULL;

C) um conjunto de linhas contendo todos os atributos de X;

D) um número de linhas igual ao número de diferentes valores do primeiro atributo de X;

E) um número de linhas igual ao número de registros de X.

Comentário: Essa questão testa o nosso conhecimento prático sobre SQL. Você saber o que acontece se executarmos o comando: "SELECT *, 1 FROM TabelaX;"?

Basicamente o banco de dados vai te retornar todas as linhas e colunas da TabelaX e adicionar uma nova coluna com nome 1 e todos os valores iguais a 1. Você pode testar isso na prática aqui. Agora que você já sabe que todos os valores da coluna 1 são iguais, o que acontece se utilizarmos a cláusula distinct? Justamente! Teremos como resultado apenas **uma linha cujo valor é 1**.

Gabarito: A.

---

51. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação

X	
a	b
1	2
3	3
4	5
5	7

Y	
c	d
1	2
3	4
5	6
7	8
9	1

delete from y

where y.c in

(select a from x union select c from y)

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que o número de registros removidos da tabela Y pela execução desse comando é:

A) 1

B) 2

C) 3

D) 4

E) 5

Comentário: Vejam que a questão vai fazer uma deleção de todas as linhas da tabela Y. Vamos começar verificando o resultado retornado pela consulta interna. Ela, basicamente, junta todos os valores de a em X com os de c em Y, resultando em (1, 3, 4, 5, 7, 9). Depois vamos deletar todas das linhas de Y cujos valores de c estejam nesta lista. Ou seja, apagaremos todas as **cinco** linhas de Y.

Gabarito: E.

---

52. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação

X	
a	b
1	2
3	3
4	5
5	7

Y	
c	d
1	2
3	4
5	6
7	8
9	1

SELECT X.a **FROM** X

WHERE NOT EXISTS

(SELECT * FROM Y WHERE Y.c = X.a + 1)

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que esse comando produz um resultado com uma única coluna contendo somente o (s) valor (es):

A) 4

B) 3, 4

C) 1, 3, 5

D) 3, 4, 5

E) 1, 3, 4, 5

Comentário: A questão permite observarmos o processo de avaliação de uma consulta correlacionada. Vamos utilizar dois cursores imaginários para analisar a questão, um cursor C1 que aponta para a primeira linha de X e um cursor C2 que aponta para a primeira linha de Y. No início do processo vamos verificar qual o valor atual de C1, no caso (1, 2). De posse desse valor vamos verificar se não existe algum elemento na consulta interna. Percebam que a consulta interna vai percorrer todas as tuplas de Y e verificar se existe algum elemento cujo valor de c é igual a x.a+1, neste caso, como x.a é igual a 1, x.a+1 ficaria igual a 2. Vejam que não existe nenhum tupla cujo valor de y.c é igual a 2. Logo, o resultado da consulta interna seria vazio e o valor atual do ponteiro C1 seria retornado na resposta.

Esse passo-a-passo seria feito para cada um dos valores de X, no final obteríamos 1, 3, 5 como valores. Essa é nossa resposta, presente na alternativa C.

Gabarito: C.

---

53. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação

X	
a	b
1	2
3	3
4	5
5	7

Y	
c	d
1	2
3	4
5	6
7	8
9	1

```
SELECT *
FROM X LEFT JOIN Y ON X.a = Y.c
ORDER BY X.a
```

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que esse comando produz:

A) 1, 2, 1, 2

3, 3, 3, 4

5, 7, 5, 6

B) 1, 2, 1, 2

3, 4, 3, 3

NULL, NULL, 4, 5

5, 6, 5, 7

C) 1, 2, 1, 2

3, 3, 3, 4

NULL, NULL, NULL, NULL

5, 7, 5, 6

D) 1, 2, 1, 2

3, 3, 3, 4

4, 5, NULL, NULL

5, 7, 5, 6

E) 1, 2

3, 3

4, 5

5, 7

Comentário: Vejam que a questão pede para executarmos um LEFT JOIN utilizando como referências os atributos x.a e y.c. Sabemos que o LEFT JOIN considera toda as tuplas da esquerda completando com NULL todas as vezes que o atributo de junção não tiver correspondência na tabela da direta. Desta forma, conseguimos observar que a tupla que não tem correspondência seria a terceira tupla de X onde o valor de a é igual a 4, valor que não existe na coluna c de Y. Neste caso, completaremos a tupla do resultado com valor nulo. Para os demais valores de X.a temos um valore correspondente de Y.

Despois da execução da junção é solicitado que o resultado seja ordenado por x.a. Como a tabela X já está ordenada pela coluna a, podemos observar o resultado da consulta na alternativa D.

Gabarito: D.

---

54. Ano: 2015 Banca: FGV Órgão: TCM-SP Cargo: Tecnologia da Informação

Views criadas nos bancos podem, de acordo com alguns critérios, ser naturalmente atualizáveis, o que significa, por exemplo, que podem ser objeto de comandos update do SQL sem a necessidade de mecanismos auxiliares ou triggers. Essa característica depende da expressão SQL que define a view e das tabelas/views de origem.

Considere alguns tipos de construções SQL que podem ser empregadas na definição de uma coluna de uma view:

I. funções de agregação, tais como sum, avg

II. funções escalares, tais como sin, trim

III. expressões aritméticas

IV. expressões condicionais, tais como case

V. literais

VI. subconsultas

Está correto concluir que uma determinada coluna **NÃO** pode ser objeto de atualização quando resultar de qualquer dos tipos:

A) apresentados, exceto I, II e III;

B) apresentados, exceto III e IV;

C) apresentados, exceto V;

D) apresentados, exceto VI;

E) apresentados.

Comentário: Sabemos que uma visão é uma tabela cujas linhas não são armazenadas explicitamente no banco de dados, mas são calculadas conforme necessário, com base em uma definição de visão. A motivação por trás do mecanismo de visões é personalizar a maneira como os usuários veem os dados.

O padrão SQL-92 permite que atualizações sejam especificadas apenas em visões definidas em uma única tabela base, usando apenas seleção e projeção, sem nenhum uso de operação de agregação. Tais visões são denominadas de **visões atualizáveis**. O padrão SQL-1999 ampliou a possibilidade de atualização, intuitivamente, podemos atualizar um campo de uma visão, se ele é obtido de exatamente uma das tabelas bases e a chave primária dessas tabelas estiver incluída na visão.

Também temos restrições de que o operador DISTINCT não pode ser usado em definições de visões atualizáveis.

Para finalizar, o Silberschatz diz que uma alteração por meio de visão somente é permitida se a visão em questão é definida em termos de uma **relação real do banco de dados.** Observem que nenhuma das alternativas apresenta valores armazenados no banco de dados, desta forma a letra E traz a nossa resposta.

Gabarito: E.

---

55. Ano: 2015 Banca: FGV Órgão: TCM-SP Cargo: Tecnologia da Informação

Considere a tabela relacional criada pelo comando

create table xx

(a int null, b int null, c int null)

Depois de instanciada com um conjunto de registros, os seguintes comandos foram executados:

select count (*) from XX

select count (distinct A) from XX

select count (distinct B) from XX

select count (*) from XX where C>10

select count (*) from XX where not C>10

Sabendo-se que esses comandos produziram como resultado, respectivamente, os números 10, 10, 0, 0 e 5, analise as quatro alternativas para a definição da tabela XX:

I. CREATE TABLE XX(

A int NULL,

B int NULL,

C int NULL)

II.CREATE TABLE XX(

A int primary key,

B int NULL,

C int NULL)

III.CREATE TABLE XX (

A int NULL,

B int NULL,

C int)

IV. CREATE TABLE XX (

A int,

B int primary key,

C int NULL)

A lista com todos os comandos que são válidos e compatíveis com a instância corrente da tabela é:

A) I, II;

B) I, II, III;

C) II, IV;

D) I, III;

E) IV.

Comentário: As conclusões que podemos tirar a partir do resultado das consultas no ajudam a responder à questão. A primeira consulta demonstra que a tabela tem 10 registros. A segunda consulta mostra que a coluna A possui 10 valores distintos e diferentes de NULL. A terceira apresenta a coluna B com todos os seus valores iguais a NULL, essa conclusão pode ser entendia pelo fato da cláusula DISTINCT desconsiderar valores nulos na sua contabilidade. Sendo assim, se temos o valor zero como resposta da consulta é porque todos os valores da coluna B são iguais a NULO.

As outras duas consultas devem ser usadas em conjunto. Se C > 10 e C<=0 (é a mesma coisa de dizer que not c>0) conseguem em conjunto contar apenas 5 tuplas é porque os demais valores de C são iguais a NULL.

Partido dessas considerações podemos construir as tabelas das alternativas I, II e III e inserir os mesmos valores sem problemas. Já a alternativa IV apresenta um erro ao definir B como chave primária, o que não é possível dado os valores nulos para o campo.

Gabarito: B

---

56. Ano: 2014 Banca: FGV Órgão: SUSAM Cargo: Técnico de Nível Superior A - Analista de Sistemas

A figura a seguir apresenta o diagrama das tabelas TBNotaFiscal e TBItemNotaFiscal. As tabelas 1 e 2 apresentam, respectivamente, os registros TBNotaFiscal e TBItemNotaFiscal.

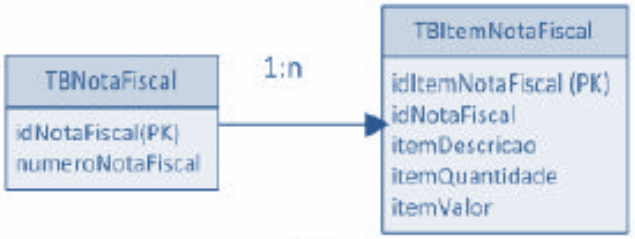


Tabela 1

TBNotaFiscal	
idNotaFiscal	numeroNotaFiscal
1	NF 0001
2	NF 0002
3	NF 0003

Tabela 2

TBItemNotaFiscal				
idItemNotaFiscal	idNotaFiscal	itemDescricao	itemQuantidade	itemValor
1	1	porca	100	0,10
2	1	parafuso	50	0,20
3	2	areia	10	10,50
4	2	cimento	15	20,00
5	2	telha	100	2,50

Os campos idCliente e idCompras são chaves primárias das tabelas TBNotaFiscal e TBItemNotaFiscal, respectivamente.

Assinale a opção que indica resposta correta para o seguinte comando SQL

```
SELECT a.numeroNotaFiscal, b.itemDescricao,
b.itemValor, b.itemQuant, b.itemValorAS
ValorUnitario,(b.itemValor)*(b.itemQuant)AS
ValorTotal
FROM TBNotaFiscal a, TBItemNotaFiscal b
WHERE a.idNotaFiscal = b.idNotaFiscal
```

A)
NF 0001	porca	0,1	100	0,1	10
NF 0001	parafuso	0,2	50	0,2	10
NF 0002	areia	10,5	10	10,5	105
NF 0002	cimento	20	15	20	300
NF 0002	telha	2,5	100	2,5	250

B)
NF 0001	porca	0,1	100	0,1	10
NF 0001	parafuso	0,2	50	0,2	10
NF 0002	areia	10,5	10	10,5	105
NF 0002	cimento	20	15	20	300
NF 0002	telha	2,5	100	2,5	250
NF 0003	NULL	NULL	NULL	NULL	NULL

C)
NF 0001	porca	0,1	100	0,1	10
NF 0001	parafuso	0,2	50	0,2	10
NF 0002	areia	10,5	10	10,5	105
NF 0002	cimento	20	15	20	300
NF 0002	telha	2,5	100	2,5	250
NF 0003	NULL	0	0	0	0

D)
NF 0001	NULL	NULL	NULL	NULL	20
NF 0002	NULL	NULL	NULL	NULL	655
NF 0003	NULL	NULL	NULL	NULL	0

E)
NF 0001	NULL	NULL	NULL	NULL	20

NF 0002 NULL NULL NULL NULL 655

Comentário: Essas alternativas ficaram um pouco difícil de visualizar, mas percebam que a nota ficas é composta de NF mais um número identificador. Agora vamos voltar nossa atenção para o comando descrito no enunciado, o SELECT lista alguns atributos presentes nas duas tabelas, em seguida faz um produto cartesiano seguido por uma seleção. Veja que estamos fazendo uma junção onde o atributo de junção é o idNotaFiscal. Este fato nos leva a percepção de que não haverá valores nulo na relação resultante da consulta. A ideia é concatenar o número da nota fiscal aos itens e suas descrições. Sendo assim, podemos observar nossa resposta na alternativa A.

Gabarito: A.

---

57. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação

Considere que as instâncias das tabelas T1, T2 e T3 têm, respectivamente, 1.000, 10.000 e 100.000 registros. O comando SQL

select 1 from t1

union

select 2 from t2

union

select 3 from t3

produz um resultado com:

A) 3 linhas;

B) 1.000 linhas;

C) 10.000 linhas;

D) 100.000 linhas;

E) 111.000 linhas.

Comentário: Vaja que comentamos essa característica do uso de valores numéricos na cláusula SELECT. Para cada um dos select teremos uma coluna com a quantidade de linhas iguais a quantidade de linha de cada tabela e os valores para cada linha iguais ao número, 1, 2 e 3, respectivamente. Agora, o pulo do gato está na operação UNION que vai remover os valores duplicados. Sendo assim nosso resultado será:

(1)

(2)

(3)

Ou seja, apenas 3 linhas serão retornadas.

Gabarito: A.

---

58. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação

Analise os comandos SQL a seguir, que produzem os resultados R1, R2 e R3, respectivamente.

I. select distinct x.* from x, y where x.a <> y.a

II. select distinct x.* from x

where x.a not in (select a from y)

III.select distinct x.* from x

where not exists

(select * from y where y.a=x.a)

Sabendo-se que nenhuma das instâncias das tabelas "x" e "y" é vazia, é correto concluir que:

A) R1 e R2 são iguais entre si e diferentes de R3;

B) R1 e R3 são iguais entre si e diferentes de R2;

C) R2 e R3 são iguais entre si e diferentes de R1;

D) R1, R2 e R3 são todos iguais entre si;

E) R1, R2 e R3 são todos diferentes entre si.

Comentário: A diferença é que na opção 1 haverá um produto cartesiano entre x e y que será então filtrado na cláusula WHERE. Nas outras opções, não haverá esse produto cartesiano. Apenas comparações entre os valores existentes nas relações x e y. Vejam uma solução para a questão executada no MySQL:

```
mysql> select * from x;
+------+
| a  |
+------+
|  1 |
|  2 |
|  3 |
+------+
3 rows in set (0.00 sec)

mysql> select * from y;
+------+
| a  |
+------+
|  2 |
```

```
| 3 |
| 4 |
+------+
3 rows in set (0.00 sec)
```

R1

```
mysql> select distinct x.* from x, y where x.a != y.a ; // <> é o mesmo que !=
+------+
| a  |
+------+
| 1 |
| 3 |
| 2 |
+------+
3 rows in set (0.00 sec)
```

R2

```
mysql> select distinct x.* from x
 -> where x.a not in (select a from y) ;
+------+
| a  |
+------+
| 1 |
+------+
1 row in set (0.00 sec)
```

R3

```
mysql>  select distinct x.* from x
 -> where not exists
 -> (select * from y where y.a=x.a) ;
+------+
| a  |
+------+
| 1 |
+------+
```

1 row in set (0.00 sec)

Analisando os comandos acima podemos perceber que R2 e R3 produzem o mesmo resultado. Já R1 possui um resultado distinto. A alternativa C contém tal informação, sendo, portanto, nossa resposta.

Gabarito: C.

---

59. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação

Analise as instâncias das tabelas R1 e R2 e o comando SQL, mostrados a seguir.

R1

x
1
3
4
6
7
8

R2

a	b
2	0
5	0
7	0
9	0
10	0

```
update R2
set b=(select count(x) from R1 where x > a)
```

Após a execução do comando, o conteúdo da coluna “b" da tabela R2 passa a ser, de cima para baixo:

A) 5, 5, 5, 5, 5;

B) NULL, NULL, NULL, NULL, NULL;

C) 5, 3, 1, 0, 0;

D) 0, 0, 0, 0, 0;

E) 5, 3, 1, NULL, NULL.

Comentário: Para resolver a questão precisamos perceber que o valor de b será atualizado com a quantidade de números presentes na coluna x que sejam maiores que a. Por exemplo, quantos valores da coluna x são maiores que 2? São 5 valores, no caso, (3, 4, 6, 7, 8). Fazendo esse exercício para as demais linhas da tabela R2 temos os seguintes valores da coluna b (5, 3, 1, 0, 0). Tais valores estão presentes na alternativa C.

Gabarito: C.

---

60. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas

Analise o comando de criação de uma tabela relacional mostrado a seguir.

```
create table inscrição
(matricula int not null,
  disciplina int not null,
  nota float null,
  constraint pk_ inscrição primary key (matricula,
disciplina),
  constraint fk_inscrição_aluno
        foreign key (matricula) references aluno
(matricula)
)
```

Sabendo-se que a coluna matricula constitui a chave primária da tabela aluno, está correto concluir que

A) aluno e inscrição têm um relacionamento 1:1 entre si.

B) aluno e inscrição têm um relacionamento 1:n entre si.

C) aluno e inscrição têm um relacionamento n:1 entre si.

D) aluno e inscrição têm um relacionamento m:n entre si.

E) inscrição é uma especialização de aluno.

Comentário: Essa questão nos relembra do assunto que vimos anteriormente quando tratamos de modelagem conceitual. Vejam que cada aluno pode ser matriculado em várias disciplinas, cada disciplina por meio de uma inscrição, mas percebam que cada inscrição só pode ter um aluno. Sendo assim, o relacionamento entre aluno e inscrição é 1:n. Temos, portanto, nossa resposta na alternativa B.

Gabarito: B

---

61. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas

Analise o comando SQL a seguir.

```
select x,
       sum (nota) as soma,
       count (nota) as numero
from inscricao
group by x
having ???
```

Assinale a opção que indica a expressão que, ao ser utilizada para substituir o trecho "???", **INVALIDA** o comando SQL acima.

A) count (nota) > 1

B) x=2

C) max (nota) = 10

D) nota > 7

E) (select max(nota) from inscricao) > 5

Comentário: Pela lógica a restrição sobre o atributo x deveria fazer parte da cláusula WHERE. Do ponto de vista de performance é melhor você restringir as linhas para depois fazer os agrupamentos e em seguida as restrições sobre o agrupamento. Contudo não é bem assim, como pode ser visto link.

O HAVING pode ser usado em funções agregadas ou AGRUPADAS. Assim, como o X que está no GROUP BY, também pode ser usado na cláusula HAVING. No caso da questão, o HAVING pode fazer restrições usando as funções de agregação e sobre os atributos que estão sendo AGRUPADOS pelo GROUP BY.

Observem que o atributo nota não aparece no group by, portanto, não pode ser usado isoladamente.

Gabarito: D.

---

62. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas

Na maioria das implementações SQL, pode-se considerar que as expressões lógicas possam assumir três valores, verdadeiro (T), falso (F) e desconhecido (?). Isso decorre principalmente da manipulação de valores nulos (NULL).

Assim sendo, analise as quatro expressões lógicas a seguir.

not ?

F or ?

T and ?

? or T

Assinale a opção que apresenta os valores finais das expressões lógicas acima, na ordem de cima para baixo.

A) F; ?; T; T

B) F; F; T; T

C) ?; ?; ?; ?

D) ?; ?; ?; T

E) ?; F; ?; ?

Comentário: A última questão trata da lógica de três valores usada em SQL. Vejam que **not ?** é igual a ?. **F or ?** é igual ?. **T and ?** é igual a ?. E ? or T é igual a T. Neste último caso basta substituir a ? por T e por F e verá que teremos os mesmos resultados. Sendo assim, podemos concluir que ? or T é igual a T.

Analisando todas as alternativas podemos encontrar nossa reposta na letra D.

Gabarito: D

QUESTÕES COMENTADAS VUNESP

QUESTÕES COMENTADAS VUNESP

1. (VUNESP - Ana SJ (TJM SP)/TJM SP/2023)

Considere a seguinte consulta SQL feita em um banco de dados relacional:

```
SELECT Operadora, Plano
 FROM Conta
 WHERE Plano LIKE 'Special%'
 ORDER BY Operadora
```

Um possível resultado obtido a partir dessa consulta é

a)

Operadora	Plano
Abc	Special Plus
Camel	Special X
Lotus	Special 1

b)

Operadora	Plano
Lotus	Special 1

Camel	Special X	
Abc	Top Plus	Special

c)

Operadora	Plano	
Lotus	Special 1	
Camel	Special X	
Abc	Top Plus	Special

d)

Operadora	Plano	
Lotus	Plano 1	Special
Camel	Plano X	Special
Abc	Top Plus	Special

e)

Operadora	Plano	

Abc	1 Special	
Camel	X Special	
Lotus	Top Special	Plus

Gabarito: A

Comentário: A opção correta, considerando a consulta SQL fornecida, é: a). A consulta seleciona as colunas Operadora e Plano da tabela Conta onde o plano começa com 'Special%', ordenado pela coluna Operadora.

2. (VUNESP - ATI (TJ RS)/TJ RS/Análise de Sistemas/2023)

Considere a seguinte tabela de um banco de dados relacional:

Aluno (ID, Nome, Curso)

O comando SQL para obter o nome e o curso dos alunos, com a condição de que o curso se inicie com a palavra Física é:

a) SELECT Nome, Curso
FROM Aluno
WHERE Curso == Física%;

b) SELECT Nome, Curso
FROM Aluno
WHERE Curso.Física%;

c) SELECT Nome, Curso
FROM Aluno
WHERE Curso LIKE Física%;

d) SELECT Nome, Curso

FROM Aluno

WHERE Curso(Física%);

e) SELECT Nome, Curso

FROM Aluno

WHERE Curso = Física%;

Gabarito: C

Comentário:

O comando SQL correto para obter o nome e o curso dos alunos, com a condição de que o curso se inicie com a palavra "Física" é:

SELECT Nome, Curso

FROM Aluno

WHERE Curso LIKE 'Física%';

Portanto, a alternativa correta é a letra c).

3. (VUNESP - Ana (Pref Marília)/Pref Marília/Dados/2023)

O comando SQL para excluir completamente uma tabela (dados e estrutura) denominada Paper de um banco de dados relacional é

a) UNDO Paper.

b) EXC Paper.

c) DROP Paper.

d) DELETE Paper.

e) TRASH Paper.

Gabarito: C

Comentário:

O comando SQL para excluir completamente uma tabela (dados e estrutura) de um banco de dados relacional é: c) DROP Paper. Portanto, a alternativa correta é a letra c).

4. (VUNESP - Ana (Pref Marília)/Pref Marília/Dados/2023)

Considere a tabela de um banco de dados relacional

TX (A, B, C, D)

e a seguinte consulta sobre essa tabela

```
SELECT C, D
FROM TX
WHERE B> 30 AND C < 50
```

Como resultado da execução dessa consulta, serão exibidos apenas os valores dos atributos

a) C e D.

b) B e C.

c) A e C.

d) B, C e D.

e) A, C e D.

Gabarito: A

Comentário: Na consulta SQL fornecida:

SELECT C, D

FROM TX

WHERE B > 30 AND C < 50;

A cláusula WHERE está filtrando as linhas onde o valor de B é maior que 30 e o valor de C é menor que 50. A projeção (SELECT) está escolhendo apenas as colunas C e D. Portanto, como resultado da execução dessa consulta, serão exibidos apenas os valores dos atributos: a) C e D.

Portanto, a alternativa correta é a letra a).

5. (VUNESP - PTIC (UNICAMP)/UNICAMP/Programador de Sistemas de Informação/2023)

Deseja-se criar uma tabela em um banco de dados relacional, contendo dois atributos (A e B), ambos do tipo string com 20 caracteres cada um. Ambos devem compor a chave primária da tabela, a ser denominada Tab.

O comando SQL para realizar a tarefa aqui descrita é:

a) CREATE TABLE Tab
(A Char (20),
B Char (20),
Primary Key (A, B);

b) CREATE TABLE Tab
(A, B Char (20),
Primary Key (A, B);

c) CREATE TABLE Tab
(A AND B Char (20),
Primary Key (A, B);

d) CREATE TABLE Tab
(A Char(20) NOT NULL,
B Char (20) NOT NULL);

e) CREATE TABLE Tab
(A, B Char (20), PK);

Gabarito: A

Comentário: A opção correta para criar uma tabela em um banco de dados relacional, com dois atributos (A e B) que compõem a chave primária, é:

```
CREATE TABLE Tab

(

   A CHAR(20),
```

B CHAR(20),

PRIMARY KEY (A, B)

);

Portanto, a alternativa correta é a letra a).

6. (VUNESP - PTIC (UNICAMP)/UNICAMP/Programador de Sistemas de Informação/2023)

Considere uma tabela denominada Tablet, de um banco de dados relacional. É necessário acrescentar um novo atributo (Fonte) a essa tabela, do tipo inteiro (supor que haja o tipo Integer no gerenciador a ser utilizado). O comando SQL para realizar tal tarefa é:

a) ALTER TABLE Tablet

PLUS Fonte (Integer);

b) ALTER TABLE Tablet

ADD Fonte (Integer);

c) ALTER TABLE

ADD Tablet.Fonte (Integer);

d) ALTER TABLE Tablet >> Fonte (Integer);

e) ALTER TABLE Tablet ++ Fonte (Integer);

Gabarito: B

Comentário: O comando SQL para adicionar um novo atributo chamado Fonte à tabela Tablet, com o tipo de dados Integer, é:

ALTER TABLE Tablet

ADD Fonte Integer;

Portanto, a alternativa correta é a letra b).

7. (VUNESP - PTIC (UNICAMP)/UNICAMP/Analista de Desenvolvimento de Sistemas/2023)

Considere a seguinte tabela de um banco de dados relacional.

Sala (Número, Capacidade, Nome)

O comando SQL para obter o número, a capacidade e o nome das salas, considerando aquelas com capacidade maior do que 4 é:

a) SELECT Sala. *
WHERE Capacidade > 4;

b) SELECT Sala (Número, Capacidade, Nome)
WHERE Capacidade > 4;

c) SELECT Sala (*)
WHERE Capacidade > 4;

d) SELECT Número ... Nome
FROM Sala
WHERE Capacidade > 4;

e) SELECT *
FROM Sala
WHERE Capacidade > 4;

Gabarito: E

Comentário: O comando SQL correto para obter o número, a capacidade e o nome das salas, considerando aquelas com capacidade maior do que 4, é:

SELECT *

FROM Sala

WHERE Capacidade > 4;

A alternativa correta, portanto, é a letra e).

8. (VUNESP - PTIC (UNICAMP)/UNICAMP/Analista de Desenvolvimento de Sistemas/2023)

Considere a seguinte tabela de um banco de dados relacional.

Veículo (Placa, Modelo, Marca, Tipo, Ano)

O comando SQL para obter parte dos atributos (placa, modelo e marca) de veículos fabricados no ano 2020 e do tipo SUV ou Misto é:

a) SELECT *
FROM Veículo
WHERE Ano = 2020 AND Tipo INTO ('SUV', 'Misto');

b) SELECT Placa, Modelo, Marca
FROM Veículo
WHERE Ano = 2020 AND Tipo ='SUV' AND
Tipo = 'Misto';

c) SELECT Placa ... Marca
FROM Veículo
WHERE Ano = 2020 AND Tipo LIKE ('SUV', 'Misto');

d) SELECT Placa, Modelo, Marca
FROM Veículo
WHERE Ano = 2020 AND Tipo IN ('SUV', 'Misto');

e) SELECT *
FROM Veículo
WHERE Ano = 2020 AND (Tipo ='SUV',
Tipo = 'Misto');

Gabarito: D

Comentário: O comando SQL correto para obter parte dos atributos (placa, modelo e marca) de veículos fabricados no ano 2020 e do tipo SUV ou Misto é usando a cláusula IN:

SELECT Placa, Modelo, Marca

FROM Veículo

WHERE Ano = 2020 AND Tipo IN ('SUV', 'Misto');

Portanto, a alternativa correta é:

d) SELECT Placa, Modelo, Marca FROM Veículo WHERE Ano = 2020 AND Tipo IN ('SUV', 'Misto');

9. (VUNESP - PTIC (UNICAMP)/UNICAMP/Analista de Desenvolvimento de Sistemas/2023)

Considere a seguinte tabela de um banco de dados relacional.

Consultório (Sala, Bloco, Tipo, Proprietário)

O comando SQL para criar um índice baseado no atributo Proprietário, índice esse de nome First, é:

a) CREATE INDEX First

ON Consutório (Proprietário);

b) CREATE INDEX Consultório (Proprietário)

AT Consultório.First;

c) CREATE INDEX Consultório.First

FOR Proprietário;

d) CREATE INDEX First

IN Proprietário OF Consultório;

e) CREATE INDEX ON First.

Consultório.Proprietário;

Gabarito: A

Comentário:

O comando SQL correto para criar um índice baseado no atributo Proprietário, com o índice nomeado como First, seria:

CREATE INDEX First

ON Consultório (Proprietário);

Portanto, a alternativa correta é:

a) CREATE INDEX First ON Consultório (Proprietário);

Lembrando que, em SQL, um índice é uma estrutura que melhora a velocidade de recuperação de dados de uma tabela. Ele é criado em uma ou mais colunas da tabela, permitindo que o sistema de gerenciamento de banco de dados (SGBD) encontre registros mais eficientemente quando a busca é feita nas colunas indexadas. Os índices são particularmente úteis em tabelas grandes.

O comando básico para criar um índice em SQL é a declaração CREATE INDEX. Aqui estão alguns detalhes sobre a sintaxe e o uso do comando:

CREATE INDEX nome_do_indice

ON nome_da_tabela (coluna1, coluna2, ...);

nome_do_indice: Especifica o nome do índice que você está criando.

nome_da_tabela: É o nome da tabela na qual o índice está sendo criado.

coluna1, coluna2, ...: São as colunas nas quais o índice será baseado. Pode ser uma ou mais colunas.

Por exemplo, considerando a tabela Consultório e a criação de um índice baseado na coluna Proprietário:

CREATE INDEX First

ON Consultório (Proprietário);

Este índice facilitará a busca e a recuperação de dados da tabela Consultório quando a cláusula WHERE envolver a coluna Proprietário.

Lembre-se de que, embora os índices possam melhorar o desempenho das consultas, eles também têm impacto nas operações de atualização (inserção, atualização, exclusão), pois o índice deve ser mantido atualizado. Portanto, é importante considerar o equilíbrio entre melhorar as consultas e o impacto nas operações de atualização ao decidir onde criar índices.

10. (VUNESP - PTIC (UNICAMP)/UNICAMP/Analista de Desenvolvimento de Sistemas/2023)

Considere as seguintes tabelas de um banco de dados relacional.

Computador (ID, Modelo, Fabricante)
Usuário (Código, Nome, Função)
Usa (Código, ID), sendo Código e ID chaves estrangeiras
com origem nas tabelas Usuário e Computador, respectivamente.

O comando SQL para obter nome do usuário e modelo de computador utilizado é:

a) SELECT Nome, Modelo
FROM Computador, Usuário;

b) SELECT Nome, Modelo
FROM Computador C, Usuário U, Usa S
WHERE C.ID = S.ID AND U.Código = S.Código;

c) SELECT Nome, Modelo
FROM Computador C, Usuário U, Usa S
WHERE ID (C, S) AND Código (U, S);

d) SELECT Nome, Modelo
FROM Computador C, Usuário U, Usa S
WHERE C AND S (ID); S AND U (Código);

e) SELECT Nome, Modelo
FROM Computador C, Usuário U, Usa S
WHERE C (*) = S (*) AND U (*) = S (*);

Gabarito: B

Comentário: O comando SQL correto para obter o nome do usuário e o modelo do computador utilizado, com base nas tabelas fornecidas e a tabela de relação Usa, é:

SELECT U.Nome, C.Modelo

FROM Computador C

JOIN Usa S ON C.ID = S.ID

JOIN Usuário U ON U.Código = S.Código;

Aqui estão os elementos chave deste comando:

JOIN: Utilizado para combinar registros de diferentes tabelas com base em uma condição.

ON: Especifica a condição de junção.

U.Nome, C.Modelo: São as colunas selecionadas na consulta.

Portanto, a alternativa correta é a letra:

b) SELECT Nome, Modelo FROM Computador C, Usuário U, Usa S WHERE C.ID = S.ID AND U.Código = S.Código;

11. (VUNESP - WDes (Pref Pinda)/Pref Pindamonhangaba/2023)

Considere as seguintes tabelas de um banco de dados relacional:

Disciplina (Código, Nome-D, ID)

Curso (ID, Nome-C, Descrição)

sendo que, na tabela Disciplina, ID é uma chave estrangeira, com origem na tabela Curso.

O comando SQL para obter os nomes das disciplinas e o nome do curso a que a disciplina pertence é

a) SELECT Nome-D, Nome-C

FROM Disciplina D, Curso C;

b) SELECT Nome-D, Nome-C

FROM Disciplina D.ID, Curso C.ID;

c) SELECT Nome-D, Nome-C

WHERE Disciplina D.ID IN Curso C.ID;

d) SELECT Nome-D, Nome-C

FROM Disciplina D, Curso C

AND Curso.ID = Disciplina.ID;

e) SELECT Nome-D, Nome-C

FROM Disciplina D, Curso C

WHERE Curso.ID = Disciplina.ID;

Gabarito: E

Comentário: O comando SQL correto para obter os nomes das disciplinas e o nome do curso a que a disciplina pertence, com base nas tabelas fornecidas, é usando a cláusula JOIN:

SELECT D.Nome-D, C.Nome-C

FROM Disciplina D

JOIN Curso C ON D.ID = C.ID;

Aqui estão os elementos chave deste comando:

JOIN: Utilizado para combinar registros de diferentes tabelas com base em uma condição.

ON: Especifica a condição de junção.

D.Nome-D, C.Nome-C: São as colunas selecionadas na consulta.

Portanto, a alternativa correta é a letra:

e) SELECT Nome-D, Nome-C FROM Disciplina D, Curso C WHERE Curso.ID = Disciplina.ID;

12. (VUNESP - WDes (Pref Pinda)/Pref Pindamonhangaba/2023)

Considere a seguinte tabela de um banco de dados relacional:

Cliente (CPF, Nome, Peso)

O comando para excluir os clientes de peso superior a 80 kg é (supondo o atributo peso expresso em kg) é:

a) DELETE Peso.Cliente > 80;

b) DELETE Cliente.Peso > 80;

c) DELETE FROM Cliente

WHERE Peso > 80;

d) DELETE Peso > 80

FROM Cliente;

e) DELETE FROM Cliente

WHERE Peso INF 80;

Gabarito: C

Comentário:

O comando SQL correto para excluir os clientes de peso superior a 80 kg é:

DELETE FROM Cliente

WHERE Peso > 80;

Portanto, a alternativa correta é a letra: c) DELETE FROM Cliente WHERE Peso > 80;

13. (VUNESP - ATI (TJ RS)/TJ RS/Análise de Suporte/2023)

Considere a seguinte tabela de um banco de dados relacional:

Loja (ID, Nome, Endereço, Cidade, Estado)

O comando SQL para obter o nome e número de lojas que não estão no Estado de São Paulo é:

a) SELECT Nome, Sum (Estado)
FROM Loja
WHERE Estado != 'São Paulo;

b) SELECT Nome, Count (Estado)
FROM Loja
WHERE NOT (Estado.'São Paulo')

c) SELECT Nome, Count (Estado)
FROM Loja
WHERE Estado <> 'São Paulo';

d) SELECT Nome, Add (Estado)
FROM Loja
WHERE Estado.~'São Paulo';

e) SELECT Nome, Join (Estado)
FROM Loja

WHERE Estado NOT IN (‘São Paulo’);

Gabarito: C

Comentário: O comando SQL correto para obter o nome e o número de lojas que não estão no Estado de São Paulo é:

SELECT Nome, COUNT(*)

FROM Loja

WHERE Estado <> 'São Paulo';

Aqui estão os elementos chave deste comando:

SELECT Nome, COUNT(*): Seleciona o nome da loja e conta o número de ocorrências.

FROM Loja: Especifica a tabela da qual os dados são selecionados.

WHERE Estado <> 'São Paulo': Estabelece a condição para incluir apenas as lojas que não estão no Estado de São Paulo.

Portanto, a alternativa correta é:

c) SELECT Nome, Count (Estado) FROM Loja WHERE Estado <> 'São Paulo';

14. (VUNESP - TTI (TJ RS)/TJ RS/Programador/2023)

Um Técnico de Tecnologia da Informação do TJ-RS se deparou com uma situação na qual, sempre que o sistema adicionasse ou excluísse um registro de processo, seria preciso atualizar automática e simultaneamente uma tabela com a função de contador.

Para a resolução desse problema, considerando um banco de dados relacional e SQL, o procedimento ou funcionalidade mais indicado é:

a) Procedure.

b) View.

c) Trigger.

d) Job.

e) Sequence.

Gabarito: C

Comentário: A funcionalidade mais indicada para lidar com a atualização automática e simultânea de uma tabela, sempre que um registro de processo é adicionado ou excluído, é a Trigger. Portanto, a resposta correta é a opção c) Trigger.

Abaixo estão comentadas as demais alternativas:

a) Procedure: Procedimentos armazenados são conjuntos de instruções SQL que podem ser chamados para execução. Eles são úteis para realizar operações específicas, mas não são automaticamente acionados por eventos como adição ou exclusão de registros.

b) View: Views são consultas predefinidas que podem ser referenciadas como tabelas virtuais. Elas não são acionadas automaticamente por eventos como adição ou exclusão de registros, sendo mais voltadas para facilitar consultas complexas ou fornecer uma visão específica dos dados.

d) Job: Jobs, normalmente associados a sistemas de agendamento de tarefas, são usados para executar tarefas em horários agendados ou em intervalos específicos, mas geralmente não são acionados automaticamente por eventos de manipulação de dados.

e) Sequence: Sequências são objetos usados para gerar valores únicos em uma coluna. Elas não estão diretamente relacionadas à atualização automática de tabelas em resposta a eventos de adição ou exclusão de registros.

15. (VUNESP - ADP (DPE SP)/DPE SP/Administrador de Redes/2023)

Considere a seguinte tabela de um banco de dados relacional:

Armário (ID, Tipo, Item, Peso)

O comando SQL para obter o número total de registros contidos na tabela Armário é:

a) COUNT (ID) FROM Armário;

b) NUM (ID) FROM Armário;

c) SELECT COUNT(ID) FROM Armário;

d) COUNT ALL FROM Armário;

e) SELECT COUNT ALL FROM Armário;

Gabarito: C

Comentário: A opção correta é: c) SELECT COUNT(ID) FROM Armário; O comando COUNT(ID) conta o número total de registros na coluna ID da tabela Armário. O uso correto do COUNT na consulta SQL é seguido pelo nome da coluna ou * (asterisco) para contar todos os registros da tabela. Portanto, a consulta SELECT COUNT(ID) FROM Armário; retorna o número total de registros na tabela Armário com base na coluna ID.

16. (VUNESP - ADP (DPE SP)/DPE SP/Analista Desenvolvedor/2023)

Considere a seguinte tabela de um banco de dados relacional.

Estoque (ID, Tipo, Item, Qtdade)

O comando SQL para obter o Tipo e a Quantidade do estoque, agrupado por Tipo, contido na tabela é:

a) SELECT Tipo, COUNT (Qtdade)

FROM Estoque;

b) SELECT Tipo, SUM (Qtdade)

FROM Estoque;

c) SELECT Tipo, Qtdade

FROM Estoque

GROUP BY COUNT(Tipo);

d) SELECT Tipo, Qtdade

FROM Estoque

GROUP BY SUM(Tipo);

e) SELECT Tipo, SUM (Qtdade)

FROM Estoque

GROUP BY (Tipo);

Gabarito: E

Comentário: O gabarito da questão é letra e) SELECT Tipo, SUM (Qtdade) FROM Estoque GROUP BY (Tipo); Esta opção utiliza a função de agregação SUM para obter a soma da quantidade (Qtdade) agrupada pelo tipo (Tipo). A diferença entre as opções B e E está na presença dos parênteses em torno de Tipo na cláusula GROUP BY.

17. (VUNESP - CFO/QC (EsFCEx)/EsFCEx/Informática/2023)

Considere a seguinte tabela de um banco de dados relacional:

Time (ID, Nome, Cidade, Estado)

O comando SQL para obter o número de registros de cada Estado presente na tabela, além do nome do Estado, é:

a) SELECT Count (Estado), Estado FROM Time GROUP BY Estado;

b) SELECT Sum (Estado), Estado FROM Time ORDER BY Estado;

c) SELECT Count (Estado), Estado FROM Time HAVING Estado IN Group;

d) SELECT Max (Estado) – Min (Estado) FROM Time GROUP BY Estado;

e) SELECT Number (Estado), Estado FROM Time WHERE Estado IS NOT NULL;

Gabarito: A

Comentário: A opção correta para obter o número de registros de cada Estado presente na tabela, além do nome do Estado, é: a) SELECT COUNT(Estado), Estado FROM Time GROUP BY Estado;

Esta consulta utiliza a função de agregação COUNT para contar o número de registros para cada valor único da coluna Estado na tabela Time. A cláusula GROUP BY Estado é usada para agrupar os resultados por Estado. Portanto, esta opção retorna a contagem de registros para cada Estado presente na tabela.

18. (VUNESP - CFO/QC (EsFCEx)/EsFCEx/Informática/2023)

Considere a seguinte tabela de um banco de dados relacional:

Material (ID, Nome, Descrição, Fabricante)

O comando SQL para obter o nome e o fabricante, com a condição de excluir os fabricantes Blue e Green é:

a) SELECT Nome, Fabricante FROM Material WHERE Fabricante NOT ('Blue','Green');

b) SELECT Nome, Fabricante FROM Material WHERE Fabricante DISTINCT ('Blue','Green');

c) SELECT Nome, Fabricante FROM Material WHERE Fabricante <> 'Blue' OR 'Green'

d) SELECT Nome, Fabricante FROM Material WHERE NOT Fabricante ('Blue','Green');

e) SELECT Nome, Fabricante FROM Material WHERE Fabricante NOT IN ('Blue', 'Green');

Gabarito: E

Comentário: A opção correta para obter o nome e o fabricante, excluindo os fabricantes Blue e Green, é: e) SELECT Nome, Fabricante FROM Material WHERE Fabricante NOT IN ('Blue', 'Green');

A cláusula NOT IN é usada para excluir as linhas em que o valor da coluna Fabricante é 'Blue' ou 'Green'. Portanto, essa opção atende à condição especificada na pergunta.

19. VUNESP - Tec CPDJ (TJM SP)/TJM SP/2023

Considere a seguinte tabela de um banco de dados relacional:

Passagem (ID, Origem, Destino, Valor)

O comando SQL para obter a Origem e Destino para valores situados entre 200,00 e 1.000,00, ordenado pela Origem, é:

a) SELECT Origem, Destino

FROM Passagem

HAVING Valor BETWEEN (200.00; 1000.00);

b) SELECT Origem, Destino

FROM Passagem

ORDER BY Origem AND

(Valor >= 200.00 OR Valor <= 1000.00);

c) SELECT Origem, Destino

FROM Passagem

WHERE Origem ASC AND

Valor BETWEEN (200.00 <> 1000.00);

d) SELECT Origem, Destino

FROM Passagem

WHERE Valor BETWEEN (200.00 AND 1000.00)

ORDER BY Origem;

e) SELECT Origem, Destino

FROM Passagem

HAVING Origem ASC AND

Valor BETWEEN (200.00 UNTIL 1000.00);

Gabarito: D

Comentário: A opção correta para obter a Origem e Destino para valores situados entre 200,00 e 1.000,00, ordenado pela Origem, é: d) SELECT Origem, Destino FROM Passagem WHERE Valor BETWEEN 200.00 AND 1000.00 ORDER BY Origem;

A cláusula WHERE é usada para filtrar as linhas onde o valor da coluna Valor está entre 200,00 e 1.000,00. A cláusula ORDER BY é usada para ordenar os resultados com base na coluna Origem.

20. (VUNESP - Ana (Pref Marília)/Pref Marília/Dados/2023)

A sintaxe básica do comando de criação de gatilhos em SQL é:

a) CREATE FUNCTION ...

b) CREATE TRIGGER ...

c) SELECT TRIGGER ...

d) DROP TRIGGER ...

e) INSERT TRIGGER ...

Gabarito: B

Comentário: A sintaxe básica do comando de criação de gatilhos (triggers) em SQL é:

```
CREATE TRIGGER nome_gatilho
[BEFORE | AFTER | INSTEAD OF] {DELETE | INSERT | UPDATE [OF coluna[, coluna...]]}
ON nome_tabela
[REFERENCING OLD AS nome_antigo NEW AS nome_novo]
[FOR EACH ROW]
WHEN (condição)
DECLARE
  -- declaração de variáveis locais
BEGIN
  -- corpo do gatilho
END;
```

Portanto, a opção correta é a letra b) CREATE TRIGGER …

21. (VUNESP - ATI (UFABC)/UFABC/2023)

Considere a seguinte tabela de um banco de dados relacional:

Paciente (ID, Nome, Idade, Convenio, Valor)

O comando SQL para majorar em 12% o valor para os convênios denominados Único e Final é:

a) OUTPUT Valor = Valor * 1.12
FROM Convenio = 'Único' AND Convenio =
'Final' FOR Paciente;

b) SELECT Valor = Valor * 1.12
HAVING Convenio = 'Único' AND Convenio =
'Final' IN Paciente;

c) WRITE Convenio.Valor = Convenio.Valor * 1.12
FROM Paciente.Convenio IN ('Único' OR 'Final');

d) UPDATE Paciente.Valor = Paciente.Valor * 1.12
FROM Convenio = 'Único' AND Convenio = 'Final';

e) UPDATE Paciente
SET Valor = Valor * 1.12
WHERE Convenio IN ('Único','Final');

Gabarito: E

Comentário: O comando SQL correto para majorar em 12% o valor para os convênios denominados Único e Final é:

UPDATE Paciente

SET Valor = Valor * 1.12

WHERE Convenio IN ('Único', 'Final');

Portanto, a opção correta é: e) UPDATE Paciente

SET Valor = Valor * 1.12

WHERE Convenio IN ('Único', 'Final');

22. (VUNESP - Téc Lab (UFABC)/UFABC/Informática/2023)

Considere a seguinte tabela de um banco de dados relacional:

Veiculo (RENAVAM, Marca, Modelo, Ano)

O comando SQL para excluir os veículos de modelo Sedan e ano superior a 2018 é:

a) DELETE FROM SELECT Veiculo WHERE Modelo = 'Sedan' AND Ano > 2018;

b) DELETE FROM Veículo WHERE Modelo = ‘Sedan’ AND Ano > 2018;

c) DELETE SELECT Veiculo FOR Modelo = ‘Sedan’ AND Ano > 2018;

d) DELETE Veiculo.Modelo = ‘Sedan’ AND Veiculo.Ano > 2018;

e) SELECT DEL FROM Veiculo HAVING (Modelo, Ano) (= ‘Sedan’, > 2018);

Gabarito: B

Comentário: O comando SQL correto para excluir os veículos de modelo Sedan e ano superior a 2018 é: b) DELETE FROM Veiculo WHERE Modelo = 'Sedan' AND Ano > 2018; Portanto, a opção correta é a letra 'b.

O comando SQL DELETE FROM é utilizado para remover registros de uma tabela. No caso específico da tabela "Veiculo", a condição WHERE é utilizada para especificar quais registros devem ser excluídos. Na opção 'b', a condição estabelece que apenas os veículos com modelo "Sedan" e ano superior a 2018 serão excluídos. Isso significa que todas as linhas da tabela "Veiculo" que atendem a essa condição serão removidas, mantendo apenas os veículos que não se enquadram nesses critérios.

Lista de Questões

1. (MPU-TO - DBA - CESPE – 2024)

Julgue os itens subsequentes, com relação à linguagem de consulta estruturada (SQL).

O comando HAVING é usado para unir duas ou mais tabelas.

2. (MPU-TO - DBA - CESPE – 2024)

Julgue os itens subsequentes, com relação à linguagem de consulta estruturada (SQL).

O comando TRUNCATE é usado para remover todas as linhas de uma tabela.

3. (MPU-TO - DBA - CESPE – 2024)

A respeito de arquitetura de bancos de dados, julgue os itens a seguir.

O esquema do banco de dados é alterado a cada comando de INSERT, UPDATE ou DELETE em uma de suas tabelas.

4. (MPU-TO - DBA - CESPE – 2024)

Quanto a linguagens de consulta, julgue os próximos itens.

Em uma consulta SQL, tuplas duplicadas são automaticamente eliminadas pelo SGBD, com o objetivo de aumentar o desempenho.

5. (MPU-TO - DBA - CESPE – 2024)

Quanto a linguagens de consulta, julgue os próximos itens.

Em uma tabela com 100 registros, uma consulta SQL com a cláusula WHERE pode apresentar resultados com quantidade variando de zero a 100 linhas.

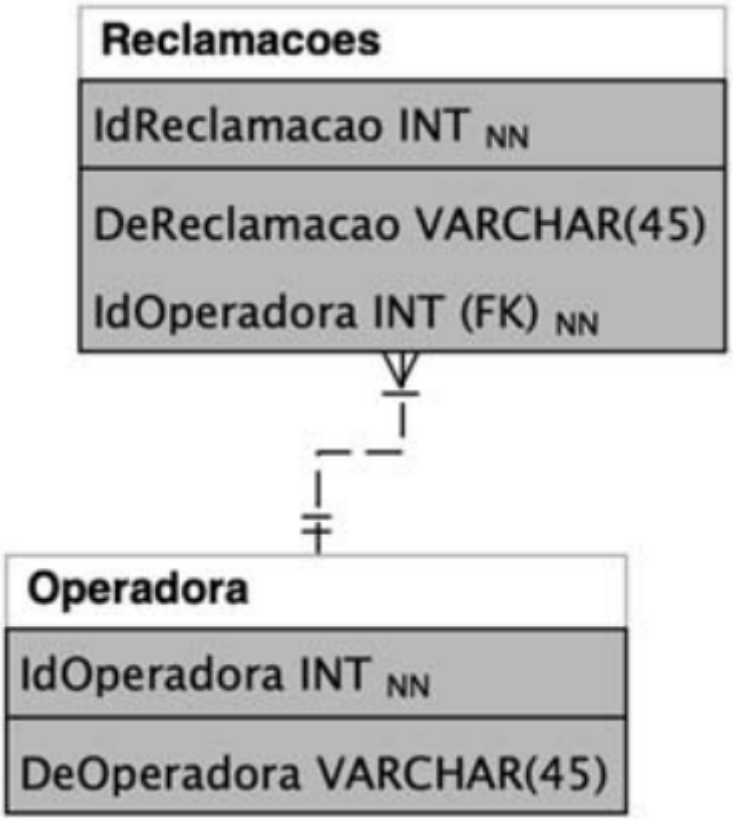
6. (MPU-TO - DBA - CESPE – 2024)

Quanto a linguagens de consulta, julgue os próximos itens.

A SQL permite a consulta simultânea de atributos com o mesmo nome, desde que a ordem desses atributos seja a mesma da ordem das tabelas onde eles estão armazenados.

7. (ANAC - Analista de Sistemas - CESPE – 2024)

Figura CB1A5



No que se refere a banco de dados, julgue os itens a seguir, tendo o modelo apresentado na figura CB1A5 como referência.

```
CREATE TABLE Operadora (
  IdOperadora INT NOT NULL,
  DeOperadora VARCHAR(45) NULL,
  PRIMARY KEY (IdOperadora));

CREATE TABLE Reclamacoes (
  IdReclamacao INT NOT NULL AUTO_INCREMENT,
  DeReclamacao VARCHAR(45) NULL,
  IdOperadora INT NOT NULL,
  PRIMARY KEY (IdReclamacao),
    FOREIGN KEY (IdOperadora)
    REFERENCES Operadora (IdOperadora));
```

37

A execução dos comandos SQL precedentes resulta na criação de duas tabelas e no relacionamento descrito no modelo apresentado na figura CB1A5.

8. (ANAC - Analista de Sistemas - CESPE – 2024)

IdOperadora	DeOperadora
1	Operadora A
2	Operadora B

IdReclamacao	DeReclamacao	IdOperadora
10	CDC	1
20	CDC	1
30	CDC	1
50	ABC	2
60	ABC	2

38

Considerando que as tabelas do modelo apresentado na figura CB1A5 possuam os registros precedentes, é correto afirmar que a execução do seguinte comando

```
SELECT o.Deoperadora, COUNT(r.idreclamacao)
AS Quantidade
FROM Reclamacoes r
INNER JOIN Operadora o ON o.idoperadora =
r.IdOperadora
GROUP BY o.Deoperadora
```

terá como resultado a tabela a seguir.

DeOperadora	Quantidade
Operadora A	3
Operadora B	2

9. TST/ Cesbraspe/2024/TI

Assinale a opção correspondente ao comando utilizado para adicionar, remover ou modificar colunas de tabelas de bancos de dados SQL.

a) delete
b) update
c) insert
d) create
e) alter

10. TST/ Cesbraspe/2024/TI

Assinale a opção que indica o comando que só pode ser definido para executar em nível de instrução no banco de dados PostgreSQL.

a) after
b) truncate
c) update
d) instead of
e) before

11. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

Com pertinência à linguagem SQL, julgue o item abaixo.

Considere-se o seguinte *script* SQL.

```
select report_code, year, month, day,
wind_speed,
case
  when wind_speed>= 40 then 'HIGH'
  when wind_speed>= 30 then 'MODERATE'
else 'LOW'
end as wind_severity
from station_data
```

O resultado da execução do script resultará em erro, pois, caso haja, na tabela station_data, algum registro no campo wind_speed com valor superior a 40, não será possível predizer se o valor da variável wind_severity será igual a 'HIGH'.

12. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

A respeito de banco de dados, julgue o próximo item.

Em um comando SELECT, a cláusula WHERE define que o resultado da consulta é o produto cartesiano das tabelas envolvidas.

13. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

A respeito de banco de dados, julgue o próximo item.

Em SQL, o comando DISTINCT é utilizado para eliminar resultados repetidos em consultas a tabelas do banco de dados.

14. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

Julgue o item seguinte a respeito dos conceitos de administração de dados.

Os comandos TRUNCATE e DROP TABLE removem todas as linhas de uma tabela, porém o comando DROP TABLE exclui também a estrutura da tabela do banco de dados bem como todos os dados armazenados na tabela.

15. CEBRASPE (CESPE) - APO (SEPLAN RR)/SEPLAN RR/Tecnologia da Informação/2023

Considerando os conceitos de tuning de banco de dados, julgue o item a seguir.

O comando EXPLAIN permite otimizar tabelas que executam muitas operações de UPDATE e DELETE em detrimento de operações de INSERT.

16. CEBRASPE (CESPE) - Per Crim (POLC AL)/POLC AL/Análise de Sistemas, Ciências da Computação, Informática. Processamento de Dados ou Sistemas da Informação/2023

Com relação aos componentes de um computador, aos barramentos de E/S, à aritmética computacional e à linguagem SQL, julgue o próximo item.

Em SQL, para que não haja erro de construção (sintaxe), as cláusulas GROUP BY e HAVING, quando usadas, devem ser definidas sempre antes da cláusula WHERE.

17. CEBRASPE (CESPE) - Per Crim (POLC AL)/POLC AL/Análise de Sistemas, Ciências da Computação, Informática. Processamento de Dados ou Sistemas da Informação/2023

Com relação aos componentes de um computador, aos barramentos de E/S, à aritmética computacional e à linguagem SQL, julgue o próximo item.

Ao ser executado, o comando SQL a seguir mostrará o CPF e o nome de todas as pessoas que cometeram pelo menos um crime do tipo hediondo.

```
select P.CPF, P.NOME
from POLITEC.PESSOA P
where P;CPF exists (select C.CPF
                    from POLITEC.CRIME C
                    where C.CPF=P.CPF
                    and C.TIPO = 'HEDIONDO');
```

18. CEBRASPE (CESPE) - Ana Reg (AGER MT)/AGER MT/Ciências da Computação e Sistemas de Informação/2023

Assinale a opção em que é apresentada a instrução utilizada para alterar alguma propriedade de campo de uma tabela em um banco de dados.

a) INSERT

b) UPDATE

c) ALTER

d) DROP

e) TRUNCATE

19. CEBRASPE (CESPE) - Ana Reg (AGER MT)/AGER MT/Ciências da Computação e Sistemas de Informação/2023

Em linguagem de manipulação de dados DML, o operador SQL BETWEEN serve para

a) delimitar o valor de uma coluna na cláusula WHERE.

b) delimitar as colunas a serem apresentadas na cláusula WHERE.

c) delimitar os limites de um campo para a cláusula INSERT.

d) restringir a quantidade de linhas a serem recuperadas na cláusula UPDATE.

e) restringir a quantidade de campos na cláusula DELETE.

20. CEBRASPE (CESPE) - Ana (SERPRO)/SERPRO/Tecnologia/2023

Considerando que existem diferentes tipos de banco de dados, como os bancos de dados relacionais e os não relacionais (ou NoSQL), julgue o item a seguir.

Considere-se que, em um banco de dados, constem duas tabelas — clientes e pedidos — e que na primeira tabela haja informações dos clientes, como ID, nome e endereço, e na segunda tabela, informações dos pedidos realizados pelos clientes, como ID do pedido, data e valor total. Nessa situação hipotética, para retornar ao nome e ao endereço de clientes que já realizaram pedidos, é correto usar o comando SELECT em conjunto com a cláusula LEFT JOIN.

21. CEBRASPE (CESPE) - AIS (EMPREL)/EMPREL/Banco de Dados/2023

Na linguagem SQL, a função que é usada para verificar se o resultado de uma subconsulta correlacionada é vazio ou não, a qual retorna como resultado o valor booleano TRUE ou FALSE, é

a) HAVING.

b) EXISTS.

c) LIKE.

d) IN.

e) AVG.

22. CEBRASPE (CESPE) - AFM (Pref Fortaleza)/Pref Fortaleza/Ciência da Computação, Informática, Processamento de Dados/2023

Julgue o item a seguir, a respeito de arquitetura de dados, metadados e linguagens de bancos de dados.

Usando-se SQL (*structured query language*), é possível unir o resultado de duas instruções SELECT quaisquer, por meio do operador UNION.

23. CEBRASPE (CESPE) - AFM (Pref Fortaleza)/Pref Fortaleza/Ciência da Computação, Informática, Processamento de Dados/2023

Julgue o item a seguir, a respeito de arquitetura de dados, metadados e linguagens de bancos de dados.

Comandos expressos em linguagem de definição de dados (DDL) são utilizados para criar estruturas de um banco de dados, e o seu processamento irá incluir ou alterar metadados desse mesmo banco de dados.

24. CEBRASPE (CESPE) - AFM (Pref Fortaleza)/Pref Fortaleza/Ciência da Computação, Informática, Processamento de Dados/2023

Julgue o item a seguir, a respeito de arquitetura de dados, metadados e linguagens de bancos de dados.

A manipulação de dados armazenados em um banco de dados é feita por meio de DML (*Data Manipulation Language*), linguagem na qual é possível apenas descrever os dados que se deseja acessar, sem especificar como obtê-los.

25. CEBRASPE (CESPE) - AFM (Pref Fortaleza)/Pref Fortaleza/Geografia/2023

No que diz respeito a banco de dados relacional e banco de dados geográfico, julgue o item a seguir.

A SQL (*structured query language*) pode ser subdividida em duas sublinguagens: a linguagem de definição de dados (DDL), que fornece comandos visando definir e modificar esquemas de tabelas, remover tabelas, criar índices e definir restrições de integridade; e a linguagem de manipulação de dados (DML), formada por comandos para consulta, inserção, modificação e remoção de dados no banco de dados.

26. CEBRASPE (CESPE) - Ana TI (FUB)/FUB/2023

Acerca de bancos de dados relacionais, julgue o item seguinte.

Triggers e *stored procedures* são componentes do modelo físico, enquanto relacionamentos e *primary keys* são componentes do modelo lógico.

27. CEBRASPE (CESPE) - Ana (MPE RO)/MPE RO/Programador/2023

Assinale a opção que apresenta a instrução em SQL que permite substituir por Pedro o nome do advogado atual da parte autora (nome_adv_autor) do processo de número 2023001, em uma tabela de nome processo.

a) UPDATE processo

SET nome_adv_autor = 'Pedro'

WHERE numero_processo = 2023001;

b) CHANGE processo

SET nome_adv_autor = 'Pedro'

WHERE numero_processo = 2023001;

c) ALTER TABLE processo

SET nome_adv_autor = 'Pedro'

WHERE numero_processo = 2023001;

d) UPDATE processo

SET nome_adv_autor := 'Pedro'

WHERE numero_processo = 2023001;

e) UPDATE processo

SET nome_adv_autor LIKE 'Pedro'

WHERE numero_processo = 2023001;

28. Cebraspe – Técnico (Analista de Sistemas) Infraestrutura e Segurança da Informação (BNB)/2022

O cursor armazena, em memória, uma única linha do comando de SELECT de uma consulta que tenha sido efetuada em tabelas do banco de dados.

29. Cebraspe – Técnico (Analista de Sistemas) Infraestrutura e Segurança da Informação (BNB)/2022

Quando acionado, um trigger pode ser executado em substituição ao comando que o disparou.

30. Cebraspe – Técnico (Analista de Sistemas) Infraestrutura e Segurança da Informação (BNB)/2022

A linguagem de controle de dados (DCL) é um dos componentes da SQL e permite gerenciar as autorizações aos dados constantes do banco de dados.

31. Cebraspe – Técnico – Analista de Sistemas – Desenvolvimento de Sistemas (BNB)/2022

Julgue os itens que se seguem, acerca dos conceitos de linguagem de consulta estruturada (SQL).

Considere-se a tabela e o script SQL a seguir.

Tabela: colaboradores

id	sexo	idade
1	Masculino	32
2	Feminino	0
3	Feminino	30

```
SELECT avg(idade), sexo
FROM colaboradores
GROUP BY sexo
```

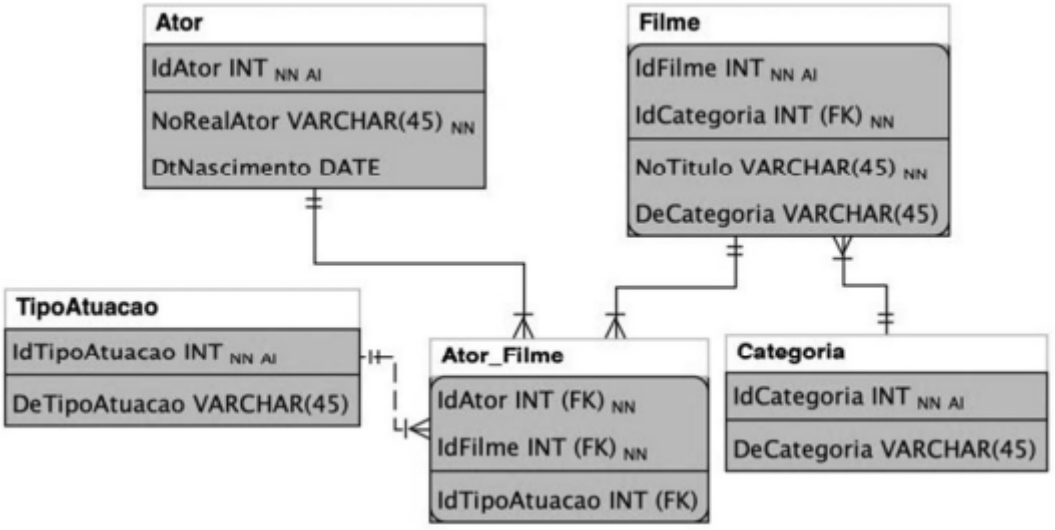
O resultado da consulta SQL é

```
avg(idade)      sexo
15              Feminino
32              Masculino
```

32. Cebraspe – Técnico – Analista de Sistemas – Desenvolvimento de Sistemas (BNB)/2022

A linguagem de manipulação de dados (DML) inclui instruções que modificam a estrutura de um banco de dados.

33. Cebraspe – Analista Judiciário – Tecnologia da Informação (TRT-AP/PA)/2022



Considere o modelo precedente, em que os campos IdFilme, IdAtor, IdTipoAtuacao e IdCategoria são chaves primárias em suas respectivas tabelas, como referência.

Considere ainda que

FK descreve que o campo é uma foreign key;

AI descreve que o campo é auto incremente;

NN descreve que o campo é not null.

A partir dessas informações, assinale a opção que apresenta o correto script SQL para criação desta tabela.

a)
```
CREATE TABLE Categoria (
  IdCategoria INT NOT NULL AUTO_INCREMENT,
  DeCategoria VARCHAR(45) NULL,
  PRIMARY KEY (IdCategoria),
    FOREIGN KEY (IdCategoria)
    REFERENCES Filme (IdCategoria)
) ENGINE = InnoDB;
```

b)
```
CREATE TABLE Filme (
  IdFilme INT NOT NULL AUTO_INCREMENT,
  NoTitulo VARCHAR(45) NOT NULL,
  DeCategoria VARCHAR(45) NULL,
  IdCategoria INT NOT NULL,
  PRIMARY KEY (IdFilme, IdCategoria)
) ENGINE = InnoDB;
```

c)
```
CREATE TABLE Ator (
  IdAtor    INT    NOT    NULL    PRIMARY    KEY
AUTO_INCREMENT,
  NoRealAtor VARCHAR(45) NOT NULL,
  DtNascimento DATE NULL,
    FOREIGN KEY (IdAtor)
    REFERENCES Ator_Filme (IdAtor)
) ENGINE = InnoDB;
```

d)
```
CREATE TABLE Ator_Filme (
  IdAtor INT NOT NULL,
  IdFilme INT NOT NULL,
  IdTipoAtuacao INT NULL,
  PRIMARY KEY (IdAtor, IdFilme),
    FOREIGN KEY (IdAtor)
    REFERENCES Ator (IdAtor),
    FOREIGN KEY (IdFilme)
    REFERENCES Filme (IdFilme),
    FOREIGN KEY (IdTipoAtuacao)
    REFERENCES TipoAtuacao (IdTipoAtuacao)
) ENGINE = InnoDB;
```

e)

```
CREATE TABLE TipoAtuacao (
  IdTipoAtuacao  INT  NOT  NULL  AUTO_INCREMENT
PRIMARY KEY,
  DeTipoAtuacao VARCHAR(45) NULL,
    FOREIGN KEY (IdTipoAtuacao)
    REFERENCES Ator_Filme (IdTipoAtuacao)
) ENGINE = InnoDB;
```

34. Cebraspe – Quality Assurance (QA) e Analista de Teste (BANRISUL)/2022

Na linguagem SQL, os comandos GRANT e REVOKE são exemplos do subconjunto DDL (linguagem de definição de dados), pois definem o acesso aos dados do banco.

35. Cebraspe – Quality Assurance (QA) e Analista de Teste (BANRISUL)/2022

No que se refere à álgebra relacional e a SQL, julgue os itens a seguir.

Considerando-se uma tabela nomeada empregados que contém, entre outras colunas, uma identificada como nome, o comando SQL que retorna o nome de todos os empregados que tenham o nome luiz ou luis é o seguinte. SELECT nome FROM empregados WHERE nome like '%lui_%'

36. Cebraspe – Analista de Contas Públicas – Especialidade: Administração (MPC-SC)/2022

Acerca do conceito de view, da modelagem dimensional, do modelo de referência CRISP-DM e da linguagem SQL, julgue os itens subsequentes.

Em banco de dados relacionais, as views são conhecidas como tabelas físicas, uma vez que elas armazenam apenas as informações de interesse do usuário, que podem ser coletadas de uma ou mais tabelas do banco.

37. Cebraspe – Analista de Contas Públicas – Especialidade: Administração (MPC-SC)/2022

O comando a seguir tem a finalidade de mostrar o nome e o email de todos os procuradores que recebem salários acima de R$ 30.000,00.

```
SELECT nome, email
FROM procurador
WHEN salario >= 30.000,00;
```

38. Cebraspe – Analista de Contas Públicas – Especialidade: Administração (MPC-SC)/2022

O CREATE é o principal comando da linguagem SQL, pois permite criar diversos objetos no banco de dados, como, por exemplo, tabelas e domínios. Entretanto, esse comando não é utilizado para criar views.

39. Ano: 2018 Banca: CESPE Órgão: STJ Cargo: Técnico Judiciário – Suporte Técnico

A respeito de sistemas gerenciadores de banco de dados (SGBD), julgue os próximos itens.

79 O comando GRANT é utilizado para conceder privilégios em um objeto do SGBD, ao passo que o comando REVOKE serve para cancelar um privilégio já concedido.

81 A linguagem de manipulação de dados (DML – data definition language) é usada para, entre outras finalidades, criar e alterar estruturas de tabelas em um SGBD.

82 Em um SGBD, o trigger pode substituir a instrução que o originou, sendo a instrução original descartada e apenas o trigger executado.

40. Ano: 2018 Banca: CESPE Órgão: STJ Cargo: Técnico Judiciário – Suporte Técnico

Acerca de banco de dados, julgue os itens que se seguem.

76 A diferença entre materialized view e view comum em um banco de dados é o fato de que a primeira é armazenada em cache como uma tabela física, enquanto a segunda existe apenas virtualmente.

41. Ano: 2018 Banca: CESPE Órgão: STJ Cargo: Técnico Judiciário – Desenvolvimento de Sistemas Questão:

Julgue os seguintes itens, relativos a métricas de qualidade desoftware, JUnit, SQL, Delphi e desenvolvimento mobile.

107 A sentença SQL seguinte produzirá como resultado a lista de todos os funcionários de uma empresa. Para aqueles em que seja verdadeira a condição

Funcionarios.CodigoDep = Departamentos.CodigoDep, será apresentado também o nome do departamento.

SELECT Funcionarios.Nome, Departamentos.NomeDep FROM Funcionarios

INNER JOIN Departamentos ON Funcionarios.CodigoDep = Departamentos.CodigoDep

ORDER BY Funcionarios.Nome;

42. Ano: 2018 Banca: CESPE Órgão: ABIN Cargo: Área 08 Questão: 144

A respeito de sistemas gerenciadores de banco de dados, julgue os próximos itens.

144 O comando SQL select campo from tabela corresponde a uma operação de projeção da álgebra relacional.

43. Ano: 2018 Banca: CESPE Órgão: ABIN Cargo: Área 09 Questões: 144 a 147

SELECT nome FROM funcionario

WHERE area = 'INTELIGENCIA'

AND endereco LIKE '%BRASILIA,DF%';

Tendo como referência o código SQL precedente, julgue os itens a seguir.

144 Na cláusula WHERE, a condição de seleção area = 'INTELIGENCIA' escolhe a tupla de interesse em particular na tabela funcionario, pois area é um atributo de funcionario.

145 O código em apreço realiza uma consulta que mostra o nome dos funcionários da área de INTELIGENCIA e que têm, como parte do endereço, a cidade de BRASILIA,DF.

146 A palavra INTELIGENCIA está entre aspas simples por pertencer a um atributo, area, o qual tem o tipo de dados definido como caractere.

147 147 Em LIKE '%BRASILIA,DF%', o recurso LIKE foi definido de forma incorreta, uma vez que a utilização da vírgula (,), sem a inclusão da palavra-chave ESCAPE, impedirá que o código seja executado.

44. BANCA: CESPE ANO: 2015 ÓRGÃO: TJDFT CARGO: PROGRAMAÇÃO DE SISTEMAS – QUESTÕES 63 E 64

Acerca de linguagens de definição e manipulação de dados, julgue os itens subsecutivos.

Apelido ou column alias não pode ser utilizado na cláusula WHERE.

Em uma coluna definida como NUMBER (7,2), o valor 34567.2255 será armazenado como 34567.23.

45. BANCA: CESPE ANO: 2015 ÓRGÃO: TJDFT CARGO: PROGRAMAÇÃO DE SISTEMAS – QUESTÕES 65 A 66

Julgue os próximos itens, relativos a SQL.

65 O comando SQL ilustrado a seguir atualiza os dados dos empregados do departamento (id_departamento) 50 que têm como função (id_funcao) VENDEDOR para o departamento 80 e gerente (id_gerente) 145.

UPDATE empregados

SET id_departamento = 80, id_gerente = 145

WHERE id_departamento = 50 AND funcao = 'VENDEDOR';

66 O comando SQL mostrado a seguir fará uma consulta na tabela empregados e retornará

os campos primeiro_nome, sobrenome e salario de todos os empregados do departamento (id_departamento) 40, ordenados pelo campo sobrenome.

SELECT primeiro_nome, sobrenome, salario FROM empregados

WHERE id_departamento = 40 ORDER BY sobrenom

46. BANCA: CESPE ANO: 2016 ÓRGÃO: TCE-SC CARGO: AUDITOR DE TI

Com relação aos bancos de dados relacionais, julgue os próximos itens.

O catálogo de um sistema de gerenciamento de banco de dados relacional armazena a descrição da estrutura do banco de dados e contém informações a respeito de cada arquivo, do tipo e formato de armazenamento de cada item de dado e das restrições relativas aos dados.

Denomina-se visão uma tabela única derivada de uma ou mais tabelas básicas do banco. Essa tabela existe em forma física e viabiliza operações ilimitadas de atualização e consulta.

Em bancos de dados relacionais, as tabelas que compartilham um elemento de dado em comum podem ser combinadas para apresentar dados solicitados pelos usuários.

47. BANCA: CESPE ANO: 2016 ÓRGÃO: TRT-08 CARGO: ANALISTA DE TI - QUESTÃO 13

Acerca de SQL (structured query language), assinale a opção correta.

a) A otimização semântica de consultas utiliza restrições existentes no banco de dados (como atributos únicos, por exemplo) com o objetivo de transformar um SELECT em outro mais eficiente para ser executado.

b) Quando os registros de uma tabela estão ordenados fisicamente em um arquivo, segundo um campo que também é campo-chave, o índice primário passa a se chamar índice cluster.

c) Em uma mesma base de dados, uma instrução de SELECT com união de duas tabelas (INNER JOIN) e uma instrução de SELECT em uma tabela utilizando um SELECT interno de outra tabela (subquery) produzem o mesmo resultado e são executadas com o mesmo desempenho ou velocidade.

d) A normalização de dados é uma forma de otimizar consultas SQL, ao apresentar um modelo de dados com um mínimo de redundância. Isso é atingido quando o modelo estiver na quinta forma normal (5FN).

e) Para obter a quantidade de linhas que atendem a determinada instrução SQL, o processo mais eficiente e rápido é executar o comando SELECT e aplicar uma estrutura de loop para contar as linhas resultantes.

48. BANCA: CESPE ANO: 2016 ÓRGÃO: TRT-08 CARGO: ANALISTA DE TI - QUESTÃO 3

CREATE TABLE predio (

id numeric(7,0), nome varchar(50), local varchar(150),

mnemonico varchar(10),

CONSTRAINT pk_sede PRIMARY KEY (id),
CONSTRAINT uq_sede UNIQUE (mnemonico)
);
CREATE TABLE salas (
codigo numeric(7,0) NOT NULL, local varchar(10),
descricao varchar(50), area numeric(10,2),
CONSTRAINT pk_salas PRIMARY KEY (codigo), CONSTRAINT fk_sede_sala FOREIGN KEY (local)
REFERENCES predio (mnemonico)
);

Considerando os algoritmos acima, em que são criadas as tabelas predio e salas, assinale a opção cuja expressão SQL apresenta informações do registro da maior sala existente.

a) select c1.local, c1.nome, c2.descricao, c2.area from predio as c1, salas as c2
where c2.local=c1.mnemonico having max(c2.area)

b) select c1.local, c1.nome, c2.descricao, max(c2.area) from predio as c1, salas as c2

where c2.local=c1.id
group by c1.local, c1.nome, c2.descricao

c) select c1.local, c1.nome, c2.descricao from predio as c1, (
select local, descricao, area from salas as c1 where area = (select max(area) from salas as c2 where area>0)
) as c2 where c2.local=c1.mnemonico;

d) select c1.local, c1.nome, c2.descricao, c2.area from predio as c1, (
select local, descricao, area from salas as c1 where area = (select max(area) from salas as c2 where area>0)
) as c2 where c2.id=c1.codigo;

e) select c1.local, c1.nome, c2.descricao, max(c2.area) from predio as c1 join salas as c2
on c2.codigo=c1.id

group by c1.local, c1.nome, c2.descricao

49. BANCA: CESPE ANO: 2015 ÓRGÃO: FUB PROVA: ANALISTA ADMINISTRATIVO - ANALISTA DE TECNOLOGIA DA INFORMAÇÃO

Julgue os itens seguintes, no que se refere à linguagem SQL.

Supondo que seja necessário buscar dados em duas tabelas distintas, o comando select não deve ser escolhido por não possuir os recursos para efetuar a busca em ambas as tabelas e exibir o resultado.

A função max, utilizada conjuntamente com o comando select, retorna o maior valor em um determinado campo que tenha sido incluído na busca.

50. BANCA: CESPE ANO: 2008 ÓRGÃO: TJ-DF PROVA: ANALISTA JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO

Quanto a bancos de dados, sistemas gerenciadores de bancos de dados e técnicas correlacionadas de modelagem de dados, julgue os próximos itens.

Na linguagem de consulta SQL (structured query language), é possível obter o resultado de uma consulta SELECT ordenado pelo valor de um ou mais atributos.

51. Ano: 2016 Banca: CESPE Órgão: TCE-PR Cargo: Analista de Controle Área: Tecnologia da Informação Questão: 62

A respeito de SQL, assinale a opção correta.

a) A função STDDEV é utilizada para calcular a média aritmética de determinada coluna
b) O produto cartesiano é o resultado da combinação de mais de uma tabela, havendo pelo menos uma coluna em comum entre elas, de maneira que se apresentem os registros que constam simultaneamente em todas as tabelas
c) O resultado de uma *subquery* é utilizado como argumento para uma *query* superior e pode conter uma única linha, múltiplas linhas ou múltiplas linhas e colunas.
d) Uma instrução SQL de *insert* deve citar nominalmente todas as colunas da tabela.
e) As instruções *insert*, *update* e *delete* são processadas no banco de dados após serem executadas pelo usuário, dispensando-se o uso de outro comando para a disponibilização de seus resultados a outros usuários.

52. Ano: 2016 Banca: CESPE Órgão: TCE-PR Cargo: Analista de Controle Área: Tecnologia da Informação Questão: 63

No que se refere a banco de dados, assinale a opção correta

a) DDL (*data definition language*) e DML (*data manipulation language*) são linguagens utilizadas pelos usuários e desenvolvedores para manipular os dados em um banco de dados.

b) Os bancos de dados objeto-relacionais representam uma evolução dos bancos de dados relacionais, pois, incorporam várias funcionalidades anteriormente implementadas nos bancos de dados orientados a objetos.

c) A restrição de asserção de um banco de dados permite a execução de ações automáticas a partir de eventos previamente definidos, por exemplo, a entrada de um CPF com formatação incorreta.
d) Uma *view* representa uma tabela em forma física consolidada a partir de outras tabelas previamente definidas.

e) Em chaves primárias compostas, formadas por mais de um atributo, o valor NULL, é adotado para qualquer atributo exceto para o primeiro na ordem de formação da chave.

53. Ano: 2016 Banca: CESPE Órgão: TRT-08 Cargo: Técnico de TI - QUESTÃO 54

tabela t1

codigo	descricao
1	Britain
2	Rich CA
3	Columbia

tabela t2

codigo	valor
1	200
1	150
3	300
4	130

resultado

codigo	descricao	soma
1	Britain	350
3	Columbia	300
2	Rich CA	

Considerando as tabelas t1 e t2, assinale a opção que apresenta a expressão SQL que gera o conteúdo da tabela resultado.

a) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 join t2 on t1.codigo=t2.codigo group by t1.codigo, t1.descricao;

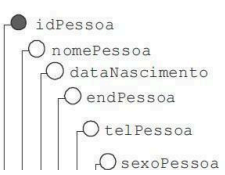
b) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 left join t2 on t1.codigo=t2.codigo group by t1.codigo, t1.descricao;

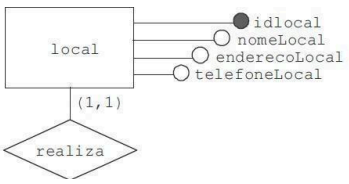
c) select t1.codigo, t1.descricao, sum(t2.valor) as soma
from t1 right join t2 on t1.codigo=t2.codigo group by t1.codigo, t1.descricao;

d) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 full join t2 on t1.codigo=t2.codigo group by t1.codigo, t1.descricao;

e) select t1.codigo, t1.descricao, sum(t2.valor) as soma from t1 , t2
where t1.codigo=t2.codigo group by t1.codigo, t1.descricao;

Texto para as próximas duas questões: a seguir, são apresentados um modelo entidade-relacionamento conceitual e, na tabela, características dos seus atributos.





pessoa	idPessoa	int(5)
	nomePessoa	varchar(45)
	dataNascimento	Date
	endPessoa	varchar(45)
	telPessoa	varchar(45)
	sexoPessoa	char(1)
local	idlocal	Int(3)
	nomeLocal	varchar(45)
	enderecoLocal	varchar(45)
	telefoneLocal	varchar(45)
encontro	idencontro	int(5)
	data_encontro	Date
preferencia	idpreferencia	int(3)
	descricaoPreferencia	varchar(45)

54. Ano: 2016 Banca: CESPE Órgão: TRE-PE - Cargo 2 TÉCNICO JUDICIÁRIO – ÁREA APOIO ESPECIALIZADO – ESPECIALIDADE PROGRAMAÇÃO DE SISTEMAS

QUESTÃO 29 - Considerando que os relacionamentos do modelo mostrado tenham sido mapeados, assinale a opção que apresenta comando em linguagem SQL capaz de criar fisicamente a tabela encontro, incluindo-se as chaves estrangeiras necessárias.

a) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL PRIMARY KEY,

'idlocal' INT(3) NULL, 'idpessoa_marca' INT(5) NULL, 'idpessoa_atende' INT(5) NULL,

'data_encontro' DATE NULL,

FOREIGN KEY ('idPessoa') REFERENCES 'pessoa' ('idPessoa'), FOREIGN KEY ('idlocal')

REFERENCES 'local' ('idlocal'));

b) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL PRIMARY KEY,

'idlocal' INT(3) NULL, 'idpessoa_marca' INT(5) NULL,

'idpessoa_atende' INT(5) NULL, 'data_encontro' DATE NULL,

FOREIGN KEY fk_pessoa_marca ('idPessoa') REFERENCES 'pessoa' ('idPessoa'), FOREIGN KEY fk_pessoa_atende ('idPessoa') REFERENCES 'pessoa' ('idPessoa'), FOREIGN KEY ('idlocal') REFERENCES 'local' ('idlocal'));

c) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL, 'idpessoa_marca' INT(5) NULL, 'idpessoa_atende' INT(5) NULL,

'data_encontro' DATE NULL, 'idlocal' INT(3) NULL,

PRIMARY KEY ('idencontro'),

CONSTRAINT 'fkpessoa_marca' FOREIGN KEY ('idpessoa_marca') REFERENCES 'pessoa' ('idPessoa'),
CONSTRAINT 'fkpessoa_atende' FOREIGN KEY ('idpessoa_atende') REFERENCES 'pessoa' ('idPessoa'),
CONSTRAINT 'fklocal_encontro' FOREIGN KEY ('idlocal') REFERENCES 'local' ('idlocal'));
d) CREATE TABLE encontro (

'idencontro' INT(5) NOT NULL, 'idpessoa_marca' INT(5) NULL, 'idpessoa_atende' INT(5) NULL, 'data_encontro' DATE NULL, 'idlocal' INT(3) NULL,
PRIMARY KEY CONSTRAINT ('idencontro'),
FOREIGN KEY CONSTRAINT 'fkpessoa_marca' ('idpessoa_marca') REFERENCES 'pessoa' ('idPessoa'),
FOREIGN KEY CONSTRAINT 'fkpessoa_atende' ('idpessoa_atende') REFERENCES 'pessoa' ('idPessoa'),
FOREIGN KEY CONSTRAINT 'fklocal_encontro' ('idlocal') REFERENCES 'local' ('idlocal'));
e) CREATE TABLE encontro (
'idencontro' INT(5) NOT NULL PRIMARY KEY,
'data_encontro' DATE NULL,
'idpessoa_marca' INT(5) NULL FOREIGN KEY ('idpessoa_marca') REFERENCES 'pessoa' ('idPessoa'),
'idpessoa_atende' INT(5) NULL FOREIGN KEY ('idpessoa_atende') REFERENCES 'pessoa' ('idPessoa'),
'idlocal' INT(3) NULL FOREIGN KEY ('idlocal') REFERENCES 'local' ('idlocal'));

55. Ano: 2016 Banca: CESPE Órgão: TRE-PE - Cargo 2 TÉCNICO JUDICIÁRIO – ÁREA APOIO ESPECIALIZADO – ESPECIALIDADE PROGRAMAÇÃO DE SISTEMAS

QUESTÃO 30 - Considere que, quando da realização do mapeamento do modelo conceitual para o modelo relacional-conceitual, tenham sido criadas na tabela encontro as chaves estrangeiras idpessoa_marca e idpessoa_atende, que identificam, respectivamente, quem marcou um encontro e quem atendeu a pessoa nesse mesmo encontro. A partir do modelo apresentado e dessas informações, assinale a opção que apresenta comando que permite apresentar uma listagem que mostre uma vez o nome de quem não marcou nenhum encontro, mas atendeu pelo menos a um.

a) select distinct nomePessoa

from encontro inner join pessoa on idpessoa_atende = idPessoa and idPessoa not in (select

idpessoa_marca from encontro);

b) select distinct nomePessoa

from pessoa where idPessoa not in (select idpessoa_marca from encontro);

c) select distinct nomePessoa

from encontro full join pessoa on idpessoa_atende = idPessoa;

d) select distinct nomePessoa

from encontro left join pessoa on idpessoa_atende = idPessoa;

e) select distinct nomePessoa

from encontro right join pessoa on idpessoa_atende = idPessoa and idPessoa not in (select idpessoa_marca from encontro);

56. Ano: 2016 Banca: CESPE Órgão: PCPE Cargo: PERITO CRIMINAL Questão: 38.

Na linguagem SQL, o comando *create table* é usado para criar uma tabela no banco de dados; enquanto o relacionamento entre duas tabelas pode ser criado pela declaração

a) null.

b) primary key.

c) constraint.

d) auto_increment.

e) not null.

57. Ano: 2016 Banca: CESPE Órgão: PCPE Cargo: PERITO CRIMINAL Questão: 39.

Na linguagem SQL, quando for necessário obter uma lista e criar uma condição, pode-se utilizar a cláusula

a) min.

b) sum.

c) where.

d) avg.

e) max.

58. Ano: 2016 Banca: CESPE Órgão: PCPE Cargo: PERITO CRIMINAL Questão: 40.

Em SQL, para alterar a estrutura de uma tabela do banco de dados e incluir nela uma nova *foreign key*, é correto utilizar o comando

a) convert.

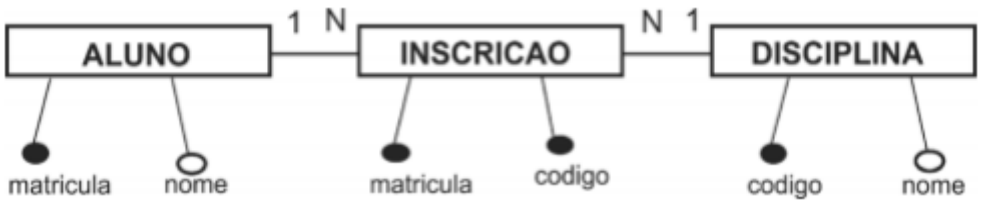
b) group by.

c) alter table.

d) update.

e) insert.

59. Ano: 2015 Banca: CESPE Órgão: TRE-PI – Questão 58



O modelo lógico apresentado dá origem às tabelas ALUNO, INSCRICAO e DISCIPLINA. Considerando esse modelo e sabendo que não há nenhum procedimento armazenado no banco de dados, assinale a opção que apresenta código em SQL ANSI que resultará corretamente na listagem de matricula e nome dos alunos que estão inscritos (INSCRICAO) em mais de duas disciplinas.

a) SELECT aluno.matricula, aluno.nome FROM inscricao, aluno, disciplina
WHERE inscricao.matricula=aluno.matricula AND inscricao.codigo=disciplina.codigo GROUP BY aluno.matricula, aluno.nome HAVING COUNT(*) > 2

b) SELECT aluno.matricula, aluno.nome FROM inscricao, aluno, disciplina
WHERE inscricao.matricula=aluno.matricula AND inscricao.codigo=disciplina.codigo AND COUNT(*) > 2

c) SELECT aluno.matricula, aluno.nome GROUP BY aluno.matricula, aluno.nome FROM inscricao, aluno, disciplina
WHERE inscricao.matricula=aluno.matricula AND inscricao.Codigo=diciplina. Codigo AND COUNT

d) SELECT aluno.matricula, aluno.nome FROM inscricao, aluno, disciplina
WHERE inscricao.matricula=aluno.matricula AND quantidade > 2
GROUP BY inscricao, aluno, disciplina

e) SELECT aluno.matricula, aluno.nome, SUM() > 2 FROM inscricao, aluno, disciplina
WHERE inscricao.matricula=aluno.matricula AND inscricao.codigo=disciplina.codigo
GROUP BY aluno.matricula, aluno.nome

60. Ano: 2015 Banca: CESPE Órgão: TRE-PI – Questão 59

Considerando um SGBD que respeite os padrões SQL ANSI-99, assinale a opção que apresenta corretamente um comando SQL para apagar determinados registros de uma tabela pessoa (cpf, nome, sexo) que contém registros cujo campo sexo apresenta valores iguais a 'M' e 'F'.

a) DROP pessoa WHERE sexo='M'

b) UPDATE pessoa SET sexo=NULL WHERE sexo='M'

c) FROM pessoa WHERE sexo='M' DELETE sexo

d) DELETE sexo FROM pessoa WHERE sexo='M'

e) DELETE FROM pessoa WHERE sexo='M'

GABARITO

01	02	03	04	05	06
Errado	Certo	Errado	Errado	Certo	Errado
07	08	09	10	11	12
Certo	Certo	E	B	Errado	Errado
13	14	15	16	17	18
Certo	Certo	Errado	Errado	Errado	C
19	20	21	22	23	24
A	Anulada	B	Errado	Certo	Errado
25	26	27	28	29	30
Certo	Certo	A	Errado	Certo	Certo
31	32	33	34	35	36
Certo	Errado	D	Errado	Certo	Errado
37	38	39	40	41	42
Errado	Errado	C; E; C	C	E	C
43	44	45	46	47	48
C; C; C; E	C; C	E; C	C; E; C	A	C
49	50	51	52	53	54
E; C	C	C	B	B	C
55	56	57	58	59	60
A	C	C	C	A	E

Lista de Questões

1. (UFMT - Analista de Sistemas - CESGRANRIO – 2024)

A respeito do uso de procedimentos armazenados e de gatilhos em um banco de dados relacional, verifica-se, em relação à sua aplicabilidade, que

(A) os gatilhos são ideais para encapsular lógica de apresentação em um banco de dados.

(B) os gatilhos são usados, principalmente, para encapsular lógica de negócios complexa e reutilizável.

(C) os procedimentos armazenados são preferíveis para impor restrições de integridade referencial.

(D) os procedimentos armazenados são adequados para automatizar a execução de ações em resposta a eventos específicos.

(E) ambos, procedimentos armazenados e gatilhos, são exclusivamente usados para consultas complexas em bancos de dados.

2. (UFMT - Analista de Sistemas - CESGRANRIO – 2024)

Considere a criação de um banco de dados relacional para a biblioteca de uma universidade.

Nesse contexto, Data Definition Language, DDL; Data Manipulation Language, DML; e Data Query Language, DQL, são utilizados para

(A) DDL: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DML: Inserir registros nas tabelas com detalhes específicos de um novo livro.

DQL: Recuperar todos os livros de um determinado autor.

(B) DDL: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DML: Atualizar a quantidade disponível de exemplares de um livro.

DQL: Inserir registros nas tabelas com detalhes específicos de um novo livro.

(C) DDL: Atualizar a quantidade disponível de exemplares de um livro.

DML: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DQL: Recuperar todos os livros emprestados por um usuário específico.

(D) DDL: Inserir registros nas tabelas com detalhes específicos de um novo livro.

DML: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DQL: Atualizar a quantidade disponível de exemplares de um livro.

(E) DDL: Recuperar todos os livros de um determinado autor.

DML: Criar tabelas para armazenar informações sobre livros, autores e editoras.

DQL: Inserir registros nas tabelas com detalhes específicos de um novo livro.

3. CESGRANRIO/IPEA/2024/Infraestrutura

Considere que um banco de dados foi criado para dar apoio à avaliação de instrumentos e políticas de gestão de trânsito no Brasil, nos últimos cinco anos. Os dados foram organizados e persistidos nas três seguintes tabelas, definidas de acordo com modelo relacional de dados: SINISTRO, com dados dos acidentes de trânsito; MUNICIPIO, com dados de municípios; e RODOVIA, com dados de rodovias estaduais e federais.

SINISTRO (cod-sinistro, data-e-hora, localizacao, cod-rodovia, cod-municipio, quantidade-de-vitimas) RODOVIA (cod-rodovia, nome, estadual-ou-federal)
MUNICIPIO (cod-municipio, uf, quantidade-de-habitantes)

Os atributos que formam as chaves primárias de cada tabela estão sublinhados.

Na tabela SINISTRO, há duas chaves estrangeiras: cod-rodovia, que indica onde ocorreu o sinistro, caso ele tenha ocorrido em uma rodovia, e cod-municipio, que indica em que município ocorreu o sinistro.

Nesse contexto, considere o seguinte comando SQL:

```
SELECT S.cod-rodovia, S.data-e-hora, quantidade-de-vitimas
FROM SINISTRO S
WHERE S.cod-rodovia IN (
SELECT R cod-rodovia
FROM RODOVIA R
WHERE R estadual-ou-federal = federal)
AND EXISTS (
SELECT *
FROM MUNICIPIO M
WHERE M.cod-municipio = S cod-municipio
AND M.quantidade-de-habitantes < 50000)
```

Os resultados produzidos pela execução desse comando apresentam o código da rodovia, a data e hora e a quantidade de vítimas de sinistros ocorridos em

a) rodovias federais que passam por municípios com menos de 50.000 habitantes.

b) rodovias federais, em municípios com menos de 50.000 habitantes.

b) rodovias federais que têm como origem ou destino municípios com menos de 50.000 habitantes.

d) município com menos de 50.000 habitantes ou em rodovias federais.

e) município com menos de 50.000 habitantes com duas ou mais rodovias federais.

4. CESGRANRIO/IPEA/2024/Ciência de Dados

Para a avaliação de políticas públicas na área de Segurança Alimentar e Nutricional, um município brasileiro utilizou dados persistidos em três relações (tabelas) organizadas de acordo com o seguinte modelo relacional:

PRODUTO (cod-produto, nome-produto, grupo-alimentar)

FORNECEDOR (CNPJ, nome-empresa, tipo)

COMPRADO (CNPJ, cod-produto, data, quantidade, valor)

Os atributos que formam as chaves primárias de cada tabela estão sublinhados.

Nesse contexto, considere o comando SQL apresentado a seguir.

```
SELECT P.cod-produto, SUM (quantidade)
FROM PRODUTO P, FORNECEDOR F, COMPRADO C
WHERE P.cod-produto = C.cod-produto
AND C.CNPJ = F.CNPJ
AND F.tipo = 'agricultura familiar'
GROUP BY P.cod-produto
HAVING SUM (quantidade) > 10000
```

Os resultados produzidos pela execução desse comando apresentam o código do produto e a soma das quantidades compradas dos produtos de

a) fornecedores com mais de 10.000 produtos distintos.

b) fornecedores do tipo 'agricultura familiar' que tiveram mais de 10.000 unidades compradas.

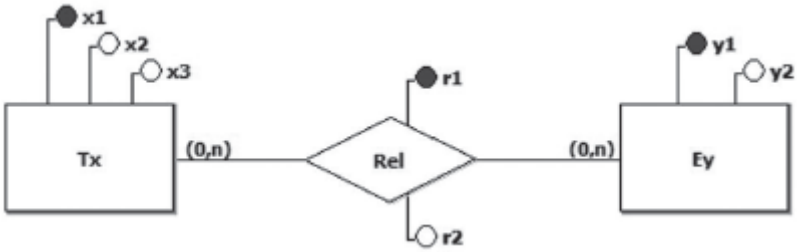
c) fornecedores do tipo 'agricultura familiar' que fornecem mais de 10.000 produtos distintos.

d) todos os fornecedores do tipo 'agricultura familiar'.

e) produtos que tiveram mais de 10.000 unidades compradas.

5. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023

Considere o seguinte diagrama E-R:



Foi criado um conjunto de tabelas relacionais, a partir do modelo E-R acima. Uma vez que as regras de transformações de entidades e relações para tabelas relacionais independem dos tipos de dados dos atributos, todos os atributos do modelo E-R acima foram tratados como itens de dados do tipo cadeia de caracteres (TEXT).

As tabelas resultantes são as seguintes:

```
CREATE TABLE TX (
        X1              TEXT            NOT NULL,
        X2              TEXT            NOT NULL,
        X3              TEXT            NOT NULL,
        PRIMARY KEY (X1));
CREATE TABLE EY (
         Y1             TEXT            NOT NULL,
         Y2             TEXT            NOT NULL,
         PRIMARY KEY (Y1));
```

Qual transformação da relação Rel irá preservar a semântica do diagrama E-R apresentado?

a) CREATE TABLE REL (

X1 TEXT NOT NULL,

Y1 TEXT NOT NULL,

R1 TEXT NOT NULL,

R2 TEXT NOT NULL,

PRIMARY KEY (X1, Y1),

FOREIGN KEY (X1)

REFERENCES TX (X1),

FOREIGN KEY (Y1)

REFERENCES EY (Y1));

b) CREATE TABLE REL (

X1 TEXT NOT NULL,

Y1 TEXT NOT NULL,

R1 TEXT NOT NULL,

R2 TEXT NOT NULL,

PRIMARY KEY (X1, R1),

FOREIGN KEY (X1)

REFERENCES TX (X1),

FOREIGN KEY (Y1)

REFERENCES EY (Y1));

c) CREATE TABLE REL (

X1 TEXT NOT NULL,

Y1 TEXT NOT NULL,

R1 TEXT NOT NULL,

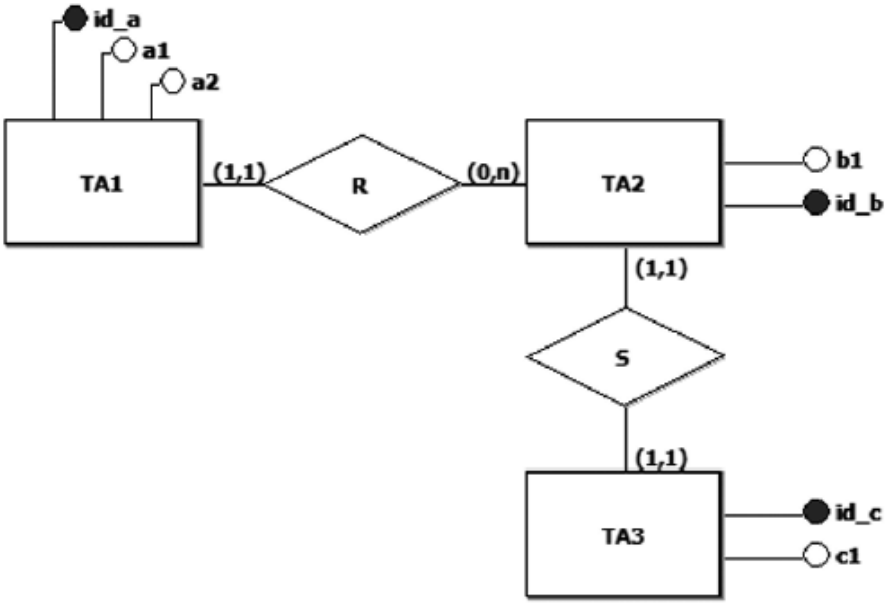
R2 TEXT NOT NULL,

```
    PRIMARY KEY (Y1, R1),
    FOREIGN KEY (X1)
            REFERENCES TX (X1),
    FOREIGN KEY (Y1)
            REFERENCES EY (Y1));
```

d)
```
CREATE TABLE REL (
    X1          TEXT        NOT NULL,
    Y1          TEXT        NOT NULL,
    R1          TEXT        NOT NULL,
    R2          TEXT        NOT NULL,
    PRIMARY KEY (X1, Y1, R1),
    FOREIGN KEY (X1)
            REFERENCES TX (X1),
    FOREIGN KEY (Y1)
            REFERENCES EY (Y1));
```

e)
```
CREATE TABLE REL (
    X1          TEXT        NOT NULL,
    Y1          TEXT        NOT NULL,
    R1          TEXT        NOT NULL,
    R2          TEXT        NOT NULL,
    PRIMARY KEY (R1),
    FOREIGN KEY (X1)
            REFERENCES TX (X1),
    FOREIGN KEY (Y1)
            REFERENCES EY (Y1));
```

6. CESGRANRIO - Ana Desenv (AgeRIO)/AgeRIO/Tecnologia da Informação/2023
Considere o diagrama E-R a seguir.



Para simplificar, todos os atributos desse modelo E-R devem ser considerados itens de dados do tipo cadeia de caracteres (TEXT).

A partir desse diagrama, foi produzido um conjunto de tabelas relacionais por meio da aplicação de regras de transformação.

Essas regras preservaram a semântica do modelo E-R, além de propiciarem mais eficiência nas operações de junção sobre as tabelas obtidas.

Qual conjunto de tabelas atende às transformações aplicadas?

a)

```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
ID_B        TEXT            NOT NULL,
PRIMARY KEY (ID_A),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
C1          TEXT            NOT NULL,
PRIMARY KEY (ID_B));
```

```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
PRIMARY KEY (ID_A));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
C1          TEXT            NOT NULL,
PRIMARY KEY (ID_B));

CREATE TABLE R (
ID_A        TEXT            NOT NULL,
ID_B        TEXT            NOT NULL,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_A)
    REFERENCES TA1(ID_A),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B));
```

b)

```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
PRIMARY KEY (ID_A));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_A        TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
C1          TEXT            NOT NULL,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_A)
    REFERENCES TA1(ID_A));
```

c)

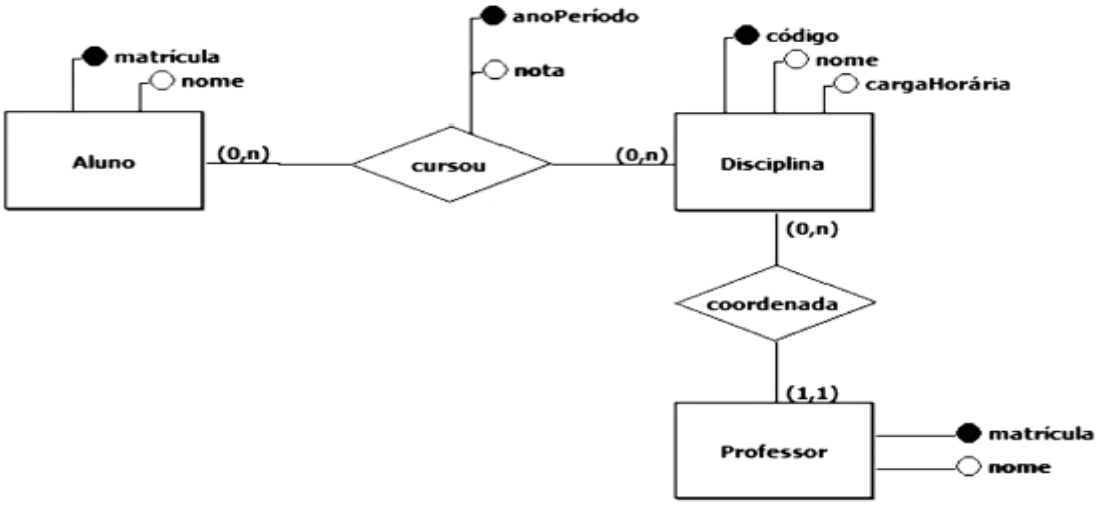
```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
ID_B        TEXT            NOT NULL,
PRIMARY KEY (ID_A),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_C)
    REFERENCES TA3(ID_C));

CREATE TABLE TA3 (
ID_C        TEXT            NOT NULL,
C1          TEXT            NOT NULL,
ID_B        TEXT            NOT NULL UNIQUE,
PRIMARY KEY (ID_C),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B));
```

d)

```
CREATE TABLE TA1 (
ID_A        TEXT            NOT NULL,
A1          TEXT            NOT NULL,
A2          TEXT            NOT NULL,
PRIMARY KEY (ID_A));

CREATE TABLE TA2 (
ID_B        TEXT            NOT NULL,
B1          TEXT            NOT NULL,
ID_A        TEXT            NOT NULL,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_A)
    REFERENCES TA1(ID_A));

CREATE TABLE TA3 (
ID_C        TEXT            NOT NULL,
C1          TEXT            NOT NULL,
PRIMARY KEY (ID_C));

CREATE TABLE S (
ID_B        TEXT            NOT NULL,
ID_C        TEXT            NOT NULL UNIQUE,
PRIMARY KEY (ID_B),
FOREIGN KEY (ID_B)
    REFERENCES TA2(ID_B),
FOREIGN KEY (ID_C)
    REFERENCES TA3(ID_C));
```

e)

7. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Infraestrutura/2023
Seja o seguinte modelo E-R:



Deverão ser criadas tabelas relacionais que preservem a semântica do modelo E-R apresentado e que atendam à 2a forma normal.

Qual esquema atende a esses requisitos?

a) CREATE TABLE ALUNO (
- MATRICULA INTEGER NOT NULL,
  NOME TEXT NOT NULL,
  PRIMARY KEY (MATRICULA));

CREATE TABLE DISCIPLINA (
CODIGO TEXT NOT NULL,
NOME TEXT NOT NULL,
CARGA_HOR INTEGER NOT NULL,
MAT_PROF INTEGER NOT NULL,
NOME_PROF TEXT NOT NULL,
PRIMARY KEY (CODIGO));

CREATE TABLE CURSOU (
MAT_ALUNO INTEGER NOT NULL,
COD_DISC TEXT NOT NULL,
ANO_PER TEXT NOT NULL,
NOTA NUMERIC (3,1) NOT NULL,
PRIMARY KEY (MAT_ALUNO, COD_DISC, ANO_PER),
FOREIGN KEY (MAT_ALUNO)
REFERENCES ALUNO (MATRICULA),
FOREIGN KEY (COD_DISC)
REFERENCES DISCIPLINA (CODIGO));

```
b) CREATE TABLE ALUNO (
        MATRICULA INTEGER NOT NULL,
        NOME TEXT NOT NULL,
        PRIMARY KEY (MATRICULA));
CREATE TABLE DISCIPLINA (
      CODIGO TEXT NOT NULL,
      NOME TEXT NOT NULL,
       CARGA_HOR INTEGER NOT NULL,
       MAT_PROF INTEGER NOT NULL,
       NOME_PROF TEXT NOT NULL,
       PRIMARY KEY (CODIGO));

CREATE TABLE CURSOU (
       MAT_ALUNO INTEGER NOT NULL,
       COD_DISC TEXT NOT NULL,
       ANO_PER TEXT NOT NULL,
       NOTA NUMERIC (3,1) NOT NULL,
       NOME_DISCIPLINA TEXT NOT NULL,
       PRIMARY KEY (MAT_ALUNO, COD_DISC, ANO_PER),
      FOREIGN KEY (MAT_ALUNO)
            REFERENCES ALUNO (MATRICULA),
      FOREIGN KEY (COD_DISC)
            REFERENCES DISCIPLINA (CODIGO));

c) CREATE TABLE ALUNO (
          MATRICULA INTEGER NOT NULL,
          NOME TEXT NOT NULL,
          PRIMARY KEY (MATRICULA));

CREATE TABLE PROFESSOR (
       MATRICULA INTEGER NOT NULL,
      NOME TEXT NOT NULL,
      PRIMARY KEY (MATRICULA));

CREATE TABLE DISCIPLINA (
      CODIGO TEXT NOT NULL,
      NOME TEXT NOT NULL,
      CARGA_HOR INTEGER NOT NULL,
      MAT_PROF INTEGER NOT NULL,
      PRIMARY KEY (CODIGO),
      FOREIGN KEY (MAT_PROF)

            REFERENCES PROFESSOR (MATRICULA));

CREATE TABLE CURSOU (
      MAT_ALUNO INTEGER NOT NULL,
      COD_DISC TEXT NOT NULL,
      ANO_PER TEXT NOT NULL,
```

```
      NOTA NUMERIC (3,1) NOT NULL,
     PRIMARY KEY (MAT_ALUNO, COD_DISC),
      FOREIGN KEY (MAT_ALUNO)
              REFERENCES ALUNO (MATRICULA),
       FOREIGN KEY (COD_DISC)
                REFERENCES DISCIPLINA (CODIGO));

d) CREATE TABLE ALUNO (
          MATRICULA INTEGER NOT NULL,
          NOME TEXT NOT NULL,
          PRIMARY KEY (MATRICULA));


CREATE TABLE PROFESSOR (
     MATRICULA INTEGER NOT NULL,
     NOME TEXT NOT NULL,
     PRIMARY KEY (MATRICULA));
CREATE TABLE DISCIPLINA (
       CODIGO TEXT NOT NULL,
       NOME TEXT NOT NULL,
       CARGA_HOR INTEGER NOT NULL,
      MAT_PROF INTEGER NOT NULL,
      PRIMARY KEY (CODIGO),
      FOREIGN KEY (MAT_PROF)
              REFERENCES PROFESSOR (MATRICULA));
CREATE TABLE CURSOU (
       MAT_ALUNO INTEGER NOT NULL,
       COD_DISC TEXT NOT NULL,
       ANO_PER TEXT NOT NULL,
       NOTA NUMERIC (3,1) NOT NULL,
       NOME_DISCIPLINA TEXT NOT NULL,
       PRIMARY KEY (MAT_ALUNO, COD_DISC, ANO_PER),
       FOREIGN KEY (MAT_ALUNO)
                REFERENCES ALUNO (MATRICULA),
        FOREIGN KEY (COD_DISC)
                 REFERENCES DISCIPLINA (CODIGO));

e)  CREATE TABLE ALUNO (
          MATRICULA INTEGER NOT NULL,
          NOME TEXT NOT NULL,
         PRIMARY KEY (MATRICULA));
CREATE TABLE PROFESSOR (
       MATRICULA INTEGER NOT NULL,
       NOME TEXT NOT NULL,
       PRIMARY KEY (MATRICULA));

CREATE TABLE DISCIPLINA (
      CODIGO TEXT NOT NULL,
```

```
    NOME TEXT NOT NULL,
    CARGA_HOR INTEGER NOT NULL,
    MAT_PROF INTEGER NOT NULL,
   PRIMARY KEY (CODIGO),
   FOREIGN KEY (MAT_PROF)
          REFERENCES PROFESSOR (MATRICULA));
CREATE TABLE CURSOU (
     MAT_ALUNO INTEGER NOT NULL,
     COD_DISC TEXT NOT NULL,
     ANO_PER TEXT NOT NULL,
     NOTA NUMERIC (3,1) NOT NULL,
     NOME_DISCIPLINA TEXT NOT NULL,
      PRIMARY KEY (MAT_ALUNO, COD_DISC, ANO_PER),
      FOREIGN KEY (MAT_ALUNO)
            REFERENCES ALUNO (MATRICULA),
      FOREIGN KEY (COD_DISC)
             REFERENCES DISCIPLINA (CODIGO));
```

8. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023

Uma empresa de investimentos financeiros busca identificar novas oportunidades de negócio para pessoas jurídicas, em especial dentre aquelas que têm como característica a adoção de governança ambiental, social e corporativa (conhecida como ESG, uma sigla em inglês).

Considere que existe um banco de dados nessa empresa com as seguintes tabelas (todas as chaves primárias são numéricas):

Empresa (CNPJ, razaoSocial, endereco)
Caracteristica (cod, sigla, nome)
Tem (CNPJ, cod)

Que comando SELECT do SQL retorna apenas o CNPJ e a razão social das empresas que não têm "ESG" como característica?

a) SELECT *

FROM Empresa

WHERE Caracteristica.Sigla <> 'ESG'

b) SELECT E.CNPJ, E.razaoSocial

FROM Empresa E

JOIN Tem T ON (E.CNPJ = T.CNPJ)

WHERE Tem.cod = 'ESG'

c) SELECT E.CNPJ, E.razaoSocial

FROM Empresa E

JOIN Tem T ON (E.CNPJ = T.CNPJ)

JOIN Caracteristica C ON (C.cod = T.cod)

WHERE C.nome <> 'ESG'

d) SELECT E.CNPJ, E.razaoSocial

FROM Empresa E

WHERE E.CNPJ NOT IN (

SELECT T.CNPJ

FROM Tem T

JOIN Caracteristica C ON (T.cod = C.cod)

WHERE C.sigla = 'ESG'

)

e) SELECT Empresa.*

FROM Empresa, Tem

WHERE Empresa.CNPJ = Tem.cod

AND Tem.cod <> 'ESG'

9. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023

O banco de dados de uma empresa que comercializa seguros pessoais possui as seguintes tabelas:

Pessoa (email, nome, unidadeFederativaNascimento, faixaEtaria)
UF (sigla, nome)
Faixa (nome, menorIdade, maiorIdade)

A coluna "unidadeFederativaNascimento" da tabela Pessoa é uma chave estrangeira que referencia a coluna "sigla" da tabela UF; a coluna "faixaEtaria" da tabela Pessoa é uma chave estrangeira que aponta para a coluna "nome" da tabela Faixa.
A tabela Faixa possui os seguintes dados:

nome	menorIdade	maiorIdade
Jovens	- não informada -	19
Adultos	20	59
Idosos	60	- não informada -

Considere o seguinte comando:

```
SELECT COUNT(*)
FROM Pessoa P, Faixa F
WHERE P.faixaEtaria = F.nome
AND P.unidadeFederativaNascimento = 'RJ'
AND F.maiorIdade <= 19
```

Esse comando SQL
a) apresenta quantas são as pessoas que estão na tabela Pessoa, que são jovens e que nasceram no estado do Rio de Janeiro.
b) apresenta o nome e o email de jovens nascidos no Rio de Janeiro.
c) agrupa pessoas por faixa etária e mostra quantos são os grupos com pessoas nascidas no Rio de Janeiro.
d) realiza uma operação equivalente à união de dois outros comandos SQL.
e) agrupa pessoas por UF e mostra quantos são os grupos com jovens.

10. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023
O banco de dados de uma empresa de investimentos financeiros possui as seguintes tabelas:
Empresa (CNPJ, razaoSocial, endereco)
UF (sigla, nome)

O que o comando SQL "SELECT CNPJ, sigla FROM Empresa, UF" recupera desse banco de dados?

a) A sigla da UF das sedes das empresas cadastradas.
b) Alguns pares de CNPJ e sigla, onde o nome da UF é igual à razão social da empresa.
c) O CNPJ das empresas cadastradas cuja sigla de UF esteja na tabela UF.
d) Pares de CNPJ e sigla de todas as empresas cadastradas com as UFs de suas respectivas sedes.
e) Todos os pares de CNPJ e sigla possíveis, de todas as empresas e de todas as UFs cadastradas.

11. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2023
A CNAE (Classificação Nacional de Atividades Econômicas), de responsabilidade do IBGE, possui códigos que são utilizados para caracterizar as atividades econômicas das empresas no Brasil. Por exemplo: empresas da área de construção de edifícios utilizam o CNAE de código 4120-4/00 para caracterizar a sua atividade econômica principal.

Considere que existe um banco de dados em uma empresa, que concede empréstimos a outras empresas, com as seguintes tabelas:

Empresa (CNPJ, razaoSocial, endereco, atividade)
CNAE (codigo, descricao)

A coluna "atividade" da tabela Empresa é uma chave estrangeira que referencia a coluna "codigo" da tabela CNAE.

Que comando SELECT do SQL retorna apenas o CNPJ e a razão social das empresas cuja atividade econômica principal é a construção de edifícios (código 4120-4/00)?

a) SELECT *

FROM Empresa E, CNAE C

WHERE E.atividade = C.codigo

b) SELECT *

FROM Empresa

WHERE atividade = 'construção de edifícios'

c) SELECT CNPJ, razaoSocial

FROM CNAE

WHERE codigo = '4120-4/00'

d) SELECT CNPJ, razaoSocial

FROM Empresa E, CNAE C

WHERE E.atividade = C.codigo

AND C.codigo = 'construção de edifícios'

e) SELECT CNPJ, razaoSocial

FROM Empresa

WHERE atividade = '4120-4/00'

12. CESGRANRIO - PTNM (TRANSPETRO)/TRANSPETRO/Informática/2023

Uma empresa aérea possui um sistema de informação para registrar as reservas de passagens de seus passageiros nos voos que oferece. O esquema desse banco de dados foi criado com os seguintes comandos SQL:

```
CREATE TABLE PASSAGEIRO (
        CODIGO                  NUMERIC(5) NOT NULL,
         NOME                   TEXT NOT NULL,
         PRIMARY KEY (CODIGO));

CREATE TABLE VOO (
        NUMERO                  NUMERIC(3) NOT NULL,
        ORIGEM                  TEXT NOT NULL,
        DESTINO                 TEXT NOT NULL,
        NUM_LUGARES             NUMERIC(3) NOT NULL,
        PRIMARY KEY (NUMERO));

CREATE TABLE RESERVA (
        NUM_VOO                 NUMERIC(3) NOT NULL,
        CD_PASS                 NUMERIC(5) NOT NULL,
         PRIMARY KEY (NUM_VOO, CD_PASS),
         FOREIGN KEY (NUM_VOO)
                REFERENCES VOO(NUMERO),
         FOREIGN KEY (CD_PASS)
                REFERENCES PASSAGEIRO(CODIGO));
```

Na Figura a seguir, são exibidos os estados atuais dessas tabelas.

--------TABELA PASSAGEIRO----------		----------TABELA VOO----------				--------TABELA RESERVA-----	
CODIGO	NOME	NUMERO	ORIGEM	DESTINO	NUM_LUGARES	NUM_VOO	CD_PASS
11232	FABIO CARNEIRO	357	RIO DE JANEIRO	FORTALEZA	30	357	65789
13121	CARLOS AMARAL	635	SALVADOR	RIO DE JANEIRO	20	357	13121
22578	ANA MARIA RIOS	784	FORTALEZA	BRASILIA	20	357	56390
22667	CECILIA LOPES	998	PORTO ALEGRE	CURITIBA	20	357	22578
23799	KAREN RIBEIRO					357	22667
44532	VICENTE DE CARVALHO					357	56123

44677	JULIA VASCONCELOS					357	44909
44909	ALEX MEDEIROS					635	45131
45131	FLAVIA NASCIMENTO					635	89567
51734	EDSON RIBEIRO					635	11232
56123	PEDRO COIMBRA					635	44532
56390	FELIPE DE SOUZA					635	78980
65123	POLIANA PEDROSA					635	44677
65789	JOANA RODRIGUES					784	23799
79980	ROSANA DA SILVA					784	51734
89567	IARA DE MELO					784	65123

Qual comando SQL será executado sem produzir erro?
a) DELETE FROM PASSAGEIRO WHERE NOME = 'VICENTE DE CARVALHO';
b) DELETE FROM VOO WHERE DESTINO = 'CURITIBA';
c) INSERT INTO RESERVA VALUES(998, 56000);
d) INSERT INTO PASSAGEIRO VALUES(56390, 'RICARDO GONÇALVES');
e) UPDATE VOO SET NUMERO = 532 WHERE DESTINO = 'RIO DE JANEIRO';

13. CESGRANRIO - PTNM (TRANSPETRO)/TRANSPETRO/Informática/2023
Na Figura abaixo, é exibido o estado atual de uma tabela que registra as disciplinas de um curso e seus pré-requisitos.

--- TABELA CURSO----	
DISCIPLINA	PRE_REQUISITO
PROJETOS DE SISTEMAS	BANCO DE DADOS I
PROJETO FINAL	PROJETO DE SISTEMAS
BANCO DE DADOS I	MODELAGEM DE DADOS

PROGRAMAÇÃO	ESTRUTURAS DE DADOS

Uma consulta SQL nessa tabela foi executada e produziu como resposta uma relação com duas linhas, cujos valores são exibidos abaixo.
PROJETO DE SISTEMAS
BANCO DE DADOS I

Qual consulta SQL foi executada?
a) SELECT DISCIPLINA FROM CURSO
UNION
SELECT PRE_REQUISITO FROM CURSO;
b) SELECT DISCIPLINA, PRE_REQUISITO FROM CURSO
WHERE DISCIPLINA = 'PROJETO DE SISTEMAS' AND
PRE_REQUISITO = 'BANCO DE DADOS I' OR
DISCIPLINA = 'BANCO DE DADOS I' AND PRE_REQUISITO = 'PROJETO DE SISTEMAS';
c) SELECT DISCIPLINA FROM CURSO
INTERSECT
SELECT PRE_REQUISITO FROM CURSO;
d) SELECT C1.DISCIPLINA, C2.PRE_REQUISITO
FROM CURSO C1
RIGHT JOIN CURSO C2
ON C1.DISCIPLINA = C2.PRE_REQUISITO;
e) SELECT C1.DISCIPLINA, C2.PRE_REQUISITO
FROM CURSO C1
LEFT JOIN CURSO C2
ON C1.DISCIPLINA = C2.PRE_REQUISITO;

14. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Infraestrutura/2023
Considere um banco de dados de uma empresa contendo a tabela HIERARQUIA, que possui as colunas Chefe e Subordinado definindo uma hierarquia de cargos. Considere, também, uma instância que permite ilustrar as informações contidas nesta tabela.

CHEFE	SUBORDINADO
----------	----------
Pedro	Renata
Roberto	Beatriz
Gabriela	Fernando
Lucas	Daniela
João	Rafaela
Carlos	Juliana
Beatriz	Carlos

Rafaela	Marcelo
Pedro	Roberto
Marcelo	Gabriela
Renata	Felipe
João	Pedro
Daniela	Antônio
Felipe	Clara
Juliana	Lucas

Uma consulta feita para determinar os subordinados, diretos ou indiretos, do Carlos retorna os seguintes nomes: {Juliana, Lucas, Daniela e Antônio}.

A expressão, na linguagem de consultas SQL, para esta consulta é

a)
```
WITH RECURSIVE subordinados AS (
        SELECT SUBORDINADO
        FROM HIERARQUIA
        WHERE CHEFE = 'Carlos'

        UNION

       SELECT t.SUBORDINADO
       FROM HIERARQUIA t
       JOIN subordinados s ON t.CHEFE = s.SUBORDINADO
)
SELECT SUBORDINADO FROM subordinados;
```

b)
```
WITH RECURSIVE subordinados AS (
        SELECT CHEFE
        FROM HIERARQUIA
        WHERE SUBORDINADO = 'Carlos'

        UNION ALL
        SELECT t.CHEFE
        FROM HIERARQUIA t
        JOIN subordinados s ON t.SUBORDINADO = s.CHEFE
)
SELECT CHEFE FROM subordinados;
```

c)
```
WITH RECURSIVE subordinados AS (
        SELECT SUBORDINADO
        FROM HIERARQUIA
        WHERE CHEFE = 'Carlos'

        UNION ALL
```

```
        SELECT t.SUBORDINADO
         FROM HIERARQUIA t
         JOIN subordinados s ON t.CHEFE = s.SUBORDINADO
)
SELECT SUBORDINADO FROM subordinados;
```

```
d) WITH subordinados AS (
        SELECT SUBORDINADO
         FROM HIERARQUIA
         WHERE CHEFE = 'Carlos'

          UNION ALL

          SELECT t.SUBORDINADO
          FROM HIERARQUIA t
          JOIN subordinados s ON t.SUBORDINADO = s.CHEFE
)
SELECT SUBORDINADO FROM subordinados;
```

```
e) WITH RECURSIVE subordinados AS (
        SELECT SUBORDINADO
        FROM HIERARQUIA
        WHERE CHEFE = 'Carlos'

        UNION ALL

        SELECT t.CHEFE
        FROM HIERARQUIA t
       JOIN subordinados s ON t.SUBORDINADO = s.CHEFE
)
SELECT SUBORDINADO FROM subordinados;
```

15. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Processos de Negócios/2023

Considere o esquema relacional simplificado de um banco de dados de uma Universidade, contendo, ao menos, as seguintes tabelas:

Departamentos (ID_Departamento, Nome, Localizacao)
Professores (ID_Professor, Nome, ID_Departamento)
Alunos (ID_Aluno, Nome, ID_Professor_Orientador)

Entre outras restrições que devem ser consideradas, sabe-se que alunos podem ser orientados por professores e que todo professor pertence a um departamento.
Considerando-se o esquema relacional apresentado, o trecho de código em linguagem SQL DDL que respeita todas as restrições de integridade citadas é

```
a) CREATE TABLE Departamentos (
        ID_Departamento INT PRIMARY KEY,
        Nome VARCHAR(100),
        Localizacao VARCHAR(100)
   );
CREATE TABLE Professores (
       ID_Professor INT PRIMARY KEY,
       Nome VARCHAR(100),
       ID_Departamento INT
);
CREATE TABLE Alunos (
      ID_Aluno INT PRIMARY KEY,
      Nome VARCHAR(100),
      ID_Professor_Orientador INT
);
b) CREATE TABLE Departamentos (
         ID_Departamento INT PRIMARY KEY,
         Nome VARCHAR(100),
         Localizacao VARCHAR(100)
);
CREATE TABLE Professores (
      ID_Professor INT PRIMARY KEY,
      Nome VARCHAR(100),
      ID_Departamento INT,
      FOREIGN KEY (ID_Departamento) REFERENCES Departamentos(ID_Departamento)
);
CREATE TABLE Alunos (
       ID_Aluno INT PRIMARY KEY,
       Nome VARCHAR(100),
       ID_Professor_Orientador INT,
      FOREIGN KEY (ID_Professor_Orientador) REFERENCES Professores(ID_Professor)
);
c) CREATE TABLE Departamentos (
         ID_Departamento INT PRIMARY KEY,
         Nome VARCHAR(100),
         Localizacao VARCHAR(100)
);
CREATE TABLE Professores (
      ID_Professor INT PRIMARY KEY,
      Nome VARCHAR(100),
      ID_Departamento INT
);
CREATE TABLE Alunos (
      ID_Aluno INT PRIMARY KEY,
      Nome VARCHAR(100),
      ID_Professor_Orientador INT,
      FOREIGN KEY (ID_Professor_Orientador) REFERENCES Professores(ID_Professor)
);
```

```
d) CREATE TABLE Departamentos (
        ID_Departamento INT PRIMARY KEY,
        Nome VARCHAR(100),
        Localizacao VARCHAR(100)
);
CREATE TABLE Professores (
      ID_Professor INT PRIMARY KEY,
      Nome VARCHAR(100),
      ID_Departamento INT
);
CREATE TABLE Alunos (
     ID_Aluno INT PRIMARY KEY,
     Nome VARCHAR(100),
     ID_Professor_Orientador INT
);
e) CREATE TABLE Departamentos (
        ID_Departamento INT PRIMARY KEY,
        Nome VARCHAR(100),
        Localizacao VARCHAR(100)
);
CREATE TABLE Professores (
     ID_Professor INT,
     Nome VARCHAR(100),
     ID_Departamento INT,
     PRIMARY KEY (ID_Professor),
     FOREIGN KEY (ID_Departamento) REFERENCES Departamentos(ID_Departamento)
);
CREATE TABLE Alunos (
      ID_Aluno INT PRIMARY KEY,
      Nome VARCHAR(100),
      ID_Professor_Orientador INT
);
```

16. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Processos de Negócios/2023

Considere a tabela Familia, apresentada abaixo, que possui as colunas Pai e Filho e uma instância exemplo que permite ilustrar as informações contidas nessa tabela.

Pai	Filho
Andre	Carlos
Andre	Claudio
Bruno	Diego
Bruno	Daniel
Carlos	Eriko
Carlos	Fabio

Daniel	Gustavo
Diego	Heleno
Eriko	Isidoro
Fabio	João

Uma consulta SQL feita para exibir os nomes dos netos de Carlos retorna os nomes Isidoro e João. A expressão, em linguagem SQL, dessa consulta é

a) SELECT f2.filho

FROM Familia f1 INNER JOIN Familia f2 ON f1.filho = f2.pai

WHERE f1.pai = 'Carlos';

b) SELECT f1.pai

FROM Familia f1 INNER JOIN Familia f2 ON f1.pai = f2.filho

WHERE f1.filho = 'Carlos';

c) SELECT f1.pai

FROM Familia f1 INNER JOIN Familia f2 ON f1.filho = f2.pai

WHERE f1.pai = 'Carlos';

d) SELECT

f2.filho FROM Familia f1 INNER JOIN Familia f2 ON f1.pai = f2.pai

WHERE f1.pai = 'Carlos';

e) SELECT f2.filho

FROM Familia f1 INNER JOIN Familia f2 ON f1.filho = f2.filho

WHERE f1.pai = 'Carlos';

17. CESGRANRIO - PTNS (TRANSPETRO)/TRANSPETRO/Análise de Sistemas/Infraestrutura/2023

Gatilhos (*triggers*) e procedimentos armazenados (*stored procedures*) são componentes fundamentais em sistemas de gerenciamento de banco de dados (SGBDs) relacionais. Tanto os gatilhos quanto os procedimentos armazenados desempenham papéis vitais e, muitas vezes, complementares em aplicações baseadas em banco de dados relacionais, sendo escolhidos de acordo com as necessidades específicas de uma aplicação ou de um sistema. A respeito de gatilhos e de procedimentos armazenados, tem-se que

a) tanto os gatilhos quanto os procedimentos armazenados são executados manual e explicitamente por um usuário ou por um programa de aplicação, e não podem ser disparados automaticamente por eventos de atualização em um banco de dados.
b) tanto os gatilhos quanto os procedimentos armazenados são disparados automaticamente por eventos de atualização em um banco de dados, e não podem ser ativados manual e explicitamente por um usuário ou por um programa de aplicação.
c) gatilhos, ao contrário dos procedimentos armazenados, não podem expressar a lógica de regras de negócios, e são utilizados exclusivamente para validação de dados.
d) gatilhos são disparados automaticamente em resposta a eventos de atualização em um banco de dados, enquanto procedimentos armazenados precisam ser chamados explicitamente.
e) gatilhos e procedimentos armazenados não podem coexistir em um mesmo sistema de banco de dados, pois possuem funcionalidades idênticas e redundantes.

18. CESGRANRIO - Tec Cien (BASA)/BASA/Tecnologia da Informação/2022

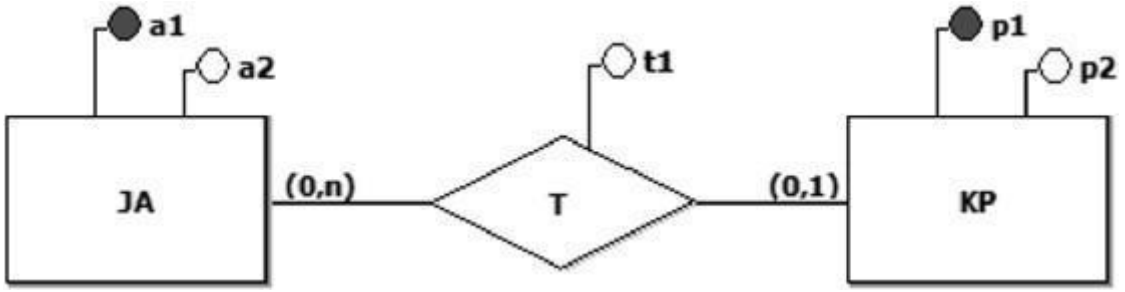
Em banco de dados, os gatilhos são utilizados, entre outros objetivos, para implementar restrições de integridade. Considere que em um banco de dados de um banco comercial há duas tabelas, CLIENTE (chave primária CPF) e SERVICO (chave primária composta por CPF e identificação do serviço), e há a restrição de que um cliente não pode estar associado a mais de cinco serviços.

Que definição deve ser utilizada para implementar essa restrição por meio de um gatilho?

a) BEFORE INSERT ON CLIENTE
b) AFTER INSERT ON CLIENTE
c) BEFORE INSERT ON SERVICO
d) AFTER INSERT ON SERVICO
e) WHEN CLIENTE INSERT ON SERVICO

19. CESGRANRIO - PNS (ELETRONUCLEAR)/ELETRONUCLEAR/Analista de Sistemas/Aplicações e Segurança de TIC/2022

Considere o diagrama E-R abaixo.



Qual esquema relacional preserva a semântica desse modelo E-R?

a)

```
CREATE TABLE JA (
A1 TEXT NOT NULL,
A2 TEXT NOT NULL,
PRIMARY KEY (A1));
 CREATE TABLE KP (
P1 TEXT NOT NULL,
P2 TEXT NOT NULL,
A1 TEXT NULL,
T1 TEXT NULL,
PRIMARY KEY (P1),
FOREIGN KEY (A1)
REFERENCES JA (A1));
```

b)

```
CREATE TABLE JA (
A1 TEXT NOT NULL,
A2 TEXT NOT NULL,
PRIMARY KEY (A1));
 CREATE TABLE KP (
P1 TEXT NOT NULL,
P2 TEXT NOT NULL,
PRIMARY KEY (P1));
 CREATE TABLE T (
P1 TEXT NOT NULL,
```

A1 TEXT NOT NULL,

T1 TEXT NOT NULL,

PRIMARY KEY (P1, A1),

FOREIGN KEY (P1)

REFERENCES KP (P1),

FOREIGN KEY (A1)

REFERENCES JA (A1));

c)

CREATE TABLE JA (

A1 TEXT NOT NULL,

A2 TEXT NOT NULL,

PRIMARY KEY (A1));

CREATE TABLE KP (

P1 TEXT NOT NULL,

P2 TEXT NOT NULL,

PRIMARY KEY (P1));

CREATE TABLE T (

P1 TEXT NOT NULL,

A1 TEXT NOT NULL,

T1 TEXT NOT NULL,

PRIMARY KEY (P1),

FOREIGN KEY (P1)

REFERENCES KP (P1),

FOREIGN KEY (A1)

REFERENCES JA (A1));

d)

CREATE TABLE T (

A1 TEXT NOT NULL,

A2 TEXT NOT NULL,

P1 TEXT NOT NULL,

P2 TEXT NOT NULL,

T1 TEXT NOT NULL,

PRIMARY KEY (P1, A1));

e)

CREATE TABLE JA (

A1 TEXT NOT NULL,

A2 TEXT NOT NULL,

P1 TEXT NULL,

T1 TEXT NULL,

PRIMARY KEY (A1)

FOREIGN KEY (P1)

REFERENCES KP (P1));

CREATE TABLE KP (

P1 TEXT NOT NULL,

P2 TEXT NOT NULL,

PRIMARY KEY (P1));

20. CESGRANRIO - Tec Cien (BASA)/BASA/Tecnologia da Informação/2022

Considere que em um banco de dados de um banco comercial há duas tabelas:
PESSOA_FISICA (CPF, nome, email, telefone)
CLIENTE (CPF, nome, email, telefone).

Um funcionário de TI recebeu a tarefa de identificar corretamente quais pessoas físicas, cadastradas na tabela PESSOA_FISICA, ainda não eram clientes, pois não estavam cadastradas na tabela CLIENTE. Para isso, ele utilizou um comando SELECT em SQL. Que trecho, em SQL, faz parte de uma das possíveis soluções para essa tarefa?

a) … WHERE PESSOA_FISICA.CPF NOT IN (SELECT CPF FROM CLIENTE…
b) … HAVING PESSOA_FISICA.CPF != CLIENTE.CPF…
c) … WHERE PESSOA_FISICA.CPF <> CLIENTE.CPF...
d) … DISTINCT PESSOA_FISICA.CPF FROM CLIENTE WHERE ...
e) … IN PESSOA_FISICA BUT NOT IN CLIENTE...

21. CESGRANRIO - Tec Cien (BASA)/BASA/Tecnologia da Informação/2022

As tabelas a seguir fazem parte do banco de dados da área de recursos humanos de uma empresa. Elas registram os dados referentes aos empregados e aos seus dependentes.

```
CREATE TABLE EMPREGADO (
   MATRICULA INTEGER NOT NULL,
    NOME TEXT NOT NULL,
    DATA_NASC TEXT NOT NULL,
    CERT_RESRV INTEGER UNIQUE NULL,
    PRIMARY KEY (MATRICULA));

CREATE TABLE DEPENDENTE (
    MAT_EMP INTEGER NOT NULL,
    NUM_SEQ INTEGER NOT NULL,
    NOME TEXT NOT NULL,
    DATA_NASC TEXT NOT NULL,
    PRIMARY KEY (MAT_EMP, NUM_SEQ)
    FOREIGN KEY (MAT_EMP)
REFERENCES EMPREGADO (MATRICULA));
```

O estado corrente desse banco de dados é exibido nas figuras abaixo.

EMPREGADO

MATRICULA	NOME	DATA_NASC	CERT_RESRV
11111	Paulo Menezes	24/05/1991 00:00	234811
22222	Ana Maria Carvalho	25/07/1983 00:00	null
33333	Alexandre Cardoso	11/08/1989 00:00	101678

DEPENDENTE

MAT_EMP	NUM_SEQ	NOME	DATA_NAS
22222	1	Valéria	31/12/2017 00:00
22222	2	Pedro	09/06/2015 00:00
33333	1	Joana	22/03/2019 00:00
33333	2	Mariana	25/07/2020 00:00

Qual comando INSERT irá falhar, na tentativa de incluir uma nova linha em uma das tabelas desse banco de dados?

a) INSERT INTO DEPENDENTE(NUM_SEQ,MAT_EMP,DATA_NASC,NOME)

VALUES(00,11111,datetime('2015-01-06'),'Luiz Paulo');

b) INSERT INTO DEPENDENTE(NOME,DATA_NASC,NUM_SEQ,MAT_EMP)

VALUES('Maria Paula',datetime('2017-08-11'),3,11111);

c) INSERT INTO EMPREGADO VALUES(55555,'Antônia Pinto',datetime('1994-04-01'),NULL);
d) INSERT INTO EMPREGADO VALUES(66666,'Adriana Andrade',datetime('1985-06-04'));
e) INSERT INTO EMPREGADO VALUES(44444,'Nilce Peçanha',datetime('1999-09-06'),'');

22. CESGRANRIO - PNS (ELETRONUCLEAR)/ELETRONUCLEAR/Analista de Sistemas/Aplicações e Segurança de TIC/2022

O controle diário da utilização de passes de metrô em uma cidade é feito por programas que utilizam um banco de dados composto pelas seguintes tabelas:

```
CREATE TABLE PASSE (
NUM INTEGER NOT NULL,
DATA_EXP DATE NOT NULL,
NUM_VIAGENS INTEGER NOT NULL,
PRIMARY KEY (NUM))

CREATE TABLE REG_VIAGEM (
NUM INTEGER NOT NULL,
NUM_ROLETA INTEGER NOT NULL,
DATA_VIAGEM DATE NOT NULL,
FOREIGN KEY (NUM)
REFERENCES PASSE (NUM))
```

A tabela PASSE contém uma linha para cada passe vendido pela empresa que administra o metrô. A coluna DATA_EXP informa a data de emissão do passe, e a coluna NUM_VIAGENS

informa o número de viagens em que o passe poderá ser usado (número máximo de viagens). Este número não sofre alteração ao longo do tempo.

A tabela REG_VIAGEM contém uma linha para cada viagem em que o passe foi usado. A coluna NUM_ROLETA informa a roleta na qual o passe foi inserido, e a coluna DATA_VIAGEM informa a data em que o usuário inseriu o passe na roleta.

Qual consulta SQL permite obter os números dos passes que nunca foram usados, juntamente com os números dos passes que já esgotaram o número de viagens realizadas?

a)

```
SELECT A.NUM

FROM PASSE A

WHERE A.NUM_VIAGENS = (SELECT COUNT(*) FROM REG_VIAGEM

WHERE NUM = A.NUM) OR A.NUM NOT IN (SELECT NUM FROM REG_VIAGEM)
```

b)

```
SELECT DISTINCT (A.NUM)

FROM PASSE A

LEFT JOIN REG_VIAGEM B

ON A.NUM=B.NUM OR A.NUM NOT IN (SELECT NUM FROM REG_VIAGEM)
```

c)

```
SELECT A.NUM

FROM PASSE A, REG_VIAGEM B

WHERE A.NUM_VIAGENS = (SELECT COUNT(*) FROM REG_VIAGEM

WHERE NUM = A.NUM) OR A.NUM != B.NUM
```

d)

```
SELECT A.NUM

FROM PASSE A

INTERSECT

SELECT A.NUM

FROM PASSE A
```

WHERE A.NUM NOT IN (SELECT NUM FROM REG_VIAGEM)

e)

SELECT NUM

FROM PASSE

EXCEPT

SELECT NUM

FROM REG_VIAGEM

23. CESGRANRIO - PNS (ELETRONUCLEAR)/ELETRONUCLEAR/Analista de Sistemas/Aplicações e Segurança de TIC/2022

O controle diário da utilização de passes de metrô em uma cidade é feito por programas que utilizam um banco de dados composto pelas seguintes tabelas:

```
CREATE TABLE PASSE (
NUM INTEGER NOT NULL,
DATA_EXP DATE NOT NULL,
NUM_VIAGENS INTEGER NOT NULL,
PRIMARY KEY (NUM))
 CREATE TABLE REG_VIAGEM (
NUM INTEGER NOT NULL,
NUM_ROLETA INTEGER NOT NULL,
DATA_VIAGEM DATE NOT NULL,
FOREIGN KEY (NUM)
REFERENCES PASSE (NUM))
```

A tabela PASSE contém uma linha para cada passe vendido pela empresa que administra o metrô. A coluna DATA_EXP informa a data de emissão do passe, e a coluna NUM_VIAGENS informa o número de viagens em que o passe poderá ser usado (número máximo de viagens). Este número não sofre alteração ao longo do tempo.

A tabela REG_VIAGEM contém uma linha para cada viagem em que o passe foi usado. A coluna NUM_ROLETA informa a roleta na qual o passe foi inserido, e a coluna DATA_VIAGEM informa a data em que o usuário inseriu o passe na roleta.

Qual comando SQL NÃO produzirá um erro de tempo de execução independentemente dos dados existentes nas duas tabelas que fazem parte do banco de dados?

a) INSERT INTO REG_VIAGEM VALUES (7777, 720, '2022-04-21')
b) UPDATE PASSE SET NUM = 8888 WHERE NUM = 4444
c) DELETE FROM REG_VIAGEM WHERE DATA_VIAGEM = '2021-09-02';
d) INSERT INTO PASSE (NUM_VIAGENS, NUM) VALUES (10, 9999)
e) DELETE FROM PASSE WHERE DATA_EXPIR = '2021-08-11'

24. CESGRANRIO - Tec Cien (BASA)/BASA/Tecnologia da Informação/2021
Um programador experiente estava revisando o código de um estagiário e detectou a instrução SQL abaixo, complicada demais para seu objetivo.

SELECT * FROM Compras where codProduto not in (select codProduto from Produtos where codProduto<3 or valor<4000)

Para simplificar o código, sem alterar a resposta, a instrução apresentada acima pode ser substituída por
a) SELECT * FROM Compras where codProduto<3 or valor<4000
b) SELECT * FROM Compras where codProduto<3 and valor<4000
c) SELECT * FROM Compras where codProduto<=3 and valor<=4000
d) SELECT * FROM Compras where codProduto>=3 and valor>=4000
e) SELECT * FROM Compras where codProduto>=3 or valor>=4000

25. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2021
Para gerar um gráfico de dispersão, um programador precisava consultar duas tabelas, T1 e T2. Ele decidiu, então, usar um LEFT JOIN, como em

SELECT * FROM T1 LEFT JOIN T2 USING (CHAVE);

Essa consulta resultou em 214 linhas.
Por motivos de segurança, ele fez outra consulta semelhante, apenas trocando o LEFT JOIN por um JOIN, e essa segunda consulta resultou em 190 linhas.

O que pode explicar corretamente a quantidade diferente de linhas nas consultas realizadas?
a) CHAVE é a chave primária de T1, mas apenas um campo da chave primária de T2.
b) CHAVE é a chave primária de T2, mas apenas um campo da chave primária de T1.
c) T1 possui linhas cujo valor de CHAVE não está presente na T2.
d) T2 possui linhas cujo valor de CHAVE não está presente na T1.
e) T2 possui linhas com todas as chaves presentes em T1, mas com campos nulos.

26. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2021
Ao coletar dados em um sistema compatível com SQL 2008 para fazer uma análise de dados, um programador percebeu que havia dois campos, data_de_nascimento e data_de_emissão_RG, em que o valor de data_de_emissão_RG sempre deve ser mais recente que data_de_nascimento. Percebeu, porém, que em 10% das linhas acontecia o inverso, isto é, data_de_nascimento era mais recente que data_de_emissão_RG. Ele corrigiu os dados nessas linhas, verificando que estavam consistentemente trocados, mas, preocupado que tal problema voltasse a acontecer, resolveu solicitar ao DBA uma alteração da tabela, de forma que data_de_emissão_RG sempre tivesse que ser mais recente que data_de_nascimento.
O DBA atendeu adequadamente a esse pedido do programador por meio de uma restrição em SQL 2008 do tipo
a) CHECK
b) INSPECT
c) TEST
d) VALIDATE
e) VERIFY

27. CESGRANRIO - Esc BB/BB/Agente de Tecnologia/2021

Após um treinamento em SQL padrão 2008, compatível com ambiente MS SQL Server 2008, um escriturário do Banco Z precisou utilizar os conhecimentos adquiridos para criar uma tabela no sistema de banco de dados desse Banco. A tabela a ser criada é de fornecedores, e tem os seguintes campos: CNPJ, nome do fornecedor e país de origem. As características gerais da tabela são:

- o campo CNPJ é chave primária e contém 14 caracteres, sendo que os caracteres devem se ater aos numéricos ["0" a "9"], e o caractere zero "0" não pode ser ignorado, seja qual for a posição dele (início, meio ou fim da chave);
- o campo NOME contém 20 caracteres e aceita valor nulo;
- o campo PAIS contém 15 caracteres e não aceita valor nulo.

Nesse contexto, o comando SQL2008 que cria uma tabela com as características descritas acima é

a) CREATE TABLE Fornecedores

(CNPJ INTEGER PRIMARY KEY,

NOME VARCHAR(20) ACCEPT NULL,

PAIS VARCHAR(15) NOT NULL)

b) CREATE TABLE Fornecedores

(CNPJ CHAR(14) PRIMARY KEY,

NOME VARCHAR(20),

PAIS VARCHAR(15) NOT NULL)

c) CREATE TABLE Fornecedores

(CNPJ CHAR(14) NOT NULL,

NOME VARCHAR(20) NOT NULL,

PAIS VARCHAR(15))

d) CREATE TABLE Fornecedores

(CNPJ CHAR(14),

NOME VARCHAR(20) NOT NULL,

PAIS VARCHAR(15) NOT NULL),

PRIMARY KEY (CNPJ)

e) CREATE TABLE Fornecedores

(CNPJ INTEGER(14) NOT NULL,

NOME VARCHAR(20),

PAIS VARCHAR(15) NOT NULL),

PRIMARY KEY (CNPJ)

28. CESGRANRIO - TBN (CEF)/CEF/Tecnologia da Informação/2021

Um sistema gerenciador de banco de dados utiliza metadados, persistidos em bancos de dados, para

a) avaliar o seu próprio desempenho, considerando metas pré-estabelecidas pelo administrador do banco de dados.
b) decidir que páginas de dados sujas na sua cache precisam ser persistidas em disco.
c) implementar o isolamento entre transações concorrentes.
d) permitir a restauração de um estado íntegro do banco de dados, em caso de falha.
e) validar comandos SQL informados por um usuário.

29. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação

Considere as Tabelas a seguir para responder a questão.

Essas Tabelas fazem parte do esquema de um banco de dados usado por uma associação de criadores de cães para organizar informações sobre os torneios que ela promove.

```
CREATE TABLE CAO (
   COD           NUMBER(5)      NOT NULL,
   NOME          VARCHAR2(50)   NOT NULL,
   RACA          VARCHAR2(50)   NOT NULL,
   NOME_PAI      VARCHAR2(50),
   NOME_PROPR    VARCHAR2(50)   NOT NULL,
   CONSTRAINT CAO_PK PRIMARY KEY (COD)
)
CREATE TABLE COMPETICAO (
   COD           NUMBER(5)      NOT NULL,
   DESCR         VARCHAR2(50)   NOT NULL,
   CONSTRAINT COMPETICAO_PK PRIMARY KEY (COD)
)
CREATE TABLE ARBITRO (
   COD           NUMBER(5)      NOT NULL,
   NOME          VARCHAR2(50)   NOT NULL,
   CONSTRAINT ARBITRO_PK PRIMARY KEY (COD)
)
CREATE TABLE PARTICIPACAO (
   COD_CAO       NUMBER(5)       NOT NULL,
   COD_COMP      NUMBER(5)       NOT NULL,
   COLOCACAO     NUMBER(4)       NOT NULL,
   CONSTRAINT PARTICIPACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP),
   CONSTRAINT PARTICIPACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT PARTICIPACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD)
)
CREATE TABLE AVALIACAO (
   COD_CAO       NUMBER(5)       NOT NULL,
   COD_COMP      NUMBER(5)       NOT NULL,
   COD_ARBTR     NUMBER(5)       NOT NULL,
   NOTA_ARBTR    NUMBER(3,1)     NOT NULL,
   CONSTRAINT AVALIACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP,COD_ARBTR),
   CONSTRAINT AVALIACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
```

Observações:

- A Tabela CAO contém os dados dos cães inscritos na referida associação. A coluna NOME_PAI indica o nome do pai de um cão, a coluna RACA indica a raça do mesmo, e a coluna NOME_PROPR indica o nome do seu proprietário. As demais colunas são autoexplicativas.
- A Tabela COMPETICAO contém informações sobre as competições patrocinadas pela associação.

Suas colunas são autoexplicativas.

- A Tabela PARTICIPACAO informa as competições das quais participaram os cães registrados na associação. Cada linha dessa tabela indica a colocação obtida por um cão em uma determinada competição. Suas colunas são autoexplicativas.
- A Tabela ARBITRO contém os dados dos árbitros que julgam os cães que participam de competições. Suas colunas são autoexplicativas.
- Cada linha da Tabela AVALIACAO representa a nota atribuída a um cão, por um determinado árbitro em uma determinada competição. Suas colunas são autoexplicativas.

As Figuras a seguir exibem os dados presentes, em determinado instante, nas Tabelas CAO, COMPETICAO e PARTICIPACAO.

TABELA CAO

COD	NOME	RACA	NOME_PAI	NOME_PROPR
1111	GINGER	SETTER		LUANA ALBUQUERQUE
3333	BIBA	POINTER	RALPH	JOAO MARTINS
6666	BETINA	SETTER	BETINO	TELMA AGUIAR
4444	QUIM	PASTOR ALEMAO		MARIA IDALINA
5555	TAINA	PASTOR ALEMAO		PEDRO ALMEIDA
7777	JANIS	COCKER	TED	ANA PAULA PINTO
2222	JUDY	POINTER		JOSE MARTINS
8888	VIVI	SHITZU		FERNANDA MATHIAS

TABELA COMPETICAO

COD	DESCR
2222	TERESÓPOLIS OPEN
3333	TORNEIO DE FRIBURGO
1111	ABERTO DO RIO

TABELA PARTICIPACAO

COD_CAO	COD_COMP	COLOCACAO
4444	1111	4
8888	2222	1
3333	2222	2
2222	2222	3
1111	1111	1
2222	1111	2
3333	1111	3
5555	1111	5
1111	2222	5
4444	2222	3

Qual comando irá inserir uma nova linha no banco de dados em questão?

a) INSERT INTO COMPETICAO VALUES ('MOSTRA COMPETITIVA DE BRASILIA',9595)

b) INSERT INTO PARTICIPACAO VALUES(8877,1111,6)

c) INSERT INTO PARTICIPACAO (COLOCACAO, COD_COMP, COD_CAO) VALUES (1,1111,8888)

d) INSERT INTO CAO (COD, NOME, RACA, NOME_PAI) VALUES (1130,'TUTU','BULDOG FRANCES','PEPEU')

e) INSERT INTO PARTICIPACAO VALUES (2222,1111,7)

30. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Exploração de Petróleo Júnior - Informática

ALUNO (cpf : string , nome : string , endereco : string, telefone : string) MATRICULA (cpf : string , cod-cad : string)

CADEIRA (cod-cad : string , nome : string , creditos : number) Que comando SQL cria tabela ALUNO?

a) CREATE TABLE ALUNO (cpf string , nome string , endereco string , telefone string) ;

b) CREATE TABLE ALUNO (cpf : string , nome : string , endereco : string , telefone : string) ;

c) CREATE TABLE ALUNO (cpf string PK, nome string , endereco string , telefone string) ;

d) CREATE ALUNO AS TABLE (cpf : string PK, nome : string , endereco : string , telefone : string) ;

e) CREATE ALUNO AS TABLE (cpf string PK , nome string , endereco string , telefone string) ;

31. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Exploração de Petróleo Júnior - Informática

ALUNO (cpf : string , nome : string , endereco : string, telefone : string) MATRICULA (cpf : string , cod-cad : string)

CADEIRA (cod-cad : string , nome : string , creditos : number)

Qual o comando SQL para alterar o nome do aluno com CPF 512.859.850-01 para “Jose da Silva”?

a) ALTER RECORD ALUNO SET nome='Jose da Silva' WHERE cpf='512.859.850-01'

b) INSERT INTO ALUNO nome='Jose da Silva' AND cpf='512.859.850-01'

c) UPDATE ALUNO WHERE nome='Jose da Silva' AND cpf='512.859.850-01'

d) UPDATE ALUNO SET nome='Jose da Silva' WHERE cpf='512.859.850-01'

e) INSERT INTO ALUNO nome='Jose da Silva' WHERE cpf='512.859.850-01'

32. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Exploração de Petróleo Júnior - Informática

ALUNO (cpf : string , nome : string , endereco : string, telefone : string) MATRICULA (cpf : string , cod-cad : string)

CADEIRA (cod-cad : string , nome : string , creditos : number)

Qual o comando SQL que obtém apenas os nomes de todos os alunos?

a) SELECT * FROM ALUNO WHERE nome IS STRING
b) SELECT nome FROM ALUNO
c) LIST * FROM ALUNO
d) SELECT nome WHERE ALUNO
e) LIST nome FROM ALUNO

33. Ano: 2014 Banca: CESGRANRIO Órgão: PetrobrasProva: Engenheiro(a) de Equipamentos Júnior

- Eletrônica

Observe a Tabela FERRAMENTAS abaixo, relativa a um banco de dados relacional.

	CODIGO	ITEM	PRECO	DESCRICAO
tupla 1 →	F-1542	alicate de pressao	23,99	medio
tupla 2 →	F-1543	alicate comum	14,11	ponta fina
tupla 3 →	F-1544	alicate de corte	15,70	pequeno
tupla 4 →	F-2376	chave de fenda	5,76	comum
tupla 5 →	F-2378	chave de fenda	8,20	phillips
tupla 6 →	F-2384	chave de fenda	9,00	phillips
tupla 7 →	F-2400	chave de teste	10,20	pequena
tupla 8 →	F-3176	chave de teste	7,40	pequena
tupla 9 →	F-3237	ferro de soldar	11,80	pequeno

A execução do comando SQL SELECT * FROM FERRAMENTAS WHERE ((ITEM = 'chave de fenda' OR ITEM = 'chave de teste') AND PRECO < 9) produzirá como resposta, respectivamente, as tuplas de números

a) 2, 6 e 7

b) 4, 5 e 8

c) 3, 7 e 9

d) 1, 2, 3 e 9

e) 4, 5, 6 e 8

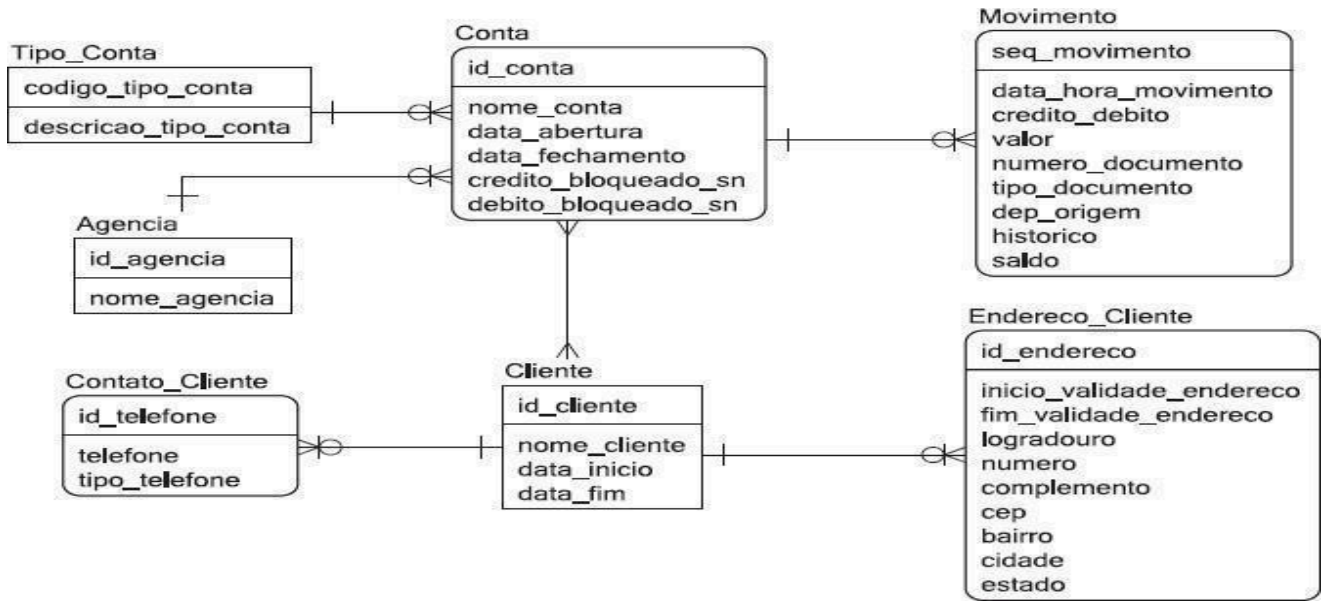
34. Ano: 2014 Banca: CESGRANRIO Órgão: CEFET-RJ Prova: Técnico de Tecnologia da Informação

Que comando SQL insere uma linha na Tabela Aluno, com idAluno=1 e nomeAluno="Aline"?

a) INSERT INTO Aluno SET nomeAluno="Aline" WHERE idAluno=1
b) INSERT INTO Aluno (idAluno, nomeAluno) VALUES (1,"Aline")
c) INSERT INTO Aluno SET nomeAluno="Aline" AND idAluno=1
d) UPDATE Aluno SET nomeAluno="Aline" WHERE idAluno=1
e) UPDATE Aluno(idAluno, nomeAluno) SET VALUES (1,"Aline")

35. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Banco de Dados

Para responder à questão, tenha como referência o diagrama de entidades e relacionamentos, apresentado abaixo, que representa parte do modelo de dados de uma instituição financeira.



Que comando SQL deve ser dado para criar a Tabela Tipo_Conta?

a) CREATE TABLE Tipo_Conta (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

b) CREATE TABLE Tipo_Conta (codigo_tipo_conta:NUMERIC PRIMARY KEY, descricao_tipo_conta:VARCHAR(256))

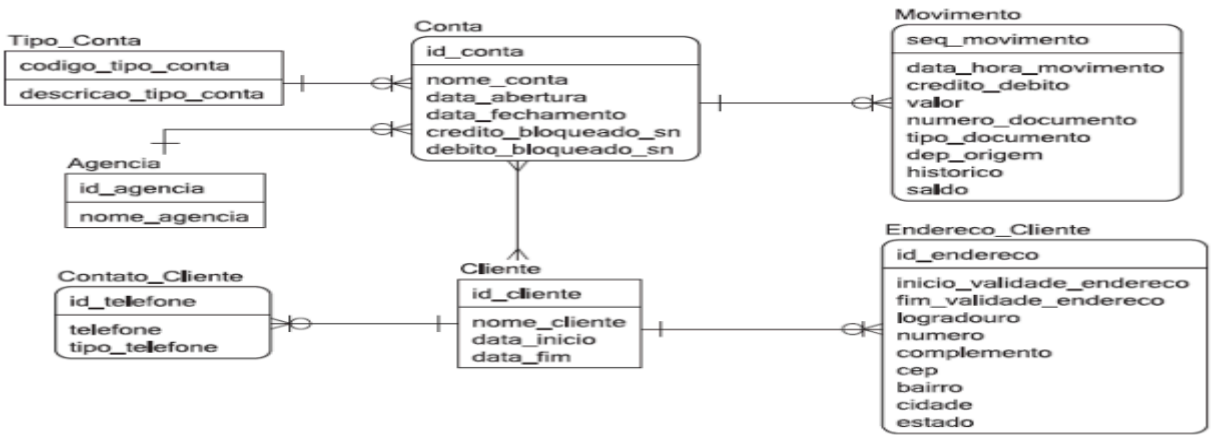
c) CREATE TABLE Tipo_Conta WITH COLUMNS (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

d) CREATE Tipo_Conta AS TABLE (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

e) CREATE Tipo_Conta AS TABLE WITH COLUMNS (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

36. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Banco de Dados

Para responder à questão, tenha como referência o diagrama de entidades e relacionamentos,



apresentado abaixo, que representa parte do modelo de dados de uma instituição financeira.

Que comando SQL deve ser dado para bloquear o crédito da conta 123456, colocando "S" no campo credito_bloqueado_sn?

a) UPDATE Conta SET credito_bloqueado_sn="S" SELECT * FROM Conta WHERE id_conta=123456

b) UPDATE Conta SET credito_bloqueado_sn="S" WHERE id_conta=123456

c) UPDATE Conta SET VALUES (id_conta,"S") WHERE id_conta=123456

d) UPDATE credito_bloqueado_sn="S" From Conta WHERE id_conta=123456

e) UPDATE INTO Conta VALUES (123456,"S")

37. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Banco de Dados

Considere um banco de dados relacional com as duas tabelas a seguir. Empregado

(emp_id, emp_nome, dno, salario)

Departamento (dep_id, dep_mome)

O campo Empregado.dno indica o dep_id do departamento onde o empregado trabalha, e os campos sublinhados são chave primária.

Nesse contexto, analise o seguinte comando SQL:

```
SELECT  d.dep_nome, COUNT(*) AS x
FROM    Departamento d, Empregado e
WHERE  d.dep_id = e.dno AND e.salario > 5000 AND
e.dno IN (SELECT f.dno FROM  Empregado f GROUP BY
f.dno HAVING COUNT(*) > 2)
GROUP BY  d.dep_nome;
```

O que calcula o comando SQL apresentado acima?

a) Quantos empregados ganham mais de R$ 5.000,00, listados por departamento.

b) Quantos empregados existem, listados por departamento, em departamentos com mais de duas pessoas que ganham mais de R$ 5.000,00.

c) Quantos empregados existem, listados por departamento, em departamentos que possuem duas pessoas que ganham mais de R$ 5.000,00.

d) Quantos empregados ganham mais de R$ 5.000,00, listados por departamento, em departamentos com mais de duas pessoas.

e) Quantos departamentos existem com mais de duas pessoas que ganham R$ 5.000,00.

38. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Analise de Sistemas

As Tabelas pertencem ao esquema de um banco de dados acadêmico de uma universidade.

```
CREATE TABLE ALUNO (
    MATRIC       NUMBER(5)      NOT NULL,
    NOME         VARCHAR2(30)   NOT NULL,
    CPF          NUMBER(11),
    CONSTRAINT   ALUNO_UK1 UNIQUE (CPF),
    CONSTRAINT   ALUNO_PK PRIMARY KEY (MATRIC))

CREATE TABLE HISTORICO (
    MATRIC      NUMBER(5)     NOT NULL,
    COD_DISC    CHAR(7)       NOT NULL,
    NOME_DISC   VARCHAR2(30)  NOT NULL,
    ANO         NUMBER(4)     NOT NULL,
    SEMESTRE    NUMBER(1)     NOT NULL,
    NOTA        NUMBER(3,1)   NOT NULL,
    CONSTRAINT   HIST_PK
        PRIMARY KEY (MATRIC,COD_DISC,ANO,SEMESTRE),
    CONSTRAINT   HIST_FK  FOREIGN KEY (MATRIC)
        REFERENCES ALUNO (MATRIC))
```

Observações:

- Um aluno pode cursar uma disciplina várias vezes, desde que em diferentes ANO/SEMESTRE.
- Um aluno será reprovado em uma disciplina caso sua nota seja inferior a 5,0.
- O estado corrente do banco de dados dessa universidade é mostrado a seguir.

Tabela ALUNO

MATRIC	NOME	CPF
33333	FERNANDA MARTINS	null
44444	GABRIEL LOPES	null
11111	ANA MARIA	12344456789
22222	FLÁVIA FERNANDES	23565677789
55555	LIVIA LEVY	23576712145

Tabela HISTORICO

MATRIC	COD_DISC	NOME_DISC	ANO	SEMESTRE	NOTA
11111	INF1009	LÓGICA	2012	1	6
33333	INF1009	LÓGICA	2013	1	8
33333	INF1010	ESTR DE DADOS	2012	2	4
33333	INF1010	ESTR DE DADOS	2013	1	4
33333	INF1010	ESTR DE DADOS	2013	2	6
55555	INF1009	LÓGICA	2013	1	4
55555	INF1009	LÓGICA	2013	2	6
55555	INF1010	ESTR DE DADOS	2013	2	8
11111	INF1010	ESTR DE DADOS	2013	2	7
33333	INF1001	PROGRAMAÇÃO II	2012	2	4
55555	INF1001	PROGRAMAÇÃO II	2012	2	6
33333	INF1001	PROGRAMAÇÃO II	2013	1	7

Qual comando irá modificar o estado corrente da Tabela ALUNO?

a) INSERT INTO ALUNO (MATRIC, CPF, NOME) VALUES (66666,'TIAGO MENEZES')

b) DELETE FROM ALUNO WHERE CPF IS NULL

c) UPDATE ALUNO SET CPF=23565677789 WHERE NOME ='GABRIEL LOPES'

d) INSERT INTO ALUNO VALUES (66666,'TIAGO MENEZES')

e) DELETE FROM ALUNO A WHERE NOT EXISTS (SELECT * FROM HISTORICO WHERE MATRIC=A.MATRIC)

39. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Analise de Sistemas

As Tabelas pertencem ao esquema de um banco de dados acadêmico de uma universidade.

```
CREATE TABLE ALUNO (
    MATRIC       NUMBER(5)      NOT NULL,
    NOME         VARCHAR2(30)   NOT NULL,
    CPF          NUMBER(11),
    CONSTRAINT   ALUNO_UK1 UNIQUE (CPF),
    CONSTRAINT   ALUNO_PK PRIMARY KEY (MATRIC))

CREATE TABLE HISTORICO (
    MATRIC      NUMBER(5)     NOT NULL,
    COD_DISC    CHAR(7)       NOT NULL,
    NOME_DISC   VARCHAR2(30)  NOT NULL,
    ANO         NUMBER(4)     NOT NULL,
    SEMESTRE    NUMBER(1)     NOT NULL,
    NOTA        NUMBER(3,1)   NOT NULL,
    CONSTRAINT  HIST_PK
        PRIMARY KEY (MATRIC,COD_DISC,ANO,SEMESTRE),
    CONSTRAINT  HIST_FK  FOREIGN KEY (MATRIC)
        REFERENCES ALUNO (MATRIC))
```

Observações:

- Um aluno pode cursar uma disciplina várias vezes, desde que em diferentes ANO/SEMESTRE.
- Um aluno será reprovado em uma disciplina caso sua nota seja inferior a 5,0.
- O estado corrente do banco de dados dessa universidade é mostrado a seguir.

Tabela ALUNO

MATRIC	NOME	CPF
33333	FERNANDA MARTINS	null
44444	GABRIEL LOPES	null
11111	ANA MARIA	12344456789
22222	FLÁVIA FERNANDES	23565677789
55555	LIVIA LEVY	23576712145

Tabela HISTORICO

MATRIC	COD_DISC	NOME_DISC	ANO	SEMESTRE	NOTA
11111	INF1009	LÓGICA	2012	1	6
33333	INF1009	LÓGICA	2013	1	8
33333	INF1010	ESTR DE DADOS	2012	2	4
33333	INF1010	ESTR DE DADOS	2013	1	4
33333	INF1010	ESTR DE DADOS	2013	2	6
55555	INF1009	LÓGICA	2013	1	4
55555	INF1009	LÓGICA	2013	2	6
55555	INF1010	ESTR DE DADOS	2013	2	8
11111	INF1010	ESTR DE DADOS	2013	2	7
33333	INF1001	PROGRAMAÇÃO II	2012	2	4
55555	INF1001	PROGRAMAÇÃO II	2012	2	6
33333	INF1001	PROGRAMAÇÃO II	2013	1	7

A execução de uma consulta SQL sobre o banco de dados dessa universidade produziu o seguinte resultado:

NOME	AVG(NOTA)
LIVIA LEVY	6
FLÁVIA FERNANDES	null
ANA MARIA	7
FERNANDA MARTINS	5,5
GABRIEL LOPES	null

Que consulta é essa?

a) SELECT A.NOME, AVG(H.NOTA) FROM ALUNO A
RIGHT OUTER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.NOME

b) SELECT A.NOME, AVG(H.NOTA)
FROM ALUNO A
INNER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.NOME

c) SELECT A.NOME, AVG(H.NOTA) FROM ALUNO A, HISTORICO H WHERE A.MATRIC=H.MATRIC GROUP BY A.NOME

d) SELECT A.NOME, AVG(H.NOTA) FROM ALUNO A
LEFT OUTER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.NOME

e) SELECT A.NOME, AVG(H.NOTA) FROM ALUNO A
RIGHT OUTER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.MATRIC

40. Ano: 2014 Banca: CESGRANRIO Órgão: Banco da Amazônia Prova: Técnico Científico - Analise de Sistemas

As Tabelas pertencem ao esquema de um banco de dados acadêmico de uma universidade.

```
CREATE TABLE ALUNO (
    MATRIC      NUMBER(5)     NOT NULL,
    NOME        VARCHAR2(30)  NOT NULL,
    CPF         NUMBER(11),
    CONSTRAINT  ALUNO_UK1 UNIQUE (CPF),
    CONSTRAINT  ALUNO_PK PRIMARY KEY (MATRIC))

CREATE TABLE HISTORICO (
    MATRIC      NUMBER(5)     NOT NULL,
    COD_DISC    CHAR(7)       NOT NULL,
    NOME_DISC   VARCHAR2(30)  NOT NULL,
    ANO         NUMBER(4)     NOT NULL,
    SEMESTRE    NUMBER(1)     NOT NULL,
    NOTA        NUMBER(3,1)   NOT NULL,
    CONSTRAINT  HIST_PK
        PRIMARY KEY (MATRIC,COD_DISC,ANO,SEMESTRE),
    CONSTRAINT  HIST_FK  FOREIGN KEY (MATRIC)
        REFERENCES ALUNO (MATRIC))
```

Observações:

- Um aluno pode cursar uma disciplina várias vezes, desde que em diferentes ANO/SEMESTRE.
- Um aluno será reprovado em uma disciplina caso sua nota seja inferior a 5,0.
- O estado corrente do banco de dados dessa universidade é mostrado a seguir.

Tabela ALUNO

MATRIC	NOME	CPF
33333	FERNANDA MARTINS	null
44444	GABRIEL LOPES	null
11111	ANA MARIA	12344456789
22222	FLÁVIA FERNANDES	23565677789
55555	LIVIA LEVY	23576712145

Tabela HISTORICO

MATRIC	COD_DISC	NOME_DISC	ANO	SEMESTRE	NOTA
11111	INF1009	LÓGICA	2012	1	6
33333	INF1009	LÓGICA	2013	1	8
33333	INF1010	ESTR DE DADOS	2012	2	4
33333	INF1010	ESTR DE DADOS	2013	1	4
33333	INF1010	ESTR DE DADOS	2013	2	6
55555	INF1009	LÓGICA	2013	1	4
55555	INF1009	LÓGICA	2013	2	6
55555	INF1010	ESTR DE DADOS	2013	2	8
11111	INF1010	ESTR DE DADOS	2013	2	7
33333	INF1001	PROGRAMAÇÃO II	2012	2	4
55555	INF1001	PROGRAMAÇÃO II	2012	2	6
33333	INF1001	PROGRAMAÇÃO II	2013	1	7

Qual consulta exibe os nomes dos alunos que nunca foram reprovados?

a) SELECT DISTINCT A.NOME FROM ALUNO A, HISTORICO H

WHERE A.MATRIC=H.MATRIC AND H.NOTA >= 5.0

b) SELECT NOME FROM ALUNO MINUS

SELECT DISTINCT A.NOME FROM ALUNO A, HISTORICO H WHERE A.MATRIC=H.MATRIC GROUP BY A.NOME, H.NOTA HAVING H.NOTA < 5.0

c) SELECT DISTINCT A.NOME FROM ALUNO A, HISTORICO H WHERE A.MATRIC=H.MATRIC GROUP BY A.NOME, H.NOTA HAVING H.NOTA >=5.0

d) SELECT A.NOME

FROM ALUNO A WHERE A. MATRIC IN

(SELECT MATRIC FROM HISTORICO WHERE NOTA >= 5.0)

e) SELECT DISTINCT A.NOME FROM ALUNO A

LEFT OUTER JOIN HISTORICO H ON A.MATRIC=H.MATRIC GROUP BY A.NOME, H.NOTA HAVING H.NOTA >=5.0

41. Ano: 2014 Banca: CESGRANRIO Órgão: EPE Prova: Analista de Gestão Corporativa - Tecnologia da Informação

As informações presentes abaixo devem ser usadas para responder às questões de nos 50 a 52.

Pessoa			
Id	Nome	Idade	Sexo
1	Ana	70	F
2	Beto	65	M
3	Carlos	45	M
4	Débora	40	F
5	Ênio	22	M
6	Fabiana	20	F
7	Guto	1	M
8	Hilda	52	F

Parentesco	
PaiMae	FilhoFilha
1	3
1	4
2	3
2	4
5	7
6	7
3	5
4	5

Que comando SQL inclui a informação de que Hilda é mãe de Fabiana?

a) INSERT INTO Parentesco SELECT F.Id,P.Id FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Hilda' AND F.Nome='Fabiana '

b) INSERT INTO Parentesco SELECT P.Id,F.Id FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Hilda' AND F.Nome='Fabiana'

c) INSERT INTO Parentesco SELECT P.Id,F.Id FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Fabiana' AND F.Nome='Hilda'

d) INSERT INTO Parentesco VALUES SELECT F.Id,P.FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Hilda' AND F.Nome='Fabiana '

e) INSERT INTO Parentesco VALUES SELECT P.Id,F.Id FROM Pessoa AS P , Pessoa AS F WHERE P.Nome='Hilda' AND F.Nome='Fabiana'

42. Ano: 2014 Banca: CESGRANRIO Órgão: EPE Prova: Analista de Gestão Corporativa - Tecnologia da Informação

As informações presentes abaixo devem ser usadas para responder às questões de nos 50 a 52.

Pessoa			
Id	Nome	Idade	Sexo
1	Ana	70	F
2	Beto	65	M
3	Carlos	45	M
4	Débora	40	F
5	Ênio	22	M
6	Fabiana	20	F
7	Guto	1	M
8	Hilda	52	F

Parentesco	
PaiMae	FilhoFilha
1	3
1	4
2	3
2	4
5	7
6	7
3	5
4	5

Que comando SQL NÃO fornecerá apenas o nome de todos os filhos de Ana?

a) SELECT F.Nome AS FF FROM (Pessoa AS P INNER JOIN Parentesco ON P.Id=PaiMae) INNER JOIN Pessoa AS F ON F.Id=FilhoFilha WHERE P.Nome='Ana'

b) SELECT F.Nome FROM Pessoa AS F WHERE F.Id IN (SELECT FilhoFilha FROM Pessoa AS P, Parentesco AS PP WHERE P.Id=PP.PaiMae AND P.Nome='Ana')

c) SELECT F.Nome FROM Pessoa AS F WHERE F.Id IN (SELECT FilhoFilha FROM Pessoa AS P INNER JOIN Parentesco AS PP ON P.Id=PP.PaiMae WHERE P.Nome='Ana')

d) SELECT F.Nome FROM Pessoa AS F WHERE F.Id IN (SELECT FilhoFilha FROM Pessoa AS P LEFT JOIN Parentesco AS PP ON P.Id=PP.PaiMae WHERE P.Nome='Ana')

e) SELECT F.Nome FROM Pessoa AS P, Pessoa AS F, Parentesco AS R WHERE P.Nome='Ana' AND F.Id=R.PaiMae AND P.Id=R.FilhoFilha

43. Ano: 2014 Banca: CESGRANRIO Órgão: FINEP Prova: Analista - Desenvolvimento de Sistemas

As tabelas abaixo pertencem ao esquema de um banco de dados de um supermercado. CREATE TABLE PRODUTO (

COD NUMBER(5) NOT NULL, DESCRICAO VARCHAR2(100) NOT NULL, PRECO NUMBER(8,2) NOT NULL, QTD_ESTOQUE NUMBER(5) ,

TIPO NUMBER(1) NOT NULL,

CONSTRAINT PRODUTO_PK PRIMARY KEY (COD)) CREATE TABLE ITEM (

NUM_SERIE NUMBER(7) NOT NULL, COR VARCHAR2(20) NOT NULL, VOLTAGEM NUMBER(5) NOT NULL, COD_PROD NUMBER(5) NOT NULL,

CONSTRAINT ITEM_PK PRIMARY KEY (NUM_SERIE), CONSTRAINT ITEM_FK FOREIGN KEY (COD_PROD)
REFERENCES PRODUTO (COD))

Observações:

- A empresa comercializa produtos controlados por quantidade (por exemplo, caixa de sabão em pó, pacote de biscoito e lata de extrato de tomate) e produtos controlados por unidade (por exemplo, televisor, máquina de lavar roupa e liquidificador).
- A quantidade em estoque de um produto controlado por quantidade (TIPO=1) é obtida diretamente da coluna QTD_ESTOQUE.
- A quantidade em estoque de um produto controlado por unidade (TIPO=2) NÃO pode ser obtida diretamente da coluna QTD_ESTOQUE, pois, para esse tipo de produto, esta coluna irá conter o valor NULL.
- Cada linha da tabela ITEM contém informações sobre um item existente no estoque da empresa relativo a um tipo de produto controlado por unidade.

Qual consulta SQL irá exibir o código, a descrição e a quantidade em estoque relativos a cada um dos produtos comercializados pelo supermercado?

a) SELECT COD, DESCRICAO, QTD_ESTOQUE FROM PRODUTO
WHERE TIPO = 1 UNION
SELECT P.COD, P.DESCRICAO, COUNT(I.COD_PROD) FROM PRODUTO P,ITEM I
WHERE TIPO = 2 AND P.COD=I.COD_PROD GROUP BY P.COD, P.DESCRICAO

b) SELECT COD, DESCRICAO, QTD_ESTOQUE FROM PRODUTO
WHERE TIPO = 1 UNION
SELECT P.COD, P.DESCRICAO, COUNT(I.COD_PROD) FROM PRODUTO P
LEFT JOIN ITEM I
ON P.COD=I.COD_PROD WHERE P.TIPO = 2
GROUP BY P.COD, P.DESCRICAO

c) SELECT P.COD, P.DESCRICAO, COUNT(DISTINCT P.TIPO) FROM PRODUTO P
LEFT OUTER JOIN ITEM I ON P.COD=I.COD_PROD
GROUP BY P.COD, P.DESCRICAO

d) SELECT P.COD, P.DESCRICAO, SUM (DISTINCT P.TIPO) FROM PRODUTO P
INNER JOIN ITEM I

ON P.COD=I.COD_PROD GROUP BY P.COD, P.DESCRICAO

e) SELECT COD, DESCRICAO, QTD_ESTOQUE FROM PRODUTO
WHERE TIPO = 1

UNION
SELECT P.COD, P.DESCRICAO, COUNT(I.COD_PROD) FROM PRODUTO P

RIGHT JOIN ITEM I

ON P.COD=I.COD_PROD WHERE P.TIPO = 2

GROUP BY P.COD,P.DESCRICAO

44. Ano: 2014 Banca: CESGRANRIO Órgão: FINEP Prova: Analista - Desenvolvimento de Sistemas

As tabelas abaixo pertencem ao esquema de um banco de dados de um supermercado. CREATE TABLE PRODUTO (

COD NUMBER(5) NOT NULL,

DESCRICAO VARCHAR2(100) NOT NULL,

PRECO NUMBER(8,2) NOT NULL, QTD_ESTOQUE NUMBER(5) , TIPO NUMBER(1) NOT NULL,

CONSTRAINT PRODUTO_PK PRIMARY KEY (COD))

CREATE TABLE ITEM (

NUM_SERIE NUMBER(7) NOT NULL, COR VARCHAR2(20) NOT NULL, VOLTAGEM NUMBER(5) NOT NULL, COD_PROD NUMBER(5) NOT NULL,

CONSTRAINT ITEM_PK PRIMARY KEY (NUM_SERIE), CONSTRAINT ITEM_FK FOREIGN KEY (COD_PROD)

REFERENCES PRODUTO (COD))

Observações:

- A empresa comercializa produtos controlados por quantidade (por exemplo, caixa de sabão em pó, pacote de biscoito e lata de extrato de tomate) e produtos controlados por unidade (por exemplo, televisor, máquina de lavar roupa e liquidificador).
- A quantidade em estoque de um produto controlado por quantidade (TIPO=1) é obtida diretamente da coluna QTD_ESTOQUE.
- A quantidade em estoque de um produto controlado por unidade (TIPO=2) NÃO pode ser obtida diretamente da coluna QTD_ESTOQUE, pois, para esse tipo de produto, esta coluna irá conter o valor NULL.
- Cada linha da tabela ITEM contém informações sobre um item existente no estoque da empresa
relativo a um tipo de produto controlado por unidade.

Qual comando SQL irá inserir corretamente uma nova linha na tabela de produtos, além de não violar restrições semânticas relativas ao banco de dados do supermercado?

a) INSERT INTO PRODUTO (COD, DESCRICAO, PRECO, TIPO)

VALUES (7777, 'COMPUTADOR BLUEX', 1000.00, 2)

b) INSERT INTO PRODUTO VALUES (7777,'COMPUTADOR BLUEX',1000.00,2)

c) INSERT INTO PRODUTO VALUES (8888,'SARDINHA EM LATA BOM PEIXE',2.50,700,2)

d) INSERT INTO PRODUTO (COD,DESCRICAO,PRECO,QTD_ESTOQUE) VALUES(7777,'COMPUTADOR BLUEX',1000.00, ,2)

e) INSERT INTO PRODUTO (COD,DESCRICAO,PRECO,TIPO,QTD_ESTOQUE) VALUES(7777,'COMPUTADOR BLUEX',1000.00,2)

45. Ano: 2014 Banca: CESGRANRIO Órgão: FINEP Prova: Analista - Desenvolvimento de Sistemas Para responder essa questão use a mesma referência da questão anterior.

O analista de suporte de banco de dados do supermercado solicitou que a coluna QTD_ESTOQUE passasse a conter a quantidade de itens em estoque de produtos do tipo 2. Embora ele reconheça que isso resultará em redundância, os relatórios de performance mostram que existe um desperdício de recursos computacionais significativo com o cálculo recorrente do total de itens em estoque de produtos do tipo 2.

Qual comando SQL irá atualizar corretamente a coluna QTD_ESTOQUE com a quantidade de itens em estoque relativa a cada um dos produtos do tipo 2 comercializados pelo supermercado?

a) UPDATE PRODUTO P SET QTD_ESTOQUE = (SELECT COUNT(*) FROM ITEM WHERE COD_PROD=P.COD)

b) UPDATE PRODUTO P SET QTD_ESTOQUE = (SELECT COUNT(*) FROM ITEM WHERE COD_PROD=P.COD AND P.TIPO=2)

c) UPDATE PRODUTO P SET QTD_ESTOQUE = (SELECT COUNT(DISTINCT COD_PROD) FROM ITEM WHERE COD_PROD=P.COD AND P.TIPO=2)

d) UPDATE PRODUTO P SET QTD_ESTOQUE=(SELECT COUNT(COD_PROD) FROM ITEM WHERE COD_PROD=P.COD) WHERE TIPO = 2

e) UPDATE PRODUTO P SET QTD_ESTOQUE=(SELECT COUNT(DISTINCT COD_PROD) FROM ITEM WHERE COD_PROD=P.COD) WHERE TIPO=2

46. Ano: 2014 Banca: CESGRANRIO Órgão: FINEP Prova: Analista - Desenvolvimento de Sistemas

Ao implementar um sistema de gerência de fornecedores, o desenvolvedor percebeu que não existia no banco de dados relacional da empresa qualquer representação da entidade PRODUTO que aparecia em seu modelo de dados. Para corrigir essa falha, preparou um comando SQL que alteraria o esquema do banco de dados.

Tal comando SQL deve ser iniciado com

a) ALTER SCHEMA ADD TABLE PRODUTO
b) ALTER TABLE PRODUTO

c) CREATE PRODUTO : TABLE

d) CREATE PRODUTO AS TABLE

e) CREATE TABLE PRODUTO

47. Ano: 2013 Banca: CESGRANRIO Órgão: IBGE Prova: Analista - Suporte Operacional

Em um banco de dados, a tabela Pessoa foi criada com a seguinte instrução: CREATE TABLE Pessoa (PessoaID int,
Nome varchar(255), Sobrenome varchar(255), Endereco varchar(255), Cidade varchar(255)
);

Que instrução SQL acrescenta um campo CEP do tipo varchar(9) a essa tabela?

a) ADD COLUMN CEP varchar(9) INTO TABLE
b) ALTER TABLE Pessoa ADD CEP varchar(9)
c) ALTER TABLE Pessoa INSERT COLUMN CEP varchar(9)
d) ALTER TABLE Pessoa ALTER COLUMN CEP varchar(9)
e) ALTER TABLE Pessoa MODIFY COLUMN ADD CEP varchar(9)

48. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Informática Júnior

```
CREATE TABLE Inquilino(
      nome                  VARCHAR(20) NULL,
      cpf                   CHAR(11) NOT NULL,
      PRIMARY KEY (cpf));

CREATE TABLE Vaga(
      andar                 INTEGER NOT NULL,
      numero                INTEGER NOT NULL,
      PRIMARY KEY (andar,numero));

CREATE TABLE Vaga_Inquilino(
      andar                 INTEGER NOT NULL,
      numero                INTEGER NOT NULL,
      cpf                   CHAR(11) NOT NULL,
      PRIMARY KEY (andar,numero,cpf));
```

Desse modo, para incluir o campo telefone na Tabela Inquilino, o comando necessário é

a) add column telefone varchar(12) on table Inquilino;
b) alter table Inquilino add column telefone varchar(12);
c) alter table Inquilino insert column telefone varchar(12);
d) insert column telefone varchar(12) on table Inquilino;
e) modify table Inquilino add column telefone varchar(12);

49. Ano: 2014 Banca: CESGRANRIO Órgão: Petrobras Prova: Técnico(a) de Informática Júnior

```
CREATE TABLE Inquilino(
      nome                  VARCHAR(20) NULL,
      cpf                   CHAR(11) NOT NULL,
      PRIMARY KEY (cpf));

CREATE TABLE Vaga(
      andar                 INTEGER NOT NULL,
      numero                INTEGER NOT NULL,
      PRIMARY KEY (andar,numero));

CREATE TABLE Vaga_Inquilino(
      andar                 INTEGER NOT NULL,
      numero                INTEGER NOT NULL,
      cpf                   CHAR(11) NOT NULL,
      PRIMARY KEY (andar,numero,cpf));
```

Nessa situação, para se obter um relatório com a quantidade de vagas por cada inquilino, listadas e agrupadas por cpf, deve ser feita a seguinte consulta:

a) select cpf,andar,numero from Vaga_Inquilino;

b) select cpf,count(*) from Vaga_Inquilino group by cpf;

c) select cpf,sum(cpf) from Vaga_Inquilino group by cpf;

d) select distinct cpf from Vaga_Inquilino;

e) select distinct count(Inquilino.cpf) from Inquilino, Vaga_Inquilino Where Inquilino.cpf = Vaga_Inquilino.cpf order by Inquilino.cpf;

50. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação Considere as Tabelas a seguir para responder a questão.

Essas Tabelas fazem parte do esquema de um banco de dados usado por uma associação de criadores de cães para organizar informações sobre os torneios que ela promove.

Observações:

- A Tabela CAO contém os dados dos cães inscritos na referida associação. A coluna NOME_PAI indica o nome do pai de um cão, a coluna RACA indica a raça do mesmo, e a coluna NOME_PROPR indica o nome do seu proprietário. As demais colunas são autoexplicativas.
- A Tabela COMPETICAO contém informações sobre as competições patrocinadas pela associação.
Suas colunas são autoexplicativas.
- A Tabela PARTICIPACAO informa as competições das quais participaram os cães registrados na associação. Cada linha dessa tabela indica a colocação obtida por um cão em uma determinada competição. Suas colunas são autoexplicativas.
- A Tabela ARBITRO contém os dados dos árbitros que julgam os cães que participam de competições. Suas colunas são autoexplicativas.
- Cada linha da Tabela AVALIACAO representa a nota atribuída a um cão, por um determinado árbitro em uma determinada competição. Suas colunas são autoexplicativas.

No contexto do torneio, cuja descrição é ABERTO DO RIO, qual consulta exibe, em ordem

decrescente de somatório de notas, os nomes dos cães participantes e o somatório das notas que cada um recebeu?

a) SELECT C.NOME,SUM(NOTA_ARBTR) FROM CAO C,COMPETICAO P,AVALIACAO N
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=N.COD_COMP AND N.COD_CAO=C.COD
GROUP BY C.NOME
ORDER BY SUM(NOTA_ARBTR) DESC

b) SELECT C.NOME,SUM(NOTA_ARBTR) FROM CAO C,AVALIACAO N
WHERE N.COD_COMP='ABERTO DO RIO' AND AND N.COD_CAO=C.COD GROUP BY C.NOME
ORDER BY SUM(NOTA_ARBTR) DESCENDING

c) SELECT C.NOME,SUM(NOTA_ARBTR) FROM CAO C,COMPETICAO P,AVALIACAO N
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=N.COD_COMP AND N.COD_CAO=C.COD
ORDER BY SUM(NOTA_ARBTR) DESCENDING

d) SELECT C.NOME,SUM(O.COLOCACAO) FROM CAO C,COMPETICAO P,PARTICIPACAO O
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=O.COD_COMP AND O.COD_CAO=C.COD
GROUP BY C.NOME
ORDER BY SUM(O.COLOCACAO)

e) SELECT C.NOME,SUM(N.NOTA_ARBTR) FROM CAO C,COMPETICAO P,AVALIACAO N
WHERE P.DESCR='ABERTO DO RIO' AND P.COD=N.COD_COMP AND N.COD_CAO=C.COD
GROUP BY C.NOME
ORDER BY SUM(N.NOTA_ARBTR)

51. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação

Considere as Tabelas a seguir para responder a questão.

Essas Tabelas fazem parte do esquema de um banco de dados usado por uma associação de criadores de cães para organizar informações sobre os torneios que ela promove.

```
CREATE TABLE CAO (
   COD            NUMBER(5)      NOT NULL,
   NOME           VARCHAR2(50)   NOT NULL,
   RACA           VARCHAR2(50)   NOT NULL,
   NOME_PAI       VARCHAR2(50),
   NOME_PROPR     VARCHAR2(50)   NOT NULL,
   CONSTRAINT CAO_PK PRIMARY KEY (COD)
)
CREATE TABLE COMPETICAO (
   COD            NUMBER(5)      NOT NULL,
   DESCR          VARCHAR2(50)   NOT NULL,
   CONSTRAINT COMPETICAO_PK PRIMARY KEY (COD)
)
CREATE TABLE ARBITRO (
   COD            NUMBER(5)      NOT NULL,
   NOME           VARCHAR2(50)   NOT NULL,
   CONSTRAINT ARBITRO_PK PRIMARY KEY (COD)
)
CREATE TABLE PARTICIPACAO (
   COD_CAO        NUMBER(5)       NOT NULL,
   COD_COMP       NUMBER(5)       NOT NULL,
   COLOCACAO      NUMBER(4)       NOT NULL,
   CONSTRAINT PARTICIPACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP),
   CONSTRAINT PARTICIPACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT PARTICIPACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD)
)
CREATE TABLE AVALIACAO (
   COD_CAO        NUMBER(5)       NOT NULL,
   COD_COMP       NUMBER(5)       NOT NULL,
   COD_ARBTR      NUMBER(5)       NOT NULL,
   NOTA_ARBTR    NUMBER(3,1)      NOT NULL,
   CONSTRAINT AVALIACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP,COD_ARBTR),
   CONSTRAINT AVALIACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT AVALIACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD),
   CONSTRAINT AVALIACAO_FK3 FOREIGN KEY (COD_ARBTR)
      REFERENCES ARBITRO (COD)
)
```

Observações:

- A Tabela CAO contém os dados dos cães inscritos na referida associação. A coluna NOME_PAI indica o nome do pai de um cão, a coluna RACA indica a raça do mesmo, e a coluna NOME_PROPR indica o nome do seu proprietário. As demais colunas são autoexplicativas.
- A Tabela COMPETICAO contém informações sobre as competições patrocinadas pela associação. Suas colunas são autoexplicativas.
- A Tabela PARTICIPACAO informa as competições das quais participaram os cães registrados na associação. Cada linha dessa tabela indica a colocação obtida por um cão em uma determinada competição. Suas colunas são autoexplicativas.
- A Tabela ARBITRO contém os dados dos árbitros que julgam os cães que participam de competições. Suas colunas são autoexplicativas.

- Cada linha da Tabela AVALIACAO representa a nota atribuída a um cão, por um determinado árbitro em uma determinada competição. Suas colunas são autoexplicativas.

Considerando-se o universo de todas as competições promovidas pela associação de criadores de cães, qual consulta exibe o nome do árbitro, cujo código é 1111, e a média das notas que ele atribuiu ao cão chamado GINGER?

a) SELECT A.NOME, AVG(N.NOTA_ARBTR) FROM CAO C,AVALIACAO N,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111
GROUP BY A.COD

b) SELECT A.NOME, AVG(N.NOTA_ARBTR)
FROM CAO C,AVALIACAO N,PARTICIPACAO P,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=P.COD_CAO AND P.COD_CAO=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111
GROUP BY N.NOTA_ARBTR

c) SELECT A.NOME, SUM(N.NOTA_ARBTR) / COUNT(*) FROM CAO C,AVALIACAO N,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111

d) SELECT A.NOME, AVG(N.NOTA_ARBTR)
FROM CAO C,AVALIACAO N,PARTICIPACAO P,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=P.COD_CAO AND P.COD_CAO=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111

e) SELECT A.NOME, SUM(N.NOTA_ARBTR) / COUNT(*) FROM CAO C,AVALIACAO N,ARBITRO A
WHERE C.NOME='GINGER' AND C.COD=N.COD_CAO AND N.COD_ARBTR=A.COD AND A.COD=1111
GROUP BY A.NOME

52. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação

Considere as Tabelas a seguir para responder a questão.

Essas Tabelas fazem parte do esquema de um banco de dados usado por uma associação de criadores de cães para organizar informações sobre os torneios que ela promove.

```
CREATE TABLE CAO (
   COD            NUMBER(5)      NOT NULL,
   NOME           VARCHAR2(50)   NOT NULL,
   RACA           VARCHAR2(50)   NOT NULL,
   NOME_PAI       VARCHAR2(50),
   NOME_PROPR     VARCHAR2(50)   NOT NULL,
   CONSTRAINT CAO_PK PRIMARY KEY (COD)
)
CREATE TABLE COMPETICAO (
   COD            NUMBER(5)      NOT NULL,
   DESCR          VARCHAR2(50)   NOT NULL,
   CONSTRAINT COMPETICAO_PK PRIMARY KEY (COD)
)
CREATE TABLE ARBITRO (
   COD            NUMBER(5)      NOT NULL,
   NOME           VARCHAR2(50)   NOT NULL,
   CONSTRAINT ARBITRO_PK PRIMARY KEY (COD)
)
CREATE TABLE PARTICIPACAO (
   COD_CAO        NUMBER(5)       NOT NULL,
   COD_COMP       NUMBER(5)       NOT NULL,
   COLOCACAO      NUMBER(4)       NOT NULL,
   CONSTRAINT PARTICIPACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP),
   CONSTRAINT PARTICIPACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT PARTICIPACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD)
)
CREATE TABLE AVALIACAO (
   COD_CAO        NUMBER(5)       NOT NULL,
   COD_COMP       NUMBER(5)       NOT NULL,
   COD_ARBTR      NUMBER(5)       NOT NULL,
   NOTA_ARBTR    NUMBER(3,1)      NOT NULL,
   CONSTRAINT AVALIACAO_PK PRIMARY KEY
      (COD_CAO,COD_COMP,COD_ARBTR),
   CONSTRAINT AVALIACAO_FK1 FOREIGN KEY (COD_CAO)
      REFERENCES CAO (COD),
   CONSTRAINT AVALIACAO_FK2 FOREIGN KEY (COD_COMP)
      REFERENCES COMPETICAO (COD),
   CONSTRAINT AVALIACAO_FK3 FOREIGN KEY (COD_ARBTR)
      REFERENCES ARBITRO (COD)
)
```

Observações:

- A Tabela CAO contém os dados dos cães inscritos na referida associação. A coluna NOME_PAI indica o nome do pai de um cão, a coluna RACA indica a raça do mesmo, e a coluna NOME_PROPR indica o nome do seu proprietário. As demais colunas são autoexplicativas.
- A Tabela COMPETICAO contém informações sobre as competições patrocinadas pela associação. Suas colunas são autoexplicativas.
- A Tabela PARTICIPACAO informa as competições das quais participaram os cães registrados na associação. Cada linha dessa tabela indica a colocação obtida por um cão em uma determinada competição. Suas colunas são autoexplicativas.
- A Tabela ARBITRO contém os dados dos árbitros que julgam os cães que participam de competições. Suas colunas são autoexplicativas.
- Cada linha da Tabela AVALIACAO representa a nota atribuída a um cão, por um

determinado árbitro em uma determinada competição. Suas colunas são autoexplicativas.

Qual consulta exibe os nomes dos cães que participaram de, pelo menos, uma competição?

a) SELECT C.NOME FROM CAO C MINUS
SELECT DISTINCT C.NOME FROM CAO C,PARTICIPACAO P WHERE C.COD=P.COD_CAO
GROUP BY C.NOME
HAVING COUNT(*) > 0

b) SELECT C.NOME FROM CAO C
WHERE C.COD NOT IN (SELECT P.COD_CAO FROM PARTICIPACAO P WHERE
P.COD_CAO=C.COD)

c) SELECT C.NOME FROM CAO C
WHERE C.COD IN (SELECT COUNT(*) FROM PARTICIPACAO P WHERE P.COD_CAO=C.COD)

d) SELECT C.NOME
FROM CAO C, PARTICIPACAO P WHERE C.COD=P.COD_CAO GROUP BY C.NOME
HAVING COUNT(*) > 0

e) SELECT C.NOME
FROM CAO C,PARTICIPACAO P
WHERE C.COD=P.COD_CAO AND COUNT(*) > 0 GROUP BY C.NOME

53. Ano: 2016 Banca: CESGRANRIO Órgão: UNIRIO Prova: Técnico em Tecnologia da Informação

O administrador de um banco de dados deseja remover do usuário RH5678 o privilégio de excluir linhas da tabela RH05_FUNCIONARIO.

Qual comando SQL executará o que esse administrador deseja?

a) REVOKE DELETE ON RH05_FUNCIONARIO FROM RH5678
b) PURGE DELETE FROM RH5678 ON RH05_FUNCIONARIO
c) DROP DELETE ON RH05_FUNCIONARIO FROM USER RH5678
d) DROP FUNCTION DELETE ON RH05_FUNCIONARIO FROM RH5678
e) DELETE FUNCTION DELETE FROM RH5678 ON RH05_FUNCIONARIO

GABARITO

1.	2.	3.	4.	5.	6.
D	A	B	B	D	C
7.	8.	9.	10.	11.	12.
A	D	A	E	E	B
13.	14.	15.	16.	17.	18.
C	C (Anulada)	B	A	D	C
19.	20.	21.	22.	23.	24.
E	A	D	A	C	D
25.	26.	27.	28.	29.	30.
C	A	B	E	C	C
31.	32.	33.	34.	35.	36.
D	B	B	B	A	B
37.	38.	39.	40.	41.	42.
D	E	D	B	B	E
43.	44.	45.	46.	47.	48.
B	A	D	E	B	B
49.	50.	51.	52.	53.	
B	A	E	D	A	

Exercícios FGV

1. **FGV - FTE (SEFAZ MT)/SEFAZ MT/2023 -TI - Banco de Dados - Sublinguagens SQL (DDL, DML, DQL, DCL e DTL)**

No contexto da concessão de privilégios/permissões na administração de bancos de dados relacionais, assinale a opção que indica dois dos comandos básicos disponíveis na maior parte dos SGDB.

a) ADD e DELETE.

b) ALLOW e CONSTAINT.

c) ATTACH e DETACH.

d) GRANT e REVOKE.

e) INSERT e REMOVE.

2. **FGV - AJ (TJ RN)/TJ RN/Apoio Especializado/Análise de Sistemas/2023 - TI - Banco de Dados - Conceitos e Fundamentos de Modelo Relacional**

Supondo que no Brasil

- todo brasileiro tem um, e somente um, CPF;

- alguns brasileiros têm um, mas somente um, passaporte válido;

- alguns brasileiros têm uma, mas somente uma, carteira de motorista (CNH) válida.

A definição **correta** desses atributos numa tabela relacional normalizada seria:

a) CPF varchar(11) not null

CNH varchar(9) UNIQUE

Passaporte varchar(9) UNIQUE

b) CPF varchar(11) not null UNIQUE

CNH varchar(9) UNIQUE

Passaporte varchar(9) UNIQUE

c) CPF varchar(11) UNIQUE

CNH varchar(9) not null

Passaporte varchar(9) not null

d) CPF varchar(11) not null

CNH varchar(9) null

Passaporte varchar(9) null

e) CPF varchar(11) null

CNH varchar(9) UNIQUE

Passaporte varchar(9) UNIQUE

3. **FGV - Ana (PGM Niterói)/Pref Niterói/Tecnologia da Informação/2023- TI - Banco de Dados - Álgebra Relacional**

Considere o comando SQL a seguir.

SELECT a.X, b.Y FROM T1 a, T2 b WHERE a.R = b.S

Dado que essa consulta pode ser expressa usando as operações primitivas da Álgebra Relacional, a lista que contém as operações necessárias e suficientes para essa conversão é, em ordem alfabética:

a) Diferença, Produto, Projeção;

b) Produto, Projeção, União;

c) Projeção, União;

d) Produto, Projeção, Seleção;

e) Seleção, União.

4. **FGV - ACE (TCE ES)/TCE ES/Tecnologia da Informação/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Junior é o administrador do Banco de Dados da sua empresa e percebeu que um programador tinha acesso de alteração indevido a uma tabela.

Para cancelar a permissão previamente concedida ao programador, Junior deve usar o comando SQL:

a) GRANT;

b) REVOKE;

c) UPDATE;

d) DELETE;

e) TRUNCATE.

5. **FGV - ATRFB/SRFB/Geral/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Os principais Sistemas Gerenciadores de Bancos de Dados oferecem total suporte à linguagem SQL. Um aspecto importante da implementação do SQL é o tratamento para valores nulos quando esses são considerados como *unknown values*.

Nesse contexto, considere uma tabela T com colunas A e B, que podem conter valores nulos. T possui 100 registros e, em 50% das linhas, há pelo menos uma coluna preenchida com o valor NULL.

Considere a consulta a seguir:

```
SELECT * FROM T t1
WHERE t1.A = NULL or t1.B = NULL
```

O número máximo de linhas de resultados que seriam retornadas pela consulta é igual a

a) 0.

b) 25.

c) 50.

d) 75.

e) 100.

6. **FGV - ATRFB/SRFB/Geral/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Num banco de dados relacional, considere uma tabela R, com duas colunas A e B, ambas do tipo *string* de caracteres, cuja instância é exibida a seguir.

A	B
Pedro	João
Maria	Ida
Maria	Ida
Pedro	João
Edson	Wilson
Edson	Maria

Nesse cenário analise os comandos a seguir.

I.

```
DELETE FROM R
WHERE EXISTS (SELECT * FROM R r1
              WHERE R.A = r1.A and R.B = r1.B)
```

II.

```
DELETE FROM R
WHERE EXISTS (SELECT * FROM R r1
              WHERE R.A + R.B > r1.A + r1.B)
```

III.

```
DELETE FROM R
WHERE R.A + R.B in (SELECT A + B FROM R)
```

Assinale a lista que contém o número de registros deletados em cada um dos comandos **I**, **II** e **III**, respectivamente, quando executados separadamente e usando a mesma instância inicial descrita.

a) 2, 2 e 0.

b) 2, 4 e 0.

c) 4, 4 e 4.

d) 6, 5 e 6.

e) 6, 6 e 6.

7. **FGV - AFRFB/SRFB/Geral/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Num banco de dados relacional, considere a tabela Vencedores, cuja instância é exibida a seguir, com duas colunas, Tenista e Torneio, que representam alguns torneios que já foram vencidos por alguns tenistas.

Tenista	Torneio
Roger Federer	Australian Open
Roger Federer	Roland Garros
Roger Federer	Wimbledon
Roger Federer	US Open
Pete Sampras	US Open
Pete Sampras	Wimbledon
Pete Sampras	Australian Open
Bjorn Borg	Roland Garros
Bjorn Borg	Wimbledon

Maria precisa escrever um comando SQL que liste os tenistas que venceram todos os torneios mencionados na coluna Torneio. O comando deve valer para qualquer instância válida da tabela, que pode conter diferentes tenistas e diferentes torneios.

Assinale o comando que Maria deve usar.

a) select distinct Tenista from Vencedores v1

where v1.Torneio in (select Torneio from Vencedores)

b) select distinct Tenista from Vencedores v1

where exists(

select * from Vencedores v2

where v1.Torneio = v1.Torneio

and v1.Tenista = v2.Tenista

and v1 <> v2))

c) select distinct Tenista from Vencedores v1

where exists (

select * from Vencedores v2

where v1.Torneio = v1.Torneio

and v1.Tenista <> v2.Tenista)

d) select distinct Tenista from Vencedores v1

where for all (

select * from Vencedores v2

where exists (

select * from Vencedores v3

where v1.Tenista = v2.Tenista))

e) select distinct Tenista from Vencedores v1

where not exists(

select * from Vencedores v2

where not exists (

select * from Vencedores v3

where v2.Torneio = v3.Torneio

and v1.Tenista = v3.Tenista))

8. **FGV - AFRFB/SRFB/Geral/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Os principais Sistemas Gerenciadores de Bancos de Dados oferecem total suporte à linguagem SQL; um aspecto importante da implementação do SQL é o tratamento para valores nulos, quando a lógica admite três estados.

T – *true*

F – *false*

? – *unknown*

Nesse contexto, considere as expressões lógicas a seguir.

I. (T OR F) AND (? OR T)

II. T AND ((? OR F) OR ?)

III. NOT (? AND (? AND ?))

Com relação às expressões acima, está correto afirmar que o valor final é *unknown* (?) em

a) **I**, apenas.

b) **I** e **II**, apenas.

c) **I** e **III**, apenas.

d) **II** e **III**, apenas.

e) **I**, **II** e **III**.

9. **FGV - Ana (PGM Niterói)/Pref Niterói/Tecnologia da Informação/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Figura 1 (a figura na prova é a composição do texto e da tabela)

Nas duas questões a seguir, considere a tabela relacional T cuja instância é exibida abaixo.

A instalação está configurada para o tratamento de valores NULL como valores desconhecidos *(unknown*).

Tabela T

A	B	C
3	6	10
1	NULL	10
3	6	NULL
3	6	5

Considerando a tabela T da figura 1, analise a execução do comando SQL a seguir.

```
SELECT *
FROM T t1 LEFT JOIN T t2
        ON t1.A = t2.A and t1.B = t2.B
            and t1.C = t2.C
```

Além da linha de títulos, o número de linhas produzidas pela execução desse comando é:

a) 0;

b) 4;

c) 10;

d) 13;

e) 16.

10. **FGV - Ana (PGM Niterói)/Pref Niterói/Tecnologia da Informação/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Figura 1

Nas duas questões a seguir, considere a tabela relacional T cuja instância é exibida abaixo.

A instalação está configurada para o tratamento de valores NULL como valores desconhecidos *(unknown)*.

Tabela T

A	B	C
3	6	10
1	NULL	10
3	6	NULL
3	6	5

Considerando a tabela T da figura 1, analise o comando a seguir.

```
DELETE FROM T
WHERE EXISTS ( SELECT * FROM T t2
                    WHERE T.A = t2.A
                    and T.B = t2.B
                    and T.C = t2.C )
```

O número de linhas removidas pela execução do comando acima é:

a) 0;

b) 1;

c) 2;

d) 3;

e) 4.

11. **FGV - FTE (SEFAZ MT)/SEFAZ MT/2023- TI - Banco de Dados - Consultas e Comandos em SQL**

No contexto das linguagens de manipulação de dados de SGBD relacionais, analise a instância da tabela T e o comando SQL a seguir.

pessoa	ancestral

Bruna	Joana
Joana	João
João	Maria
Maria	Gabriel
Paulo	Gabriel

insert into T

select t1.pessoa, t2.ancestral

from T t1, T t2

where t1.ancestral = t2.pessoa

and not exists

(select * from T tt

where tt.pessoa = t1.pessoa

and tt.ancestral = t2.ancestral)

Dado que o comando SQL acima foi executado por três vezes consecutivas, assinale o número de linhas inseridas na tabela **T** em cada execução, na ordem.

a) 0, 0, 0.

b) 3, 5, 0.

c) 5, 2, 1.

d) 5, 3, 0.

e) 8, 0, 0.

12. FGV - AJ (TJ RN)/TJ RN/Apoio Especializado/Análise de Sistemas/2023 - TI - Banco de Dados - Consultas e Comandos em SQL

Considere a estrutura e uma instância da tabela relacional FILIACAO exibida a seguir.

FILIACAO

Pessoa	Genitor
Alexandre	Francisco

Francisco	João
Joaquina	João
Carlota	Joaquina
João	Manuel
Paulo	Manuel
Maria	Paulo

Com relação à tabela FILIACAO, analise o comando SQL a seguir.

```
select distinct t3.Pessoa, t4.Pessoa
FROM FILIACAO t1, FILIACAO t2, FILIACAO t3, FILIACAO t4
WHERE t1.Pessoa < t2.Pessoa
and t1.Genitor = t2.Genitor
and t3.Genitor = t1.Pessoa
and t4.Genitor = t2.Pessoa
```

Afora a linha de títulos, o número de linhas produzidas pela execução do referido comando SQL é:

a) 3;

b) 4;

c) 5;

d) 6;

e) 7.

13. **FGV - AJ (TJ RN)/TJ RN/Apoio Especializado/Análise de Sistemas/2023 -TI - Banco de Dados - Consultas e Comandos em SQL**

Considere a estrutura e uma instância da tabela relacional FILIACAO exibida a seguir.

FILIACAO

Pessoa	**Genitor**
Alexandre	Francisco

Francisco	João
Joaquina	João
Carlota	Joaquina
João	Manuel
Paulo	Manuel
Maria	Paulo

Com relação à tabela FILIACAO, definida anteriormente, o comando SQL que produz a lista dos nomes das pessoas que são citadas na instância da referida tabela, sem repetições, é:

a) select Pessoa FROM FILIACAO

UNION ALL select Genitor FROM FILIACAO

b) select Pessoa FROM FILIACAO

UNION select Genitor FROM FILIACAO

c) select distinct Pessoa, Genitor FROM FILIACAO

d) select distinct Pessoa UNION distinct Genitor

FROM FILIACAO

e) select Pessoa FROM FILIACAO and

select Genitor FROM FILIACAO

14. **FGV - AJ (TJ RN)/TJ RN/Apoio Especializado/Análise de Suporte/2023 -TI - Banco de Dados - Consultas e Comandos em SQL**

Luiz é o DBA do TJRN e atendeu um chamado da equipe de desenvolvimento que pedia para criar um banco de dados com 5 Gigabytes, mas, na hora de criar, Luiz digitou 50 Gigabytes.

Para apagar o banco de dados criado equivocadamente, Luiz deve utilizar o comando:

a) DROP;

b) ALTER;

c) DELETE;

d) UPDATE;

e) INJECTION.

15. **FGV - Ana (BBTS)/BBTS/Perfil Tecnológico/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Considere uma tabela T, com uma única coluna A, definida como uma chave primária, e o comando SQL a seguir.

```
delete from T
where not exists
(select * from T tt where T.A > tt.A)
```

Dado que a tabela tem 100 linhas preenchidas, assinale a opção que indica o número de linhas que será deletado pela execução do referido comando.

a) 0.

b) 1.

c) 50.

d) 99.

e) 100.

16. **FGV - Ana (BBTS)/BBTS/Perfil Tecnológico/2023 - TI - Banco de Dados - Consultas e Comandos em SQL**

Considere o comando de criação de uma tabela relacional num ambiente Oracle exibido a seguir.

```
create table T (
codigo int not null,
produto varchar (40) not null,
preço float not null default 0 check (preço >= 0),
imposto float not null default 0
check (imposto <=100))
```

Assinale o comando SQL de inserção de registro que provocaria um ***erro***.

a) insert into T(produto,código) values ('mesa',100)

b) insert into T values (100, 'mesa',100,10)

c) insert into T values (100, 'mesa',10,100)

d) insert into T(codigo) values (100)

e) insert into T(codigo,produto) values (100,'mesa')

17. **FGV - Analista Legislativo (ALERO)/Tecnologia da Informação/Banco de Dados/2018**

Pessoa	Descendente
Ana	Vitoria
João	Maria
João	Rafael
Maria	Tiago
Natalia	Ana
Rafael	Natalia
Ana	Vitoria

Analise o comando a seguir utilizando a tabela **arvore,** definida anteriormente.

```
delete from arvore
where exists
      (select * from arvore a
       where a.pessoa = 'João'
          and a.descendente = arvore.pessoa)
```

Assinale o número de registros que é removido na execução desse comando.

A Zero.

B Um.

C Dois.

D Três.

E Quatro.

18. **FGV - Analista Especializado (IMBEL)/Analista de Sistemas/2021 (e mais 1 concurso)**

Considere o comando SQL a seguir, executado num banco de dados relacional com duas tabelas, R1 e R2, contendo 2.000 e 5.000 registros, respectivamente. R1 e R2 possuem chaves primárias definidas.

```
SELECT DISTINCT * FROM A, B
```

Assinale a opção que indica o número de linhas produzidas na execução.

A 1

B 2.000

C 5.000

D 7.000

E 10.000.000

Comentário: Questão clássica da FGV, pede para que o candidato calcule a quantidade de linhas geradas por um produto cartesiano. Esse produto pode ser verificado pelos valores A e B, separados por vírgulas, presentes no comando. Neste caso vamos multiplicar a quantidade de linhas da primeira tabela pela quantidade de linhas da segunda (2000x5000) = 10.000.000. Logo, temos a nossa resposta na alternativa E.

Gabarito: E.

19. **FGV - Auditor Fiscal da Receita Estadual (SEFAZ ES)/2021**

Considere um banco de dados relacional contendo as tabelas T, R e S, cujas instâncias são exibidas a seguir.

T

A	B	C
10	LPG Participações	S
20	Serviços & Gerenciamento Remoto	N
50	Academia Americana	S
70	Distribuidora São João de Artigos para Festas	S

R

D	E	F
12040	21/06/2021	200,00
12041	23/06/2021	548,00
1497	15/06/2021	147,10
1498	15/06/2021	85,00
214	18/06/2021	99,00
215	19/06/2021	997,45

S

G	H	I
10	12040	12
10	12041	12
50	1497	12
50	1498	10
70	214	20
50	215	12

Considere a tabela T e a execução dos dois comandos SQL a seguir.

```
SELECT T.*
FROM T LEFT JOIN S ON T.A = S.G
ORDER BY 2

SELECT T.*
FROM T RIGHT JOIN S ON T.A = S.G
ORDER BY 2
```

Sem considerar as linhas de títulos, assinale o número de linhas produzidas por cada comando, na ordem.

A Seis/zero.

B Seis/seis.

C Seis/sete.

D Sete/seis.

E Sete/sete.

20. **FGV - Auditor Fiscal da Receita Estadual (SEFAZ ES)/2021**

No contexto da instância da tabela S, considere a execução do comando SQL a seguir.

```
SELECT *
FROM S
WHERE (NOT G=10 OR I=12)
  AND NOT (H > 100 and H < 1000)
```

Assinale o conjunto de linhas que corresponde ao resultado produzido pelo referido comando.

A
```
10   12040   12
10   12041   12
```

B
```
10   12040   12
10   12041   12
50   1497    12
```

C
```
50   215     12
70   214     20
```

D
```
10   12040   12
10   12041   12
50   1497    12
50   1498    10
```

E
```
50   1497    12
50   1498    10
```

21. **FGV - Auditor Técnico de Controle Externo (TCE-AM)/Auditoria de Tecnologia da Informação/2021**

Na questão, considere uma tabela de banco de dados T cuja instância é exibida a seguir.

A	B	C	D
12	2	3	1
14	3	8	2
18	2	9	3
21	5	4	4

Considerando-se a instância da tabela T (descrita anteriormente), analise o comando SQL abaixo.

select distinct * from T t1, T t2, T t3

A execução desse comando produz um resultado que, além da linha de títulos, contém:

A 8 linhas;

B 24 linhas;

C 32 linhas;

D 64 linhas;

E 128 linhas.

22. **FGV - Auditor Técnico de Controle Externo (TCE-AM)/Auditoria de Tecnologia da Informação/2021**

Na questão, considere uma tabela de banco de dados T cuja instância é exibida a seguir.

A	B	C	D
12	2	3	1
14	3	8	2
18	2	9	3
21	5	4	4

Considerando-se a instância da tabela T (descrita anteriormente), analise o comando SQL abaixo.

```
update T
set a = a + 32
where
exists (select * from T t2 where T.c > t2.D)
```

O número de registros da tabela T afetados pela execução desse comando é:

A zero;

B um;

C dois;

D três;

E quatro.

23. **FGV - Auditor Técnico de Controle Externo (TCE-AM)/Auditoria de Tecnologia da Informação/2021**

Na questão, considere uma tabela de banco de dados T cuja instância é exibida a seguir.

A	B	C	D
12	2	3	1
14	3	8	2
18	2	9	3
21	5	4	4

Considerando-se a instância da tabela T (descrita anteriormente), analise o comando SQL abaixo.

delete from T where b + d = c

O número de registros da tabela T afetados pela execução desse comando é:

A zero;

B um;

C dois;

D três;

E quatro.

24. **FGV - Técnico Superior Especializado (DPE RJ)/Tecnologia da Informação/2019**

FAMILIA

pessoa1	pessoa2	relação
João	Rafael	pai
Maria	Rafael	mãe
Rafael	Gabriela	pai
Gabriela	Rita	mãe
Rita	Bruna	mãe
Bruna	Ana	mãe
Rafael	Rita	avo

Em cada registro, a relação entre a primeira e a segunda pessoa é descrita. Por exemplo, João é pai de Rafael, Gabriela é mãe de Rita, e Rafael, por sua vez, é avô/avó de Rita. Nem todas as relações de avô/avó estão registradas na tabela.

Considere a tabela FAMILIA descrita anteriormente e o comando SQL a seguir.

```
select f1.pessoa1, f2.pessoa2
from família f1, família f2
where f1.pessoa2 = f2.pessoa1
and f1.relação in ('mãe','pai')
and f2.relação in ('mãe','pai')
union
select pessoa1, pessoa2
from familia
where relação = 'avo'
```

Além dos títulos, o número de linhas exibidas na execução desse comando é:

A 1;

B 2;

C 3;

D 4;

E 5.

25. **FGV - Técnico Superior Especializado (DPE RJ)/Tecnologia da Informação/2019**

FAMILIA

pessoa1	pessoa2	relação
João	Rafael	pai
Maria	Rafael	mãe
Rafael	Gabriela	pai
Gabriela	Rita	mãe
Rita	Bruna	mãe
Bruna	Ana	mãe
Rafael	Rita	avo

Em cada registro, a relação entre a primeira e a segunda pessoa é descrita. Por exemplo, João é pai de Rafael, Gabriela é mãe de Rita, e Rafael, por sua vez, é avô/avó de Rita. Nem todas as relações de avô/avó estão registradas na tabela.

Considere a tabela FAMILIA descrita anteriormente e o comando SQL a seguir.

```
select relação, sum(1)
from familia
group by relação
having count(*) > 1
order by 2 desc, 1
```

Os valores exibidos pela execução desse comando, na ordem, são:

A mãe 4
pai 2
avo 1

B mãe 2
pai 4

C pai 2
mãe 4

D mãe 4
pai 2

E mãe 4
pai 2
avo 0

26. **FGV - Auditor Fiscal de Tributos Estaduais (SEFIN RO)/2018**

Considere as tabelas de bancos de dados T1, T2 e T3, que contêm, respectivamente, 10, 500 e 2.000 registros, e o comando SQL a seguir.

select count(*) FROM T1, T2, T3

Assinale a opção que apresenta o número exibido no resultado da execução desse comando.

A 10000000

B 1000000

C 2000

D 500

E 10

27. **FGV - Analista Legislativo Municipal (CM Salvador)/Tecnologia da Informação/2018**

Uma tabela de banco de dados denominada TTT, com atributos A, B e C, contém em sua instância sete registros, com os seguintes valores:

1,2,1

2,3,4

5,4,4

8,7,5

4,3,2

2,3,4

4,3,2

O número de registros removidos pela execução do comando

delete from TTT

where exists

(select * FROM TTT t2

where TTT.a = t2.a

and TTT.b = t2.b)

seria:

A 3;

B 4;

C 5;

D 6;

E 7.

28. **FGV - Analista Legislativo Municipal (CM Salvador)/Tecnologia da Informação/2018**

Uma tabela de banco de dados denominada TT, com atributos A e B, contém em sua instância seis registros, com os seguintes valores:

1,2

2,3

3,4

3,7

4,3

2,3

O número de registros alterados pela execução do comando

update TT set b = b + 1

where TT.a in (select b FROM TT)

seria:

A 1;

B 2;

C 3;

D 4;

E 5.

29. **FGV - Analista de Políticas Públicas e Gestão Governamental (CGM Niterói)/Gestão de Tecnologia/2018**

A otimização de consultas em gerenciadores de bancos de dados é fundamental para o desempenho do sistema. Consultas escritas em SQL são particularmente propícias à otimização, porque essa linguagem

A não é procedural, permitindo a criação de diferentes planos de execução.

B tem uma sintaxe simples e é largamente utilizada.

C suporta todas as operações da Álgebra Relacional.

D permite o uso de subconsultas, facilitando os processos de busca.

E permite a criação de camadas de software de persistência.

30. FGV - Analista de Tecnologia da Informação (BANESTES)/Desenvolvimento de Sistemas/2018

Considere um banco de dados com duas tabelas, R e S, cujas instâncias são exibidas a seguir.

R

a	b
1	2
2	3
4	5

S

c	d
3	2
4	2
6	1

Na execução do comando SQL

select * from R left join S on a=b

UNION

select * from R right join S on b=a

o número de células contendo o valor nulo no resultado é:

A 0;

B 3;

C 6;

D 9;

E 12.

31. **FGV - Analista de Tecnologia da Informação (BANESTES)/Desenvolvimento de Sistemas/2018**

Considere um banco de dados com duas tabelas. A primeira tabela, números, possui dez registros e apenas uma coluna, cujos valores são 1, 2, 3, 4, 5, 5, 9, 9, 9, 10. A segunda tabela, denominada teste, com cinco registros, também possui apenas uma coluna, cujos valores são 1, 3, 3, 4, 5.

Considere ainda o seguinte comando SQL

insert into teste

select numero from numeros n

where not exists

(select * from teste t

where t.numero = n.numero)

Quando da execução desse comando, o número de registros inseridos na tabela teste é:

A 2;

B 3;

C 5;

D 8;

E 10.

32. **FGV - Analista de Tecnologia da Informação (BANESTES)/Desenvolvimento de Sistemas/2018**

Considere uma tabela de bancos de dados com dez registros, e apenas uma coluna cujos valores são 1, 2, 2, 3, 3, 3, 4, 4, 4, 4.

Requisitado para remover os registros com valores repetidos dessa tabela, um programador produziu um script com dois comandos.

delete from exemplo

where exists (select * from exemplo e1

where exemplo.x = e1.x)

select count(distinct x) from exemplo

Na execução desse script, o número produzido no segundo comando foi:

A 0;

B 1;

C 2;

D 3;

E 4.

33. **Ano: 2016 Banca: FGV Órgão: IBGE Cargo: WEB MOBILE Questão: 45**

Os comandos SQL

create table R (a int, b int)

create table S (c int, d int)

insert into R values(1,2)

insert into R values(2,3)

insert into R values (2,3)

insert into R values (3,5)

insert into R values (4,1)

insert into S values (1,2)

insert into S values (2,1)

insert into S values (2,3)

insert into S values (3,5)

select r.a, r.b from R

where not exists

(select * from S where s.c=r.a and s.d=r.b)

Produzem um resultado que, além da linha de títulos, contém:

(A) uma linha;

(B) duas linhas;

(C) três linhas;

(D) quatro linhas;

(E) cinco linhas.

34. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: WEB MOBILE Questão: 46

O comando SQL

select a, sum(b) x, COUNT(*) y

from T

group by a

produz como resultado as linhas abaixo.

a	x	y
1	6	1
3	6	2
4	4	1
5	1	1

Na tabela T, composta por duas colunas, a e b, nessa ordem, há um registro duplicado que contém os valores:

(A) 1 e 3

(B) 3 e 3

(C) 3 e 6

(D) 4 e 2

(E) 5 e 1

35. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 31

Atenção: Algumas das questões seguintes fazem referência a um banco de dados relacional intitulado BOOKS, cujas tabelas e respectivas instâncias são exibidas a seguir. Essas questões referem-se às instâncias mostradas.

AUTOR

AutorID	AutorNome
1	Arthur Conan Doyle
2	Agatha Christie
3	Edgar Allan Poe

LIVRARIA

LivrariaID	LivrariaNome
1	Cultural
2	Travessia
3	Amazonas
4	Kremlin

LIVRO

LivroID	AutorID	Titulo	NumLivrarias
1	1	O Cão dos Baskervilles	NULL
2	1	As Aventuras de Sherlock Holmes	2
3	2	Assassinato no Expresso do Oriente	2
4	2	O Mistério dos Sete Relógios	3
5	3	Assassinatos na Rua Morgue	NULL

OFERTA

LivrariaID	LivroID	Preco
1	1	32
1	2	28
1	3	45
1	4	38
1	5	23
2	1	56
2	2	54
2	4	43
3	3	35
3	4	38

A tabela Livro representa livros. Cada livro tem um autor, representado na tabela Autor. A tabela Oferta representa os livros que são ofertados pelas livrarias, estas representadas pela tabela Livraria. NULL significa um campo não preenchido.

AutorID, LivrariaID e LivroID, respectivamente, constituem as chaves primárias das tabelas Autor, Livraria e Livro.

LivrariaID e LivroID constituem a chave primária da tabela Oferta.

Com relação ao banco de dados BOOKS, analise os comandos SQL exibidos a seguir:

I. select *

from oferta o, livro l, autor a, livraria ll

where o.livroid=l.livroid and

o.livrariaid=ll.livrariaid and l.autorid=a.autorid

II.select *

from oferta o inner join livro l on

o.livroid=l.livroid

inner join autor a on l.autorid=a.autorid

inner join livraria ll on

o.livrariaid=ll.livrariaid

III. select *

from oferta o left join livro l on

o.livroid=l.livroid

left join autor a on l.autorid=a.autorid

left join livraria ll on

o.livrariaid=ll.livrariaid

É correto afirmar que:

(A) somente I e II produzem resultados equivalentes;

(B) somente I e III produzem resultados diferentes;

(C) somente II e III produzem resultados diferentes;

(D) todos os resultados são diferentes;

(E) todos os resultados são equivalentes.

36. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 33

No banco de dados BOOKS, o campo NumLivrarias, da tabela Livro, contém informação redundante, pois denota o número de livrarias que oferecem o livro e pode ser computado. O comando SQL que calcula e atualiza esse campo corretamente é:

(A) update livro

set numlivrarias=

(select count(*) from oferta o where

o.livroid=livro.livroid)

where numlivrarias=null

(B) update livro

set numlivrarias=

(select count(*) from oferta o where

o.livroid=livro.livroid)

(C) update numlivrarias

from livro as

(select count from oferta o where

o.livroid=livro.livroid)

where numlivrarias=null

(D) set livro.numlivrarias=

(select count from oferta o where

o.livroid=livro.livroid)

(E) set livro.numlivrarias=

(select count(livrariaid) from oferta o where

o.livroid=livro.livroid)

where numlivrarias=null

37. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 34

Quando executado no contexto do banco de dados BOOKS, o comando SQL

select numlivrarias from livro

where numlivrarias > 0

union

select numlivrarias from livro

where numlivrarias <= 0

produz um resultado cujo número de linhas, além da linha de título, é:

(A) 1

(B) 2

(C) 3

(D) 4

(E) 5

38. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: DESENVOLVIMENTO DE SISTEMAS Questão: 41

João foi incumbido de rever um lote de consultas SQL. Como ainda é iniciante nesse assunto, João solicitou ajuda ao colega que lhe pareceu ser o mais experiente, e recebeu as seguintes recomendações gerais:

I. use a cláusula DISTINCT somente quando estritamente necessária;

II. dê preferência às junções externas (LEFT, RIGHT, OUTER) em relação às internas (INNER);

III. use subconsultas escalares no comando SELECT, tais como “SELECT x,y,(SELECT ...) z ..." sempre que possível.

Sobre essas recomendações, é correto afirmar que:

(A) nenhuma é adequada;

(B) somente I é adequada;

(C) somente I e II são adequadas;

(D) somente II e III são adequadas;

(E) todas são adequadas.

39. Ano: 2016 Banca: FGV Órgão: IBGE Cargo: SUPORTE OPERACIONAL Questão: 42

João escreveu a consulta SQL a seguir, executou-a corretamente e obteve um resultado contendo 100 linhas, além da linha de títulos.

SELECT curso, nome

FROM aluno, curso

WHERE aluno.codcurso = curso.codcurso

ORDER BY curso, nome

As tabelas aluno e curso possuem, respectivamente, 120 e 12 linhas. No banco há ainda outras duas tabelas, pauta e disciplina, com 200 e 5 registros, respectivamente. Nessas condições, o número de linhas, além da linha de títulos, produzidas pelo comando

SELECT curso, nome

FROM aluno, curso, disciplina, pauta

WHERE aluno.codcurso = curso.codcurso

ORDER BY curso, nome

seria:

(A) 100;

(B) 305;

(C) 500;

(D) 20.000;

(E) 100.000.

40. BANCA: FGV ANO: 2014 ÓRGÃO: CM-RECIFE PROVA: ANALISTA LEGISLATIVO - ANALISTA DE SISTEMAS

No SQL, o comando grant permite outorgar a um usuário (ou papel) privilégios sobre determinados recursos. Quando usado com a opção with grant option, o comando grant permite que:

A no caso de privilégios outorgados a papéis seja possível identificar usuários que excepcionalmente não devem receber esses privilégios;

B os privilégios outorgados não possam ser alvo de comandos revoke emitidos por usuários diferentes daquele que outorgou incialmente;

C os privilégios sejam outorgados em caráter temporário, sendo automaticamente removidos quando da expiração do prazo estabelecido;

D o usuário que recebe um privilégio possa concedê-lo a outros usuários;

E os privilégios sejam concedidos condicionalmente, e posteriormente confirmados mediante a execução automática de um procedimento de autorização.

41. BANCA: FGV ANO: 2015 ÓRGÃO: CM CARUARU PROVA: ANALISTA LEGISLATIVO - INFORMÁTICA

Em geral, a definição de chaves estrangeiras em bancos de dados relacionais pode vir acompanhada da especificação de procedimentos adicionais a serem adotados quando da exclusão/alteração de valores nos registros da tabela estrangeira.

```
create table R2 (
        a int not null primary key,
        b int null,
        x int not null,
        constraint FK foreign key (x) references
R1(x)
)
```

Considerando o script acima, as opções complementares compatíveis para a definição da chave estrangeira FK são:

A on delete cascade

on update set null

B on delete cascade

on update cascade

C on delete no action

on update set null

D on delete set null

on update restrict

E on delete restrict

on update set null

42. ANO: 2015 BANCA: FGV ÓRGÃO: TCE-SE PROVA: ANALISTA DE TECNOLOGIA DA INFORMAÇÃO - SEGURANÇA DA INFORMAÇÃO

Considere duas tabelas X e Y, com as seguintes instâncias:

X	
a	b
1	2
3	3
4	5
5	7

Y	
c	d
1	2
3	4
5	6
7	8
9	1

O comando SQL que retorna

a	b	c	d
1	2	1	2
3	3	3	4
4	5	NULL	NULL
5	7	5	6
NULL	NULL	7	8
NULL	NULL	9	1

é:

A
```
select *
from X FULL JOIN Y on X.a=Y.c
```

B
```
select *
from X LEFT JOIN Y on X.a=Y.c
```

C
```
select *
from Y RIGHT JOIN X on X.a=Y.c
```

D
```
select *
from X CROSS JOIN Y on X.a=Y.c
```

E
```
select *
from X INNER JOIN Y on X.a=Y.c
```

43. Ano: 2013 Banca: FGV Órgão: SUDENE-PE Cargo: Analista Técnico Administrativo - Ciência da Computação

Com relação aos bancos de dados, os índices são uma das técnicas mais utilizadas na otimização de desempenho de consultas SQL.

A respeito dos índices, assinale V para a afirmativa verdadeira e F para a falsa.

(...) Os índices provavelmente serão utilizados quando uma coluna indexada aparecer nos critérios de busca de uma cláusula WHERE ou HAVING.

(...) Os índices provavelmente serão utilizados quando uma coluna indexada aparecer em uma cláusula GROUP BY e ORDER BY.

(...) Os índices provavelmente serão utilizados quando a seletividade dos dados de uma coluna indexada for baixa.

As afirmativas são, respectivamente,

A) F, V e F.

B) V, V e F.

C) V, F e F.

D) V, F e V.

E) F, F e V.

44. Ano: 2013 Banca: FGV Órgão: MPE-MS Cargo: Técnico - Informática

Observe o comando SQL a seguir:

SELECT nome, sobrenome, PIS, anos_de_servico FROMEmpregados

A cláusula que deve ser adicionada ao comando acima para ordenar os registros por anos de serviço, com os empregados que estão há mais tempo na empresa aparecendo primeiro na listagem, é

A) ORDER 'anos_de_servico' BY ASC

B) ORDER BY 'anos_de_servico'

C) ORDER BY anos_de_servico DESC

D) SORTED BY anos_de_servico DESC

E) ORDER BY anos_de_servico ASC

Comentário: Percebam que quanto maior o tempo de empresa maior será a quantidade de anos de serviço. Desta forma, como a questão pede que os empregados a mais tempo na empresa apareçam primeiro devemos ordenar de forma decrescente. Outro ponto importante para responder à questão é utilizar a sintaxe correta de ordenação em SQL, qual seja, ORDER BY nome_do_campo DESC. Desta forma, podemos marcar o gabarito correto na alternativa C.

Gabarito: C.

---

45. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas

A necessidade de construir consultas aplicadas a sistemas de apoio à decisão levou à introdução de algumas construções especiais na linguagem SQL, que facilitam e estendem a agregação de dados. Dentre essas estão:

A) oltp e olap;

B) intersection e minus;

C) rank e decode;

D) cube e rollup;

E) slicing e dicing.

46. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas

Considere as seguintes tabelas relacionais e respectivas instâncias.

R

A	B
1	3
2	3
3	4
5	5
5	7
8	8

S

C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

SELECT A,

(**SELECT COUNT** (*)

FROM S **WHERE NOT** R.B = S.C **OR** R.A = S.D

) X **FROM** R

ORDER BY A;

Os números que aparecem na coluna X do resultado da execução desse comando, de cima para baixo, são:

A) 0, 0, 1, 0, 0, 0.

B) 0, 0, 0, 0, 0, 0.

C) 2, 2, 5, 5, 5, 5.

D) 2, 2, 4, 5, 5, 5.

E) 3, 3, 1, 0, 0, 0.

47. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas

Considere as seguintes tabelas relacionais e respectivas instâncias.

R

A	B
1	3
2	3
3	4
5	5
5	7
8	8

S

C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

DELETE FROM S

WHERE NOT EXISTS

(**SELECT** * **FROM** R

WHERE R.A = S.C **AND** R.B = S.D)

O número de registros deletados por esse comando é:

A) 0;

B) 1;

C) 2;

D) 4;

E) 5.

48. Ano: 2015 Banca: FGV Órgão: TJ-RO Cargo: Analista de Sistemas

Considere as seguintes tabelas relacionais e respectivas instâncias.

R

A	B
1	3
2	3
3	4
5	5
5	7
8	8

S

C	D
1	9
2	3
3	4
3	4
3	4

Analise o comando SQL a seguir.

SELECT * FROM R UNION SELECT * FROM S

O número de linhas produzidas por esse comando, excetuada a linha de títulos de colunas, é:

A) 2;

B) 5;

C) 6;

D) 7;

E) 11.

49. Ano: 2015 Banca: FGV Órgão: DPE-RO Cargo: Analista de redes e comunicação de dados

Observe o comando SQL a seguir.

UPDATE X SET Y = 'Z';

Para que esse comando esteja corretamente formulado, quando analisado isoladamente, pressupõe-se que:

A) Y seja uma coluna da tabela X;

B) X seja uma coluna da tabela Y;

C) X e Y sejam tabelas;

D) X seja um banco de dados e Y seja uma tabela;

E) Y seja uma coluna da tabela X e Z seja o nome de um tipo de dados válido.

50. Ano: 2015 Banca: FGV Órgão: DPE-RO Cargo: Analista de redes e comunicação de dados

Analise o comando SQL a seguir.

SELECT DISTINCT 1 FROM X

Sabendo-se que a instância da tabela X não é vazia, conclui-se que a execução desse comando produz um resultado com:

A) apenas uma linha;

B) um conjunto de linhas contendo o valor NULL;

C) um conjunto de linhas contendo todos os atributos de X;

D) um número de linhas igual ao número de diferentes valores do primeiro atributo de X;

E) um número de linhas igual ao número de registros de X.

51. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação

X	
a	b
1	2
3	3
4	5
5	7

Y	
c	d
1	2
3	4
5	6
7	8
9	1

delete from y

where y.c in

(select a from x union select c from y)

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que o número de registros removidos da tabela Y pela execução desse comando é:

A) 1

B) 2

C) 3

D) 4

E) 5

52. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação

X	
a	b
1	2
3	3
4	5
5	7

Y	
c	d
1	2
3	4
5	6
7	8
9	1

SELECT X.a **FROM** X

WHERE NOT EXISTS

(SELECT * FROM Y WHERE Y.c = X.a + 1)

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que esse comando produz um resultado com uma única coluna contendo somente o (s) valor (es):

A) 4

B) 3, 4

C) 1, 3, 5

D) 3, 4, 5

E) 1, 3, 4, 5

53. Ano: 2015 Banca: FGV Órgão: TCE-SE Cargo: Analista de Tecnologia da Informação

X	
a	b
1	2
3	3
4	5
5	7

Y	
c	d
1	2
3	4
5	6
7	8
9	1

SELECT *

FROM X LEFT JOIN Y ON X.a = Y.c

ORDER BY X.a

Considerando-se as tabelas e o comando SQL mostrados acima, é correto concluir que esse comando produz:

A) 1, 2, 1, 2

3, 3, 3, 4

5, 7, 5, 6

B) 1, 2, 1, 2

3, 4, 3, 3

NULL, NULL, 4, 5

5, 6, 5, 7

C) 1, 2, 1, 2

3, 3, 3, 4

NULL, NULL, NULL, NULL

5, 7, 5, 6

D) 1, 2, 1, 2

3, 3, 3, 4

4, 5, NULL, NULL

5, 7, 5, 6

E) 1, 2

3, 3

4, 5

5, 7

54. Ano: 2015 Banca: FGV Órgão: TCM-SP Cargo: Tecnologia da Informação

Views criadas nos bancos podem, de acordo com alguns critérios, ser naturalmente atualizáveis, o que significa, por exemplo, que podem ser objeto de comandos update do SQL sem a necessidade de mecanismos auxiliares ou triggers. Essa característica depende da expressão SQL que define a view e das tabelas/views de origem.

Considere alguns tipos de construções SQL que podem ser empregadas na definição de uma coluna de uma view:

I. funções de agregação, tais como sum, avg

II. funções escalares, tais como sin, trim

III. expressões aritméticas

IV. expressões condicionais, tais como case

V. literais

VI. subconsultas

Está correto concluir que uma determinada coluna **NÃO** pode ser objeto de atualização quando resultar de qualquer dos tipos:

A) apresentados, exceto I, II e III;

B) apresentados, exceto III e IV;

C) apresentados, exceto V;

D) apresentados, exceto VI;

E) apresentados.

55. Ano: 2015 Banca: FGV Órgão: TCM-SP Cargo: Tecnologia da Informação

Considere a tabela relacional criada pelo comando

create table xx

(a int null, b int null, c int null)

Depois de instanciada com um conjunto de registros, os seguintes comandos foram executados:

select count (*) from XX

select count (distinct A) from XX

select count (distinct B) from XX

select count (*) from XX where C>10

select count (*) from XX where not C>10

Sabendo-se que esses comandos produziram como resultado, respectivamente, os números 10, 10, 0, 0 e 5, analise as quatro alternativas para a definição da tabela XX:

I. CREATE TABLE XX(
A int NULL,
B int NULL,
C int NULL)

II.CREATE TABLE XX(
A int primary key,
B int NULL,
C int NULL)

III.CREATE TABLE XX (
A int NULL,
B int NULL,
C int)

IV. CREATE TABLE XX (
A int,
B int primary key,
C int NULL)

A lista com todos os comandos que são válidos e compatíveis com a instância corrente da tabela é:

A) I, II;

B) I, II, III;

C) II, IV;

D) I, III;

E) IV.

56. Ano: 2014 Banca: FGV Órgão: SUSAM Cargo: Técnico de Nível Superior A - Analista de Sistemas

A figura a seguir apresenta o diagrama das tabelas TBNotaFiscal e TBItemNotaFiscal. As tabelas 1 e 2 apresentam, respectivamente, os registros TBNotaFiscal e TBItemNotaFiscal.

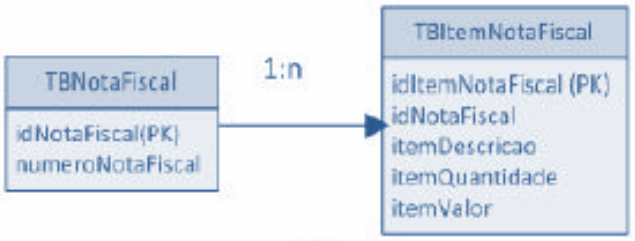


Tabela 1

TBNotaFiscal	
idNotaFiscal	numeroNotaFiscal
1	NF 0001
2	NF 0002
3	NF 0003

Tabela 2

TBItemNotaFiscal				
idItemNotaFiscal	idNotaFiscal	itemDescricao	itemQuantidade	itemValor
1	1	porca	100	0,10
2	1	parafuso	50	0,20
3	2	areia	10	10,50
4	2	cimento	15	20,00
5	2	telha	100	2,50

Os campos idCliente e idCompras são chaves primárias das tabelas TBNotaFiscal e TBItemNotaFiscal, respectivamente.

Assinale a opção que indica resposta correta para o seguinte comando SQL

```
SELECT a.numeroNotaFiscal, b.itemDescricao,
b.itemValor, b.itemQuant, b.itemValorAS
ValorUnitario,(b.itemValor)*(b.itemQuant)AS
ValorTotal
FROM TBNotaFiscal a, TBItemNotaFiscal b
```

WHERE a.idNotaFiscal = b.idNotaFiscal

A) NF 0001 porca 0,1 100 0,1 10
NF 0001 parafuso 0,2 50 0,2 10
NF 0002 areia 10,5 10 10,5 105
NF 0002 cimento 20 15 20 300
NF 0002 telha 2,5 100 2,5 250

B) NF 0001 porca 0,1 100 0,1 10
NF 0001 parafuso 0,2 50 0,2 10
NF 0002 areia 10,5 10 10,5 105
NF 0002 cimento 20 15 20 300
NF 0002 telha 2,5 100 2,5 250
NF 0003 NULL NULL NULL NULL NULL

C) NF 0001 porca 0,1 100 0,1 10
NF 0001 parafuso 0,2 50 0,2 10
NF 0002 areia 10,5 10 10,5 105
NF 0002 cimento 20 15 20 300
NF 0002 telha 2,5 100 2,5 250
NF 0003 NULL 0 0 0 0

D) NF 0001 NULL NULL NULL NULL 20
NF 0002 NULL NULL NULL NULL 655
NF 0003 NULL NULL NULL NULL 0

E) NF 0001 NULL NULL NULL NULL 20
NF 0002 NULL NULL NULL NULL 655

57. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação

Considere que as instâncias das tabelas T1, T2 e T3 têm, respectivamente, 1.000, 10.000 e 100.000 registros. O comando SQL

select 1 from t1

union

select 2 from t2

union

select 3 from t3

produz um resultado com:

A) 3 linhas;

B) 1.000 linhas;

C) 10.000 linhas;

D) 100.000 linhas;

E) 111.000 linhas.

58. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação

Analise os comandos SQL a seguir, que produzem os resultados R1, R2 e R3, respectivamente.

I. select distinct x.* from x, y where x.a <> y.a

II. select distinct x.* from x

where x.a not in (select a from y)

III.select distinct x.* from x

where not exists

(select * from y where y.a=x.a)

Sabendo-se que nenhuma das instâncias das tabelas “x” e “y” é vazia, é correto concluir que:

A) R1 e R2 são iguais entre si e diferentes de R3;

B) R1 e R3 são iguais entre si e diferentes de R2;

C) R2 e R3 são iguais entre si e diferentes de R1;

D) R1, R2 e R3 são todos iguais entre si;

E) R1, R2 e R3 são todos diferentes entre si.

59. Ano: 2015 Banca: FGV Órgão: TJ-BA Cargo: Analista Judiciário - Tecnologia da Informação

Analise as instâncias das tabelas R1 e R2 e o comando SQL, mostrados a seguir.

R1

x
1
3
4
6
7
8

R2

a	b
2	0
5	0
7	0
9	0
10	0

```
update R2
set b=(select count(x) from R1 where x > a)
```

Após a execução do comando, o conteúdo da coluna “b" da tabela R2 passa a ser, de cima para baixo:

A) 5, 5, 5, 5, 5;

B) NULL, NULL, NULL, NULL, NULL;

C) 5, 3, 1, 0, 0;

D) 0, 0, 0, 0, 0;

E) 5, 3, 1, NULL, NULL.

60. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas

Analise o comando de criação de uma tabela relacional mostrado a seguir.

```
create table inscrição
(matricula int not null,
  disciplina int not null,
  nota float null,
  constraint pk_ inscrição primary key (matricula,
disciplina),
  constraint fk_inscrição_aluno
        foreign key (matricula) references aluno
(matricula)
)
```

Sabendo-se que a coluna matricula constitui a chave primária da tabela aluno, está correto concluir que

A) aluno e inscrição têm um relacionamento 1:1 entre si.

B) aluno e inscrição têm um relacionamento 1:n entre si.

C) aluno e inscrição têm um relacionamento n:1 entre si.

D) aluno e inscrição têm um relacionamento m:n entre si.

E) inscrição é uma especialização de aluno.

61. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas

Analise o comando SQL a seguir.

```
select x,
       sum (nota) as soma,
       count (nota) as numero
from inscricao
group by x
having ???
```

Assinale a opção que indica a expressão que, ao ser utilizada para substituir o trecho "???", **INVALIDA** o comando SQL acima.

A) count (nota) > 1

B) x=2

C) max (nota) = 10

D) nota > 7

E) (select max(nota) from inscricao) > 5

62. Ano: 2015 Banca: FGV Órgão: DPE-MT Cargo: Analista - Análise de Sistemas

Na maioria das implementações SQL, pode-se considerar que as expressões lógicas possam assumir três valores, verdadeiro (T), falso (F) e desconhecido (?). Isso decorre principalmente da manipulação de valores nulos (NULL).

Assim sendo, analise as quatro expressões lógicas a seguir.

not ?

F or ?

T and ?

? or T

Assinale a opção que apresenta os valores finais das expressões lógicas acima, na ordem de cima para baixo.

A) F; ?; T; T

B) F; F; T; T

C) ?; ?; ?; ?

D) ?; ?; ?; T

E) ?; F; ?; ?

GABARITO

1. D
2. B
3. D
4. B
5. A
6. D
7. E
8. D
9. B
10. A
11. B
12. A
13. B
14. A
15. B
16. D
17. C
18. E
19. D
20. D
21. D
22. E
23. B
24. E
25. D
26. A
27. E
28. E
29. A
30. E
31. C
32. A
33. A
34. B
35. A
36. B
37. B
38. B
39. E
40. D
41. B
42. A
43. B
44. C
45. D
46. D
47. B
48. D
49. A
50. A
51. E
52. C
53. D
54. E
55. B
56. A
57. A
58. C
59. C
60. B
61. D
62. D

Lista de Questões Vunesp

Lista de Questões Vunesp

1. (VUNESP - Ana SJ (TJM SP)/TJM SP/2023)

Considere a seguinte consulta SQL feita em um banco de dados relacional:

```
SELECT Operadora, Plano
 FROM Conta
 WHERE Plano LIKE 'Special%'
 ORDER BY Operadora
```

Um possível resultado obtido a partir dessa consulta é

a)

Operadora	Plano
Abc	Special Plus
Camel	Special X
Lotus	Special 1

b)

Operadora	Plano
Lotus	Special 1

Camel	Special X	
Abc	Top Plus	Special

c)

Operadora	Plano	
Lotus	Special 1	
Camel	Special X	
Abc	Top Plus	Special

d)

Operadora	Plano	
Lotus	Plano 1	Special
Camel	Plano X	Special
Abc	Top Plus	Special

e)

Operadora	Plano	

Abc	1 Special	
Camel	X Special	
Lotus	Top Special	Plus

2. (VUNESP - ATI (TJ RS)/TJ RS/Análise de Sistemas/2023)

Considere a seguinte tabela de um banco de dados relacional:

Aluno (ID, Nome, Curso)

O comando SQL para obter o nome e o curso dos alunos, com a condição de que o curso se inicie com a palavra Física é:

a) SELECT Nome, Curso
FROM Aluno
WHERE Curso == Física%;

b) SELECT Nome, Curso
FROM Aluno
WHERE Curso.Física%;

c) SELECT Nome, Curso
FROM Aluno
WHERE Curso LIKE Física%;

d) SELECT Nome, Curso
FROM Aluno
WHERE Curso(Física%);

e) SELECT Nome, Curso

FROM Aluno

WHERE Curso = Física%;

3. (VUNESP - Ana (Pref Marília)/Pref Marília/Dados/2023)

O comando SQL para excluir completamente uma tabela (dados e estrutura) denominada Paper de um banco de dados relacional é

a) UNDO Paper.

b) EXC Paper.

c) DROP Paper.

d) DELETE Paper.

e) TRASH Paper.

4. (VUNESP - Ana (Pref Marília)/Pref Marília/Dados/2023)

Considere a tabela de um banco de dados relacional

TX (A, B, C, D)

e a seguinte consulta sobre essa tabela

```
SELECT C, D
FROM TX
WHERE B> 30 AND C < 50
```

Como resultado da execução dessa consulta, serão exibidos apenas os valores dos atributos

a) C e D.

b) B e C.

c) A e C.

d) B, C e D.

e) A, C e D.

5. (VUNESP - PTIC (UNICAMP)/UNICAMP/Programador de Sistemas de Informação/2023)

Deseja-se criar uma tabela em um banco de dados relacional, contendo dois atributos (A e B), ambos do tipo string com 20 caracteres cada um. Ambos devem compor a chave primária da tabela, a ser denominada Tab.

O comando SQL para realizar a tarefa aqui descrita é:

a) CREATE TABLE Tab
(A Char (20),
B Char (20),
Primary Key (A, B);

b) CREATE TABLE Tab
(A, B Char (20),
Primary Key (A, B);

c) CREATE TABLE Tab
(A AND B Char (20),
Primary Key (A, B);

d) CREATE TABLE Tab
(A Char(20) NOT NULL,
B Char (20) NOT NULL);

e) CREATE TABLE Tab
(A, B Char (20), PK);

6. (VUNESP - PTIC (UNICAMP)/UNICAMP/Programador de Sistemas de Informação/2023)

Considere uma tabela denominada Tablet, de um banco de dados relacional. É necessário acrescentar um novo atributo (Fonte) a essa tabela, do tipo inteiro (supor que haja o tipo Integer no gerenciador a ser utilizado). O comando SQL para realizar tal tarefa é:

a) ALTER TABLE Tablet

PLUS Fonte (Integer);

b) ALTER TABLE Tablet

ADD Fonte (Integer);

c) ALTER TABLE

ADD Tablet.Fonte (Integer);

d) ALTER TABLE Tablet >> Fonte (Integer);

e) ALTER TABLE Tablet ++ Fonte (Integer);

7. (VUNESP - PTIC (UNICAMP)/UNICAMP/Analista de Desenvolvimento de Sistemas/2023)

Considere a seguinte tabela de um banco de dados relacional.

Sala (Número, Capacidade, Nome)

O comando SQL para obter o número, a capacidade e o nome das salas, considerando aquelas com capacidade maior do que 4 é:

a) SELECT Sala. *
WHERE Capacidade > 4;

b) SELECT Sala (Número, Capacidade, Nome)
WHERE Capacidade > 4;

c) SELECT Sala (*)
WHERE Capacidade > 4;

d) SELECT Número ... Nome
FROM Sala
WHERE Capacidade > 4;

e) SELECT *
FROM Sala
WHERE Capacidade > 4;

8. (VUNESP - PTIC (UNICAMP)/UNICAMP/Analista de Desenvolvimento de Sistemas/2023)

Considere a seguinte tabela de um banco de dados relacional.

Veículo (Placa, Modelo, Marca, Tipo, Ano)

O comando SQL para obter parte dos atributos (placa, modelo e marca) de veículos fabricados no ano 2020 e do tipo SUV ou Misto é:

a) SELECT *
FROM Veículo
WHERE Ano = 2020 AND Tipo INTO ('SUV', 'Misto');

b) SELECT Placa, Modelo, Marca
FROM Veículo
WHERE Ano = 2020 AND Tipo ='SUV' AND
Tipo = 'Misto';

c) SELECT Placa ... Marca
FROM Veículo
WHERE Ano = 2020 AND Tipo LIKE ('SUV', 'Misto');

d) SELECT Placa, Modelo, Marca
FROM Veículo
WHERE Ano = 2020 AND Tipo IN ('SUV', 'Misto');

e) SELECT *
FROM Veículo
WHERE Ano = 2020 AND (Tipo ='SUV',
Tipo = 'Misto');

9. (VUNESP - PTIC (UNICAMP)/UNICAMP/Analista de Desenvolvimento de Sistemas/2023)

Considere a seguinte tabela de um banco de dados relacional.

Consultório (Sala, Bloco, Tipo, Proprietário)

O comando SQL para criar um índice baseado no atributo Proprietário, índice esse de nome First, é:

a) CREATE INDEX First

ON Consutório (Proprietário);

b) CREATE INDEX Consultório (Proprietário)

AT Consultório.First;

c) CREATE INDEX Consultório.First

FOR Proprietário;

d) CREATE INDEX First

IN Proprietário OF Consultório;

e) CREATE INDEX ON First.

Consultório.Proprietário;

10. (VUNESP - PTIC (UNICAMP)/UNICAMP/Analista de Desenvolvimento de Sistemas/2023)

Considere as seguintes tabelas de um banco de dados relacional.

Computador (ID, Modelo, Fabricante)
Usuário (Código, Nome, Função)
Usa (Código, ID), sendo Código e ID chaves estrangeiras
com origem nas tabelas Usuário e Computador, respectivamente.

O comando SQL para obter nome do usuário e modelo de computador utilizado é:

a) SELECT Nome, Modelo
FROM Computador, Usuário;

b) SELECT Nome, Modelo
FROM Computador C, Usuário U, Usa S
WHERE C.ID = S.ID AND U.Código = S.Código;

c) SELECT Nome, Modelo
FROM Computador C, Usuário U, Usa S
WHERE ID (C, S) AND Código (U, S);

d) SELECT Nome, Modelo
FROM Computador C, Usuário U, Usa S
WHERE C AND S (ID); S AND U (Código);

e) SELECT Nome, Modelo
FROM Computador C, Usuário U, Usa S
WHERE C (*) = S (*) AND U (*) = S (*);

11. (VUNESP - WDes (Pref Pinda)/Pref Pindamonhangaba/2023)

Considere as seguintes tabelas de um banco de dados relacional:

Disciplina (Código, Nome-D, ID)

Curso (ID, Nome-C, Descrição)

sendo que, na tabela Disciplina, ID é uma chave estrangeira, com origem na tabela Curso.

O comando SQL para obter os nomes das disciplinas e o nome do curso a que a disciplina pertence é

a) SELECT Nome-D, Nome-C

FROM Disciplina D, Curso C;

b) SELECT Nome-D, Nome-C

FROM Disciplina D.ID, Curso C.ID;

c) SELECT Nome-D, Nome-C

WHERE Disciplina D.ID IN Curso C.ID;

d) SELECT Nome-D, Nome-C

FROM Disciplina D, Curso C

AND Curso.ID = Disciplina.ID;

e) SELECT Nome-D, Nome-C

FROM Disciplina D, Curso C

WHERE Curso.ID = Disciplina.ID;

12. (VUNESP - WDes (Pref Pinda)/Pref Pindamonhangaba/2023)

Considere a seguinte tabela de um banco de dados relacional:

Cliente (CPF, Nome, Peso)

O comando para excluir os clientes de peso superior a 80 kg é (supondo o atributo peso expresso em kg) é:

a) DELETE Peso.Cliente > 80;

b) DELETE Cliente.Peso > 80;

c) DELETE FROM Cliente

WHERE Peso > 80;

d) DELETE Peso > 80

FROM Cliente;

e) DELETE FROM Cliente

WHERE Peso INF 80;

13. (VUNESP - ATI (TJ RS)/TJ RS/Análise de Suporte/2023)

Considere a seguinte tabela de um banco de dados relacional:

Loja (ID, Nome, Endereço, Cidade, Estado)

O comando SQL para obter o nome e número de lojas que não estão no Estado de São Paulo é:

a) SELECT Nome, Sum (Estado)

FROM Loja

WHERE Estado != 'São Paulo;

b) SELECT Nome, Count (Estado)

FROM Loja

WHERE NOT (Estado.'São Paulo')

c) SELECT Nome, Count (Estado)

FROM Loja

WHERE Estado <> 'São Paulo';

d) SELECT Nome, Add (Estado)

FROM Loja

WHERE Estado.~'São Paulo';

e) SELECT Nome, Join (Estado)

FROM Loja

WHERE Estado NOT IN ('São Paulo');

14. (VUNESP - TTI (TJ RS)/TJ RS/Programador/2023)

Um Técnico de Tecnologia da Informação do TJ-RS se deparou com uma situação na qual, sempre que o sistema adicionasse ou excluísse um registro de processo, seria preciso atualizar automática e simultaneamente uma tabela com a função de contador.

Para a resolução desse problema, considerando um banco de dados relacional e SQL, o procedimento ou funcionalidade mais indicado é:

a) Procedure.

b) View.

c) Trigger.

d) Job.

e) Sequence.

15. (VUNESP - ADP (DPE SP)/DPE SP/Administrador de Redes/2023)

Considere a seguinte tabela de um banco de dados relacional:

Armário (ID, Tipo, Item, Peso)

O comando SQL para obter o número total de registros contidos na tabela Armário é:

a) COUNT (ID) FROM Armário;

b) NUM (ID) FROM Armário;

c) SELECT COUNT(ID) FROM Armário;

d) COUNT ALL FROM Armário;

e) SELECT COUNT ALL FROM Armário;

16. (VUNESP - ADP (DPE SP)/DPE SP/Analista Desenvolvedor/2023)

Considere a seguinte tabela de um banco de dados relacional.

Estoque (ID, Tipo, Item, Qtdade)

O comando SQL para obter o Tipo e a Quantidade do estoque, agrupado por Tipo, contido na tabela é:

a) SELECT Tipo, COUNT (Qtdade)

FROM Estoque;

b) SELECT Tipo, SUM (Qtdade)

FROM Estoque;

c) SELECT Tipo, Qtdade

FROM Estoque

GROUP BY COUNT(Tipo);

d) SELECT Tipo, Qtdade

FROM Estoque

GROUP BY SUM(Tipo);

e) SELECT Tipo, SUM (Qtdade)

FROM Estoque

GROUP BY (Tipo);

17. (VUNESP - CFO/QC (EsFCEx)/EsFCEx/Informática/2023)

Considere a seguinte tabela de um banco de dados relacional:

Time (ID, Nome, Cidade, Estado)

O comando SQL para obter o número de registros de cada Estado presente na tabela, além do nome do Estado, é:

a) SELECT Count (Estado), Estado FROM Time GROUP BY Estado;

b) SELECT Sum (Estado), Estado FROM Time ORDER BY Estado;

c) SELECT Count (Estado), Estado FROM Time HAVING Estado IN Group;

d) SELECT Max (Estado) – Min (Estado) FROM Time GROUP BY Estado;

e) SELECT Number (Estado), Estado FROM Time WHERE Estado IS NOT NULL;

18. (VUNESP - CFO/QC (EsFCEx)/EsFCEx/Informática/2023)

Considere a seguinte tabela de um banco de dados relacional:

Material (ID, Nome, Descrição, Fabricante)

O comando SQL para obter o nome e o fabricante, com a condição de excluir os fabricantes Blue e Green é:

a) SELECT Nome, Fabricante FROM Material WHERE Fabricante NOT ('Blue','Green');

b) SELECT Nome, Fabricante FROM Material WHERE Fabricante DISTINCT ('Blue','Green');

c) SELECT Nome, Fabricante FROM Material WHERE Fabricante <> 'Blue' OR 'Green'

d) SELECT Nome, Fabricante FROM Material WHERE NOT Fabricante ('Blue','Green');

e) SELECT Nome, Fabricante FROM Material WHERE Fabricante NOT IN ('Blue', 'Green');

19. VUNESP - Tec CPDJ (TJM SP)/TJM SP/2023

Considere a seguinte tabela de um banco de dados relacional:

Passagem (ID, Origem, Destino, Valor)

O comando SQL para obter a Origem e Destino para valores situados entre 200,00 e 1.000,00, ordenado pela Origem, é:

a) SELECT Origem, Destino

FROM Passagem

HAVING Valor BETWEEN (200.00; 1000.00);

b) SELECT Origem, Destino

FROM Passagem

ORDER BY Origem AND

(Valor >= 200.00 OR Valor <= 1000.00);

c) SELECT Origem, Destino

FROM Passagem

WHERE Origem ASC AND

Valor BETWEEN (200.00 <> 1000.00);

d) SELECT Origem, Destino

FROM Passagem

WHERE Valor BETWEEN (200.00 AND 1000.00)

ORDER BY Origem;

e) SELECT Origem, Destino

FROM Passagem

HAVING Origem ASC AND

Valor BETWEEN (200.00 UNTIL 1000.00);

20. (VUNESP - Ana (Pref Marília)/Pref Marília/Dados/2023)

A sintaxe básica do comando de criação de gatilhos em SQL é:

a) CREATE FUNCTION ...

b) CREATE TRIGGER ...

c) SELECT TRIGGER ...

d) DROP TRIGGER ...

e) INSERT TRIGGER ...

21. (VUNESP - ATI (UFABC)/UFABC/2023)

Considere a seguinte tabela de um banco de dados relacional:

Paciente (ID, Nome, Idade, Convenio, Valor)

O comando SQL para majorar em 12% o valor para os convênios denominados Único e Final é:

a) OUTPUT Valor = Valor * 1.12
FROM Convenio = 'Único' AND Convenio =
'Final' FOR Paciente;

b) SELECT Valor = Valor * 1.12
HAVING Convenio = 'Único' AND Convenio =
'Final' IN Paciente;

c) WRITE Convenio.Valor = Convenio.Valor * 1.12
FROM Paciente.Convenio IN ('Único' OR 'Final');

d) UPDATE Paciente.Valor = Paciente.Valor * 1.12
FROM Convenio = 'Único' AND Convenio = 'Final';

e) UPDATE Paciente
SET Valor = Valor * 1.12
WHERE Convenio IN ('Único','Final');

22. (VUNESP - Téc Lab (UFABC)/UFABC/Informática/2023)

Considere a seguinte tabela de um banco de dados relacional:

Veiculo (RENAVAM, Marca, Modelo, Ano)

O comando SQL para excluir os veículos de modelo Sedan e ano superior a 2018 é:

a) DELETE FROM SELECT Veiculo WHERE Modelo = 'Sedan' AND Ano > 2018;

b) DELETE FROM Veículo WHERE Modelo = 'Sedan' AND Ano > 2018;

c) DELETE SELECT Veiculo FOR Modelo = 'Sedan' AND Ano > 2018;

d) DELETE Veiculo.Modelo = 'Sedan' AND Veiculo.Ano > 2018;

e) SELECT DEL FROM Veiculo HAVING (Modelo, Ano) (= 'Sedan', > 2018);

GABARITO

GABARITO

1. A
2. C
3. C
4. A
5. A
6. B
7. E
8. D
9. A
10. B
11. E
12. C
13. C
14. C
15. C
16. E
17. A
18. E
19. D
20. B
21. E
22. B

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.