Estratégia Concursos

Aula 03

Banco do Brasil (Escriturário - Agente de Tecnologia) Banco de Dados - 2023 (Pós-Edital)

Autor:
Thiago Rodrigues Cavalcanti

07 de Janeiro de 2023

Índice

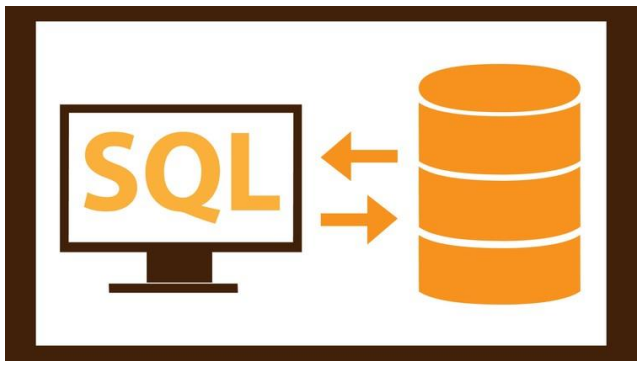
INTRODUÇÃO A SQL

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.

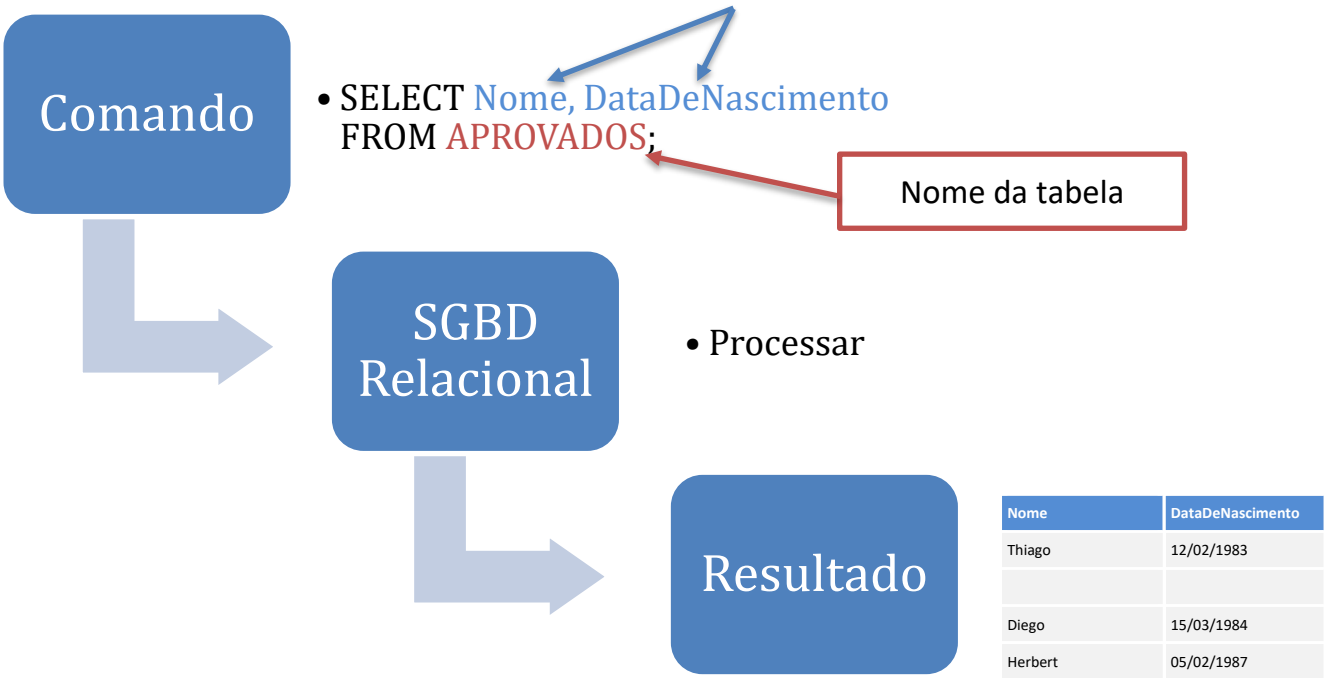


Figura 1 - Um exemplo de consulta

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.

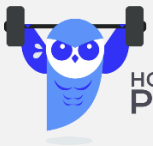
SUBLINGUAGENS DE SQL

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.

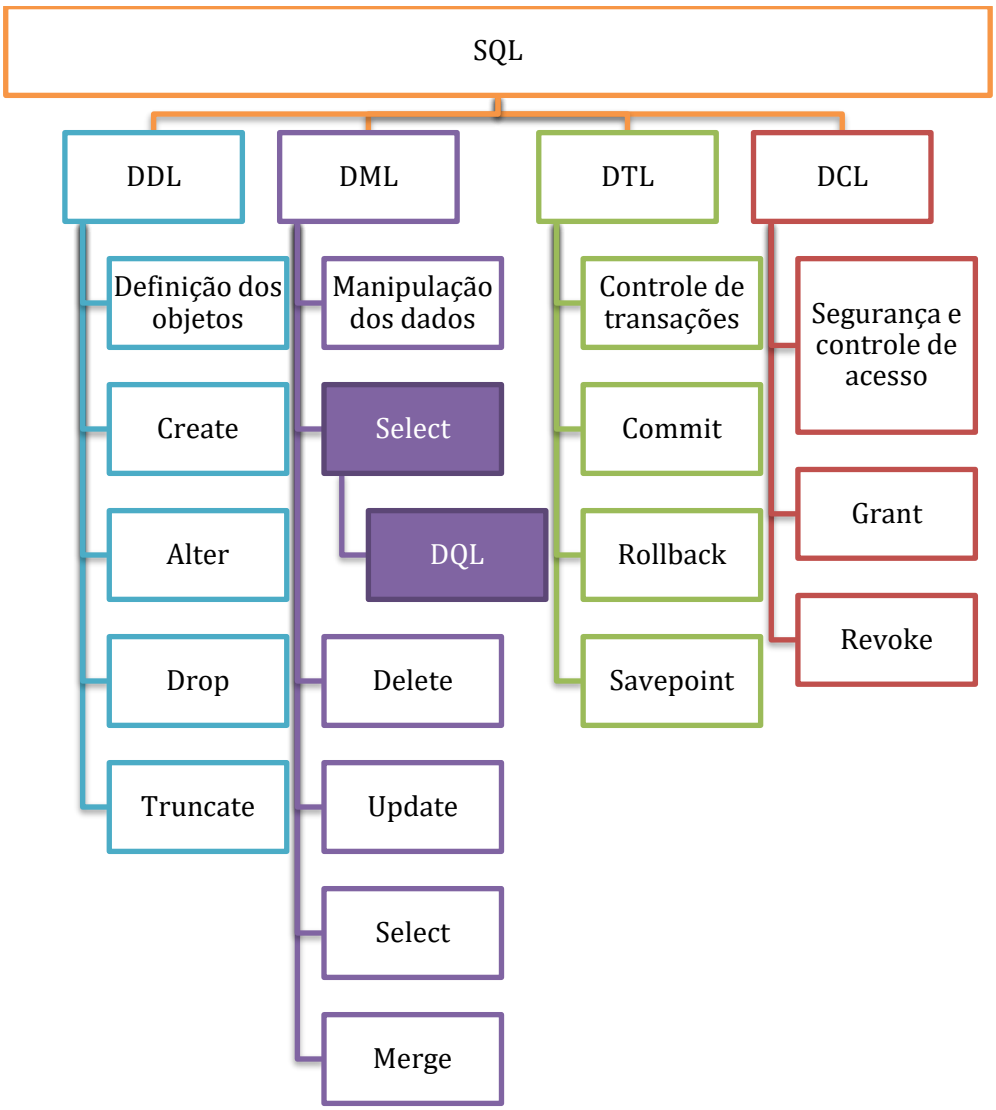


Figura 2 - Sublinguagem de SQL

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*).

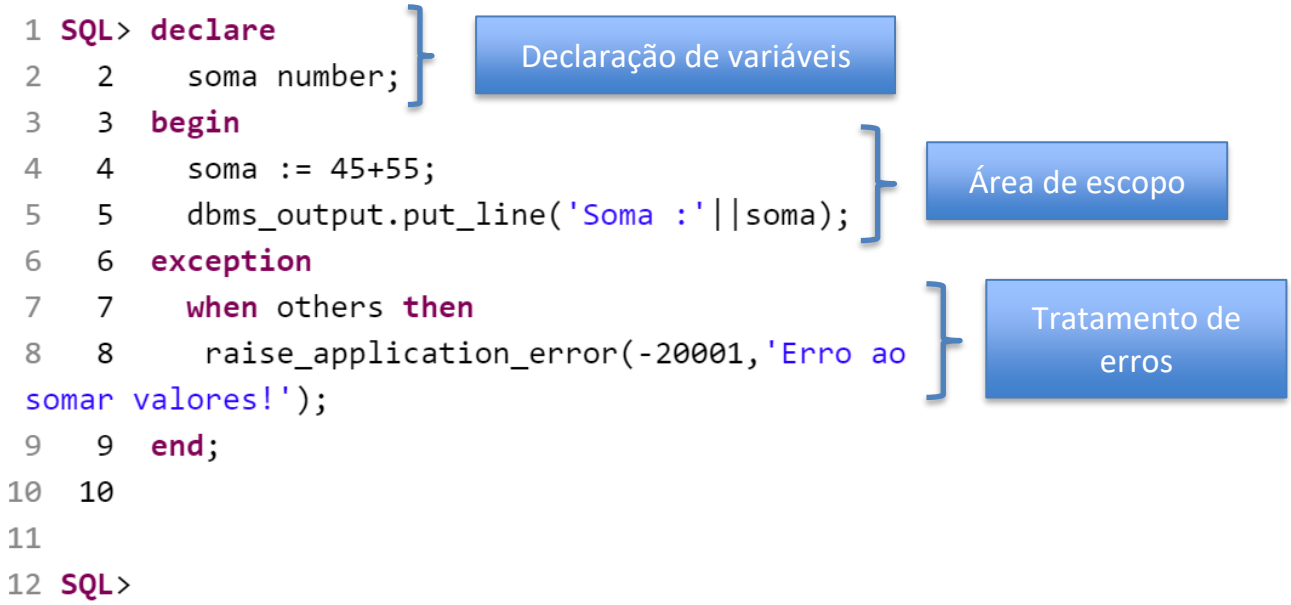


Figura 3 - Exemplo de código PL/SQL

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.

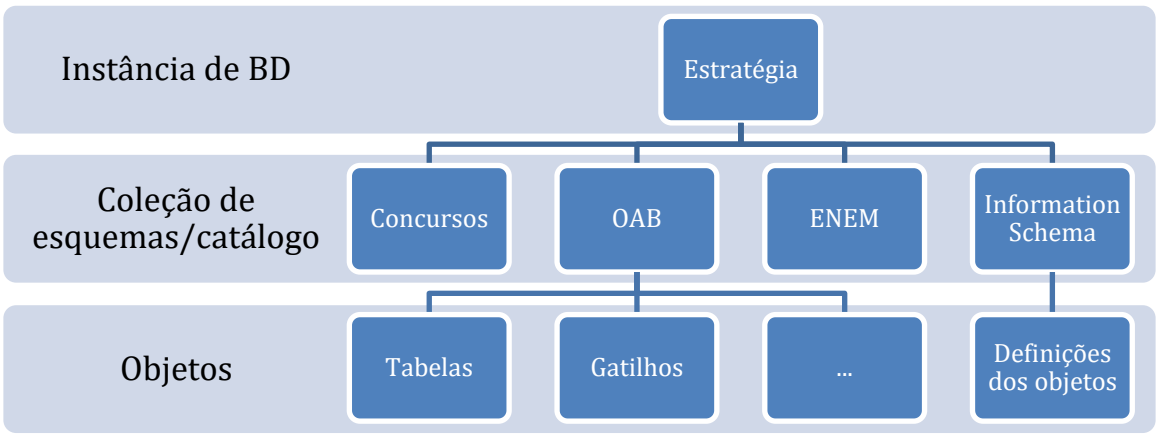


Figura 1- Hierarquia entre instâncias, esquemas e objetos de banco de dados.

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;
```

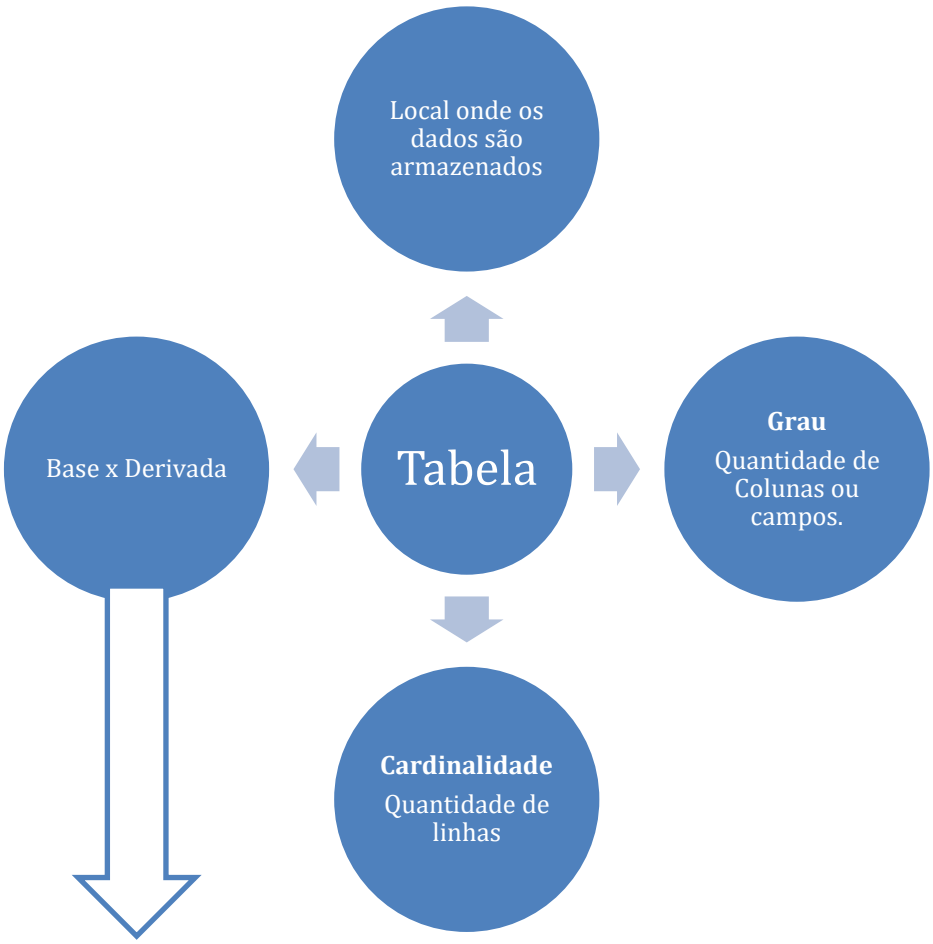
Depois de criar um esquema é possível removê-lo. Deve-se utilizar o comando **DROP SCHEMA**, seguido pelo nome do esquema a ser excluído. Podendo, ainda, utilizar as opções **RESTRICT** e **CASCADE** após o nome do esquema. Se a opção **CASCADE** for especificada, todos os objetos de esquema e de dados 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:

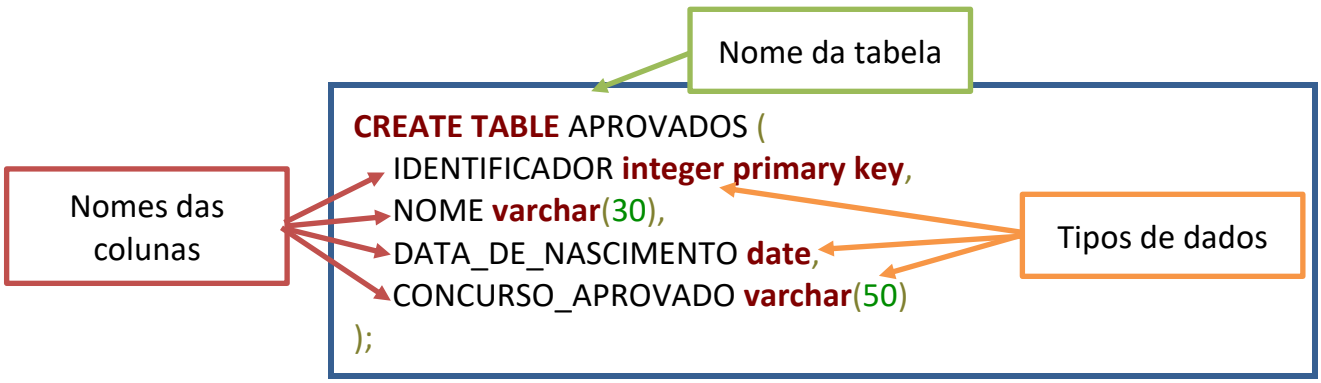


Figura 2 - Criando a tabela aprovados.

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

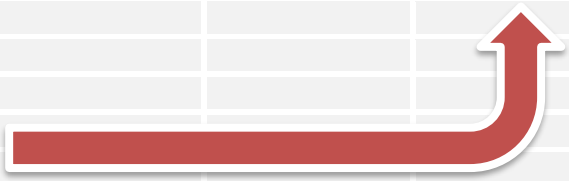
```
ALTER TABLE NOME_DA_TABELA ADD [COLUMN] NOME_COLUNA1 TIPO_DE_DADO;
ALTER TABLE NOME_DA_TABELA DROP [COLUMN] NOME_COLUNA1 {CASCADE|RESTRICT};
ALTER TABLE NOME_DA_TABELA ALTER COLUMN NOME_COLUNA1 TIPO_DE_DADO {SET DEFAULT VALOR_PADRAO|DROP
DEFAULT};
```

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

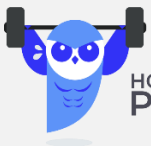
NOME	CPF	EMAIL	DATA_NASCIMENTO	CIDADE	VALOR_PAGO



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.

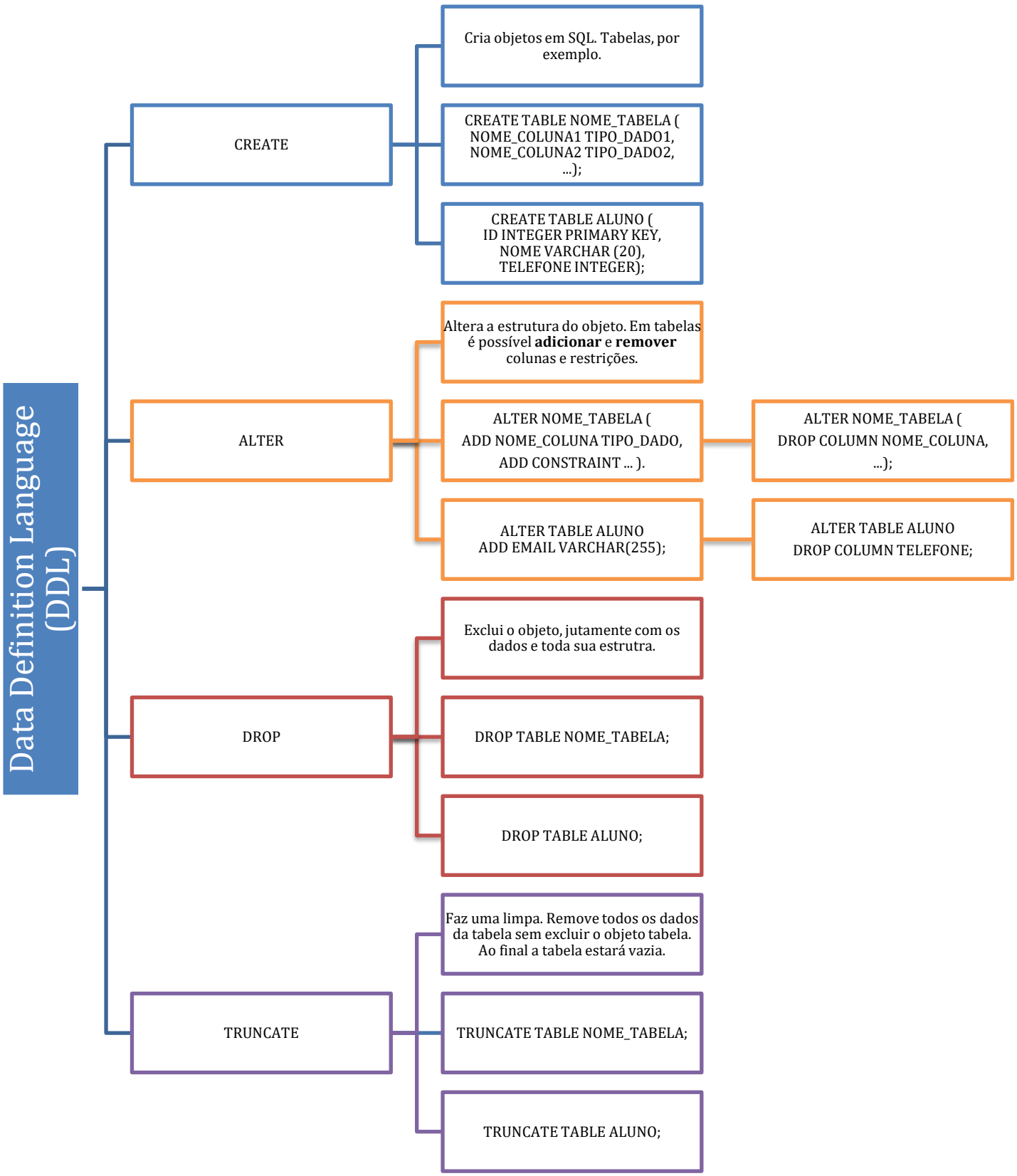


Figura 3 - Resumo dos comandos DDL.

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
);
```

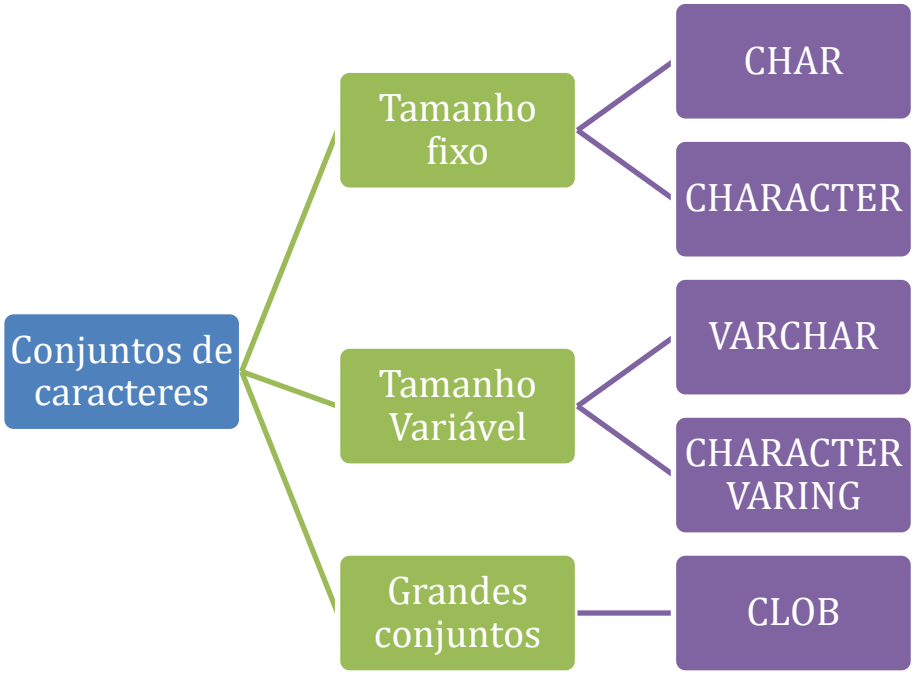


Figura 1 - Tipos de dados textuais

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.

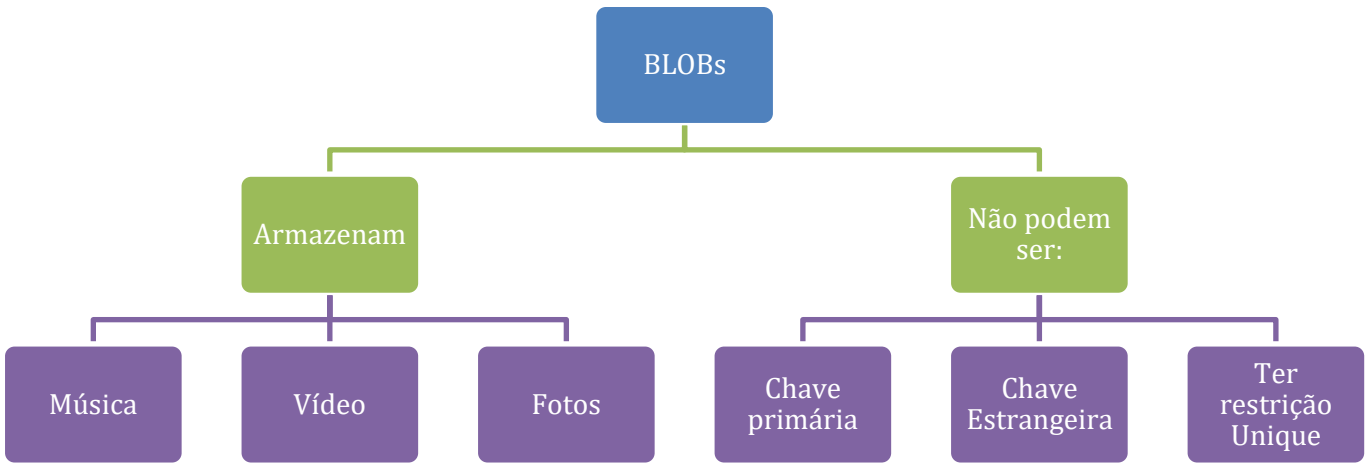


Figura 2 - Esquematizando os BLOBs

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.

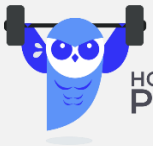
Deu para entender? 😊

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.

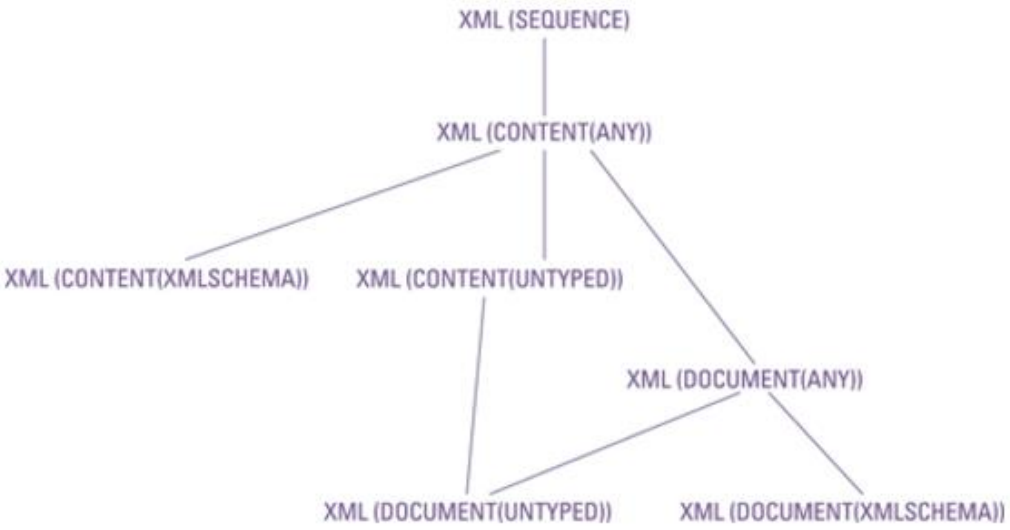
```
CREATE TABLE T1(
Col1 int primary key,
Col2 xml);
```

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);
```

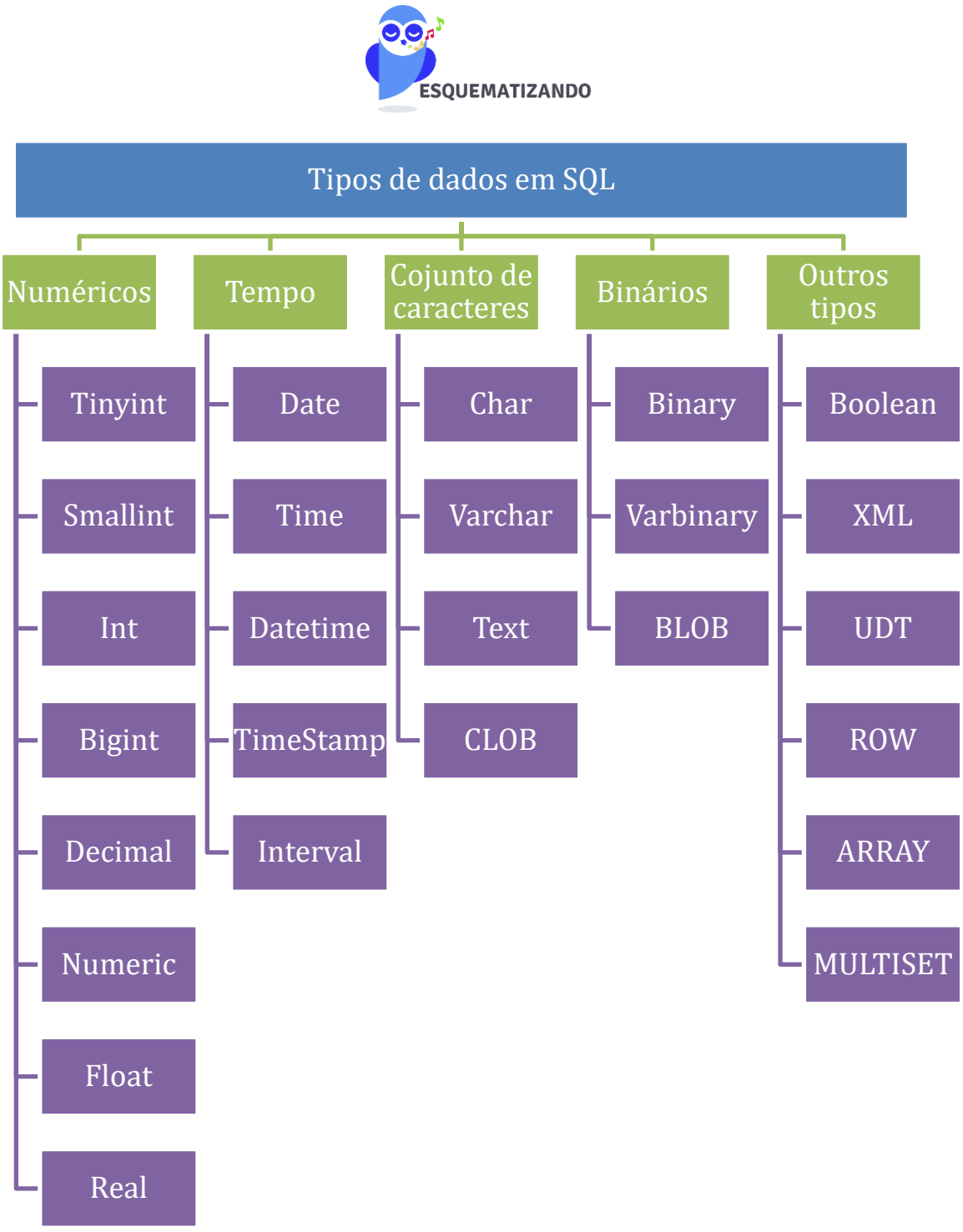


Figura 3 - Resumo dos tipos de dados de SQL

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.

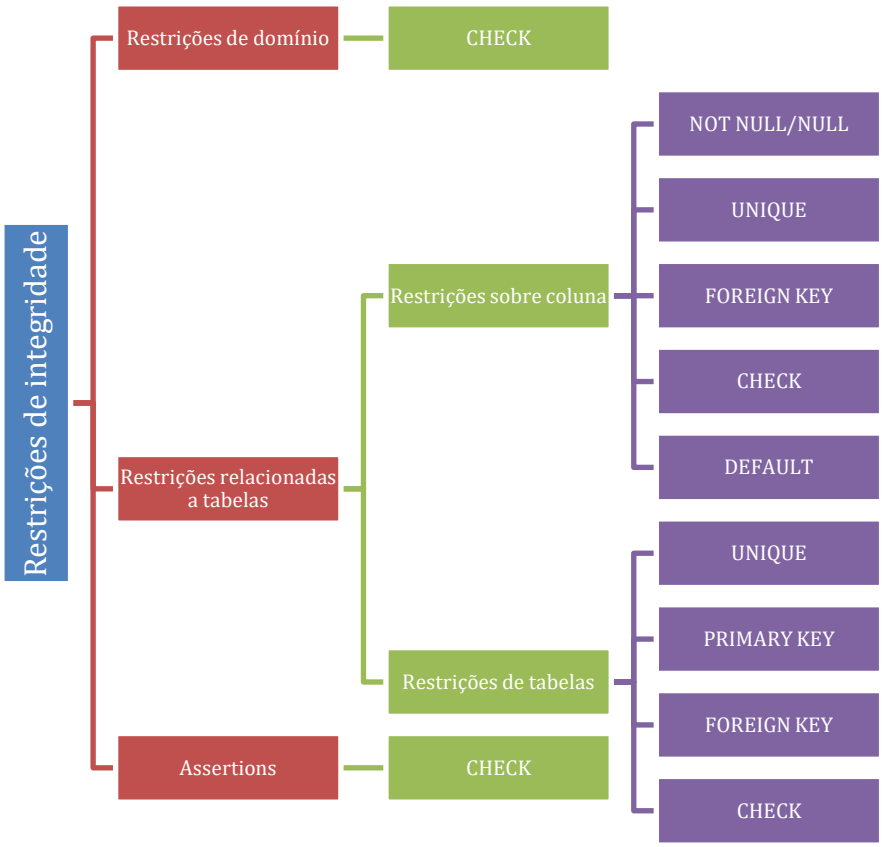


Figura 1 – Tipos de restrições de integridade

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:

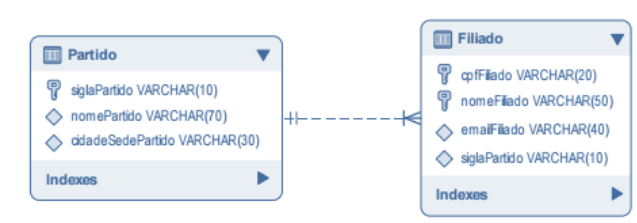
[CONSTRAINT <nome_da_restrição>] CHECK (<search condition>).

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.

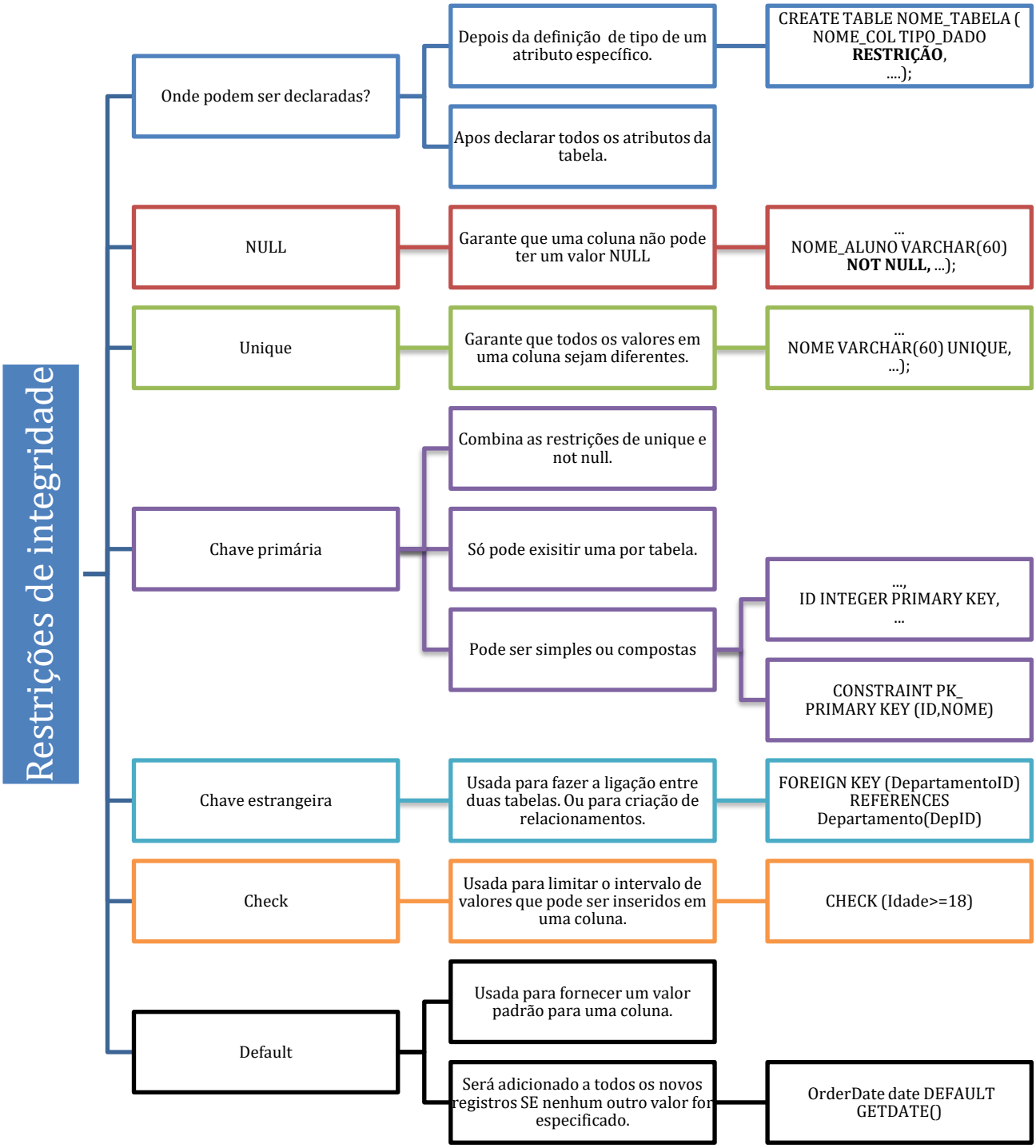


Figura 2 – Exemplos de restrições de integridade em SQL

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.

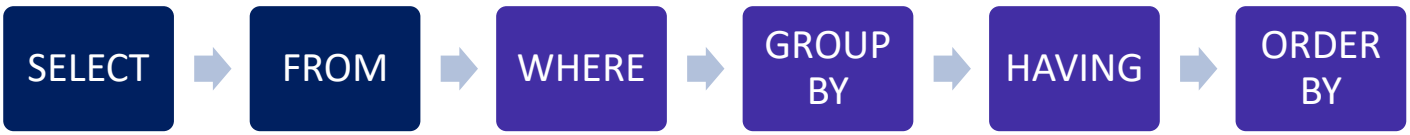

```
SELECT [ DISTINCT | ALL ] { * | <select list> }
FROM <table reference> [ { , <table reference> } . . . ]
[ WHERE <search condition> ]
[ GROUP BY <grouping specification> ]
[ HAVING <search condition> ]
[ ORDER BY <order condition> ]
```

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 ;
```

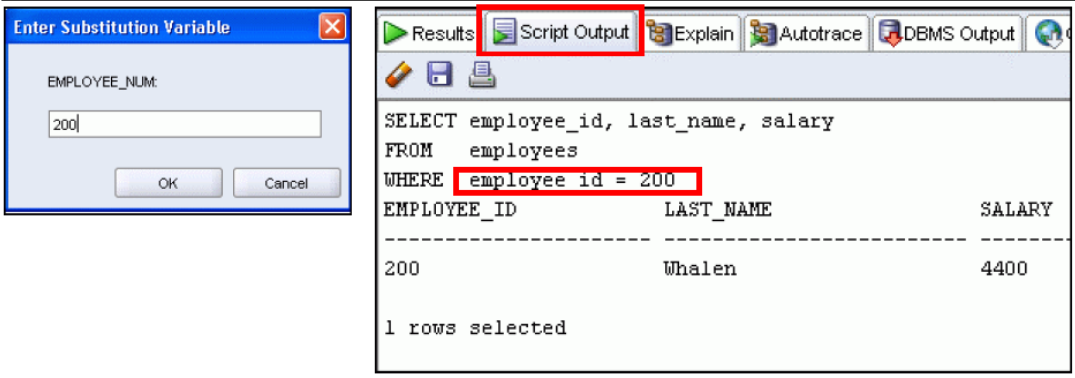
```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```

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;
```



HORA DE PRATICAR!

(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

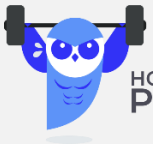
OPERADOR	SIGNIFICADO
1	Operadores aritméticos
2	Operadores de concatenação
3	Operadores condicionais
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	Operador lógico NOT
8	Operador lógico AND
9	Operador lógico OR

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 =
                (SELECT MIN(salary)
                 FROM   employees);
```

2500

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:

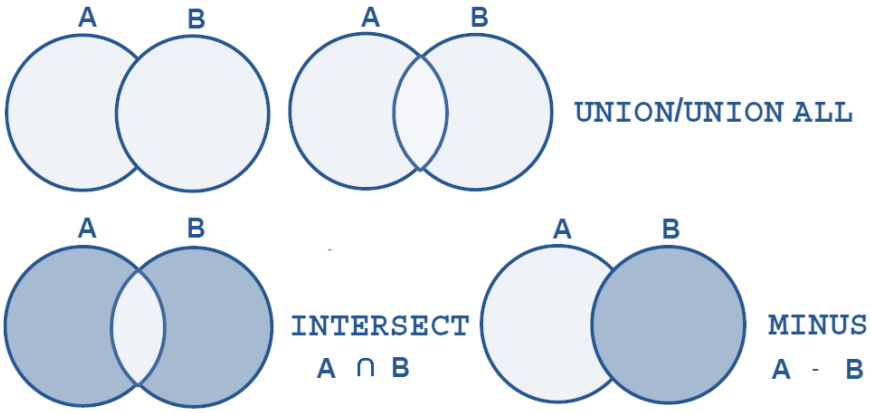
(A) SELECT sexo="FEMININO" FROM TRAB WHERE AVG(salario);
(B) SELECT sexo, AVG(salario) as MediaSalarial FROM TRAB GROUP BY sexo;
(C) SELECT AVG(salario) FROM TRAB WHERE sexo='FEMININO';
(D) SELECT sexo, AVG(salario) FROM TRAB GROUP BY sexo="FEMININO";
(E) SELECT * FROM TRAB WHERE sexo='FEMININO' as AVG(salario);

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.

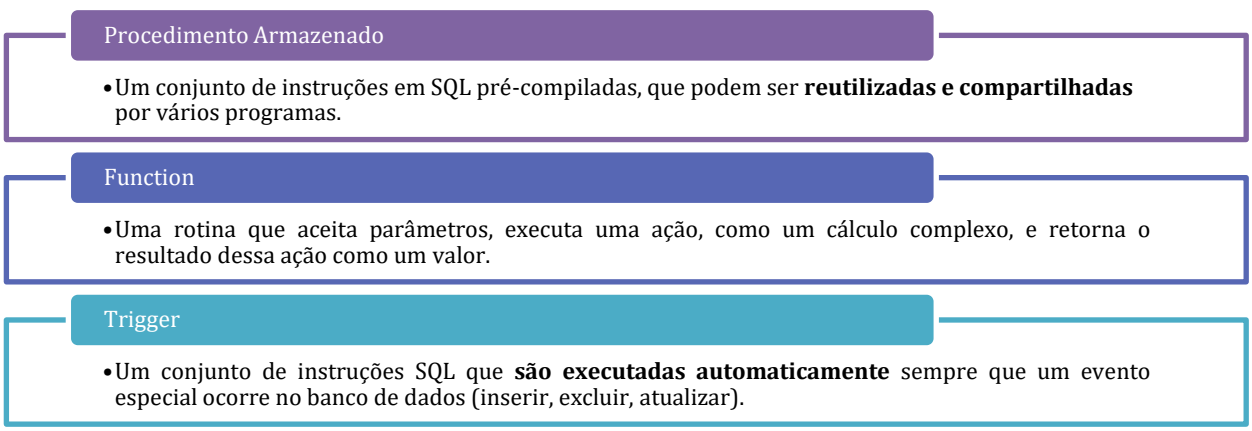
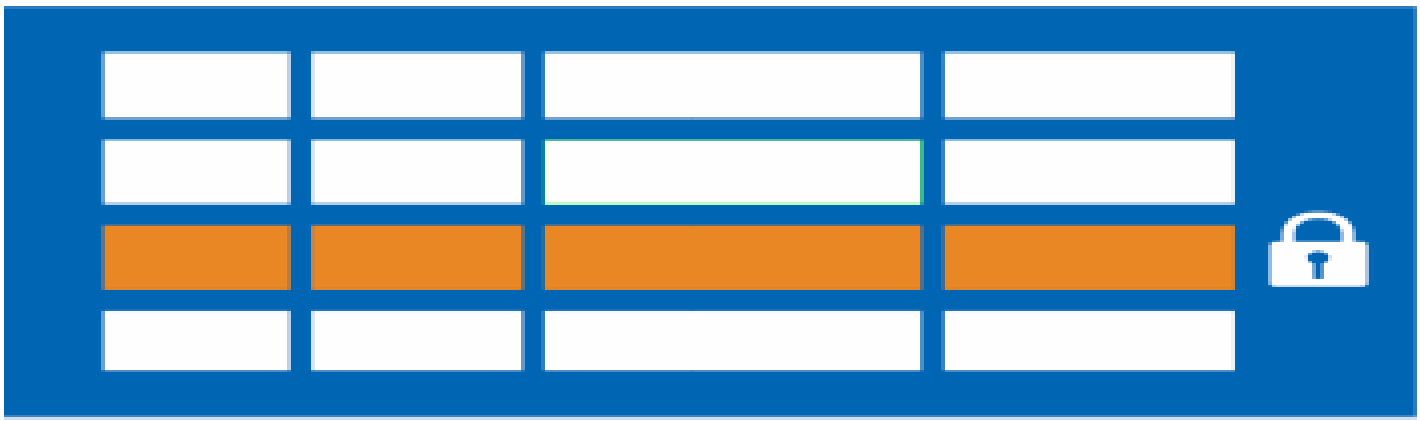


Figura 1 - Extensões procedurais

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.

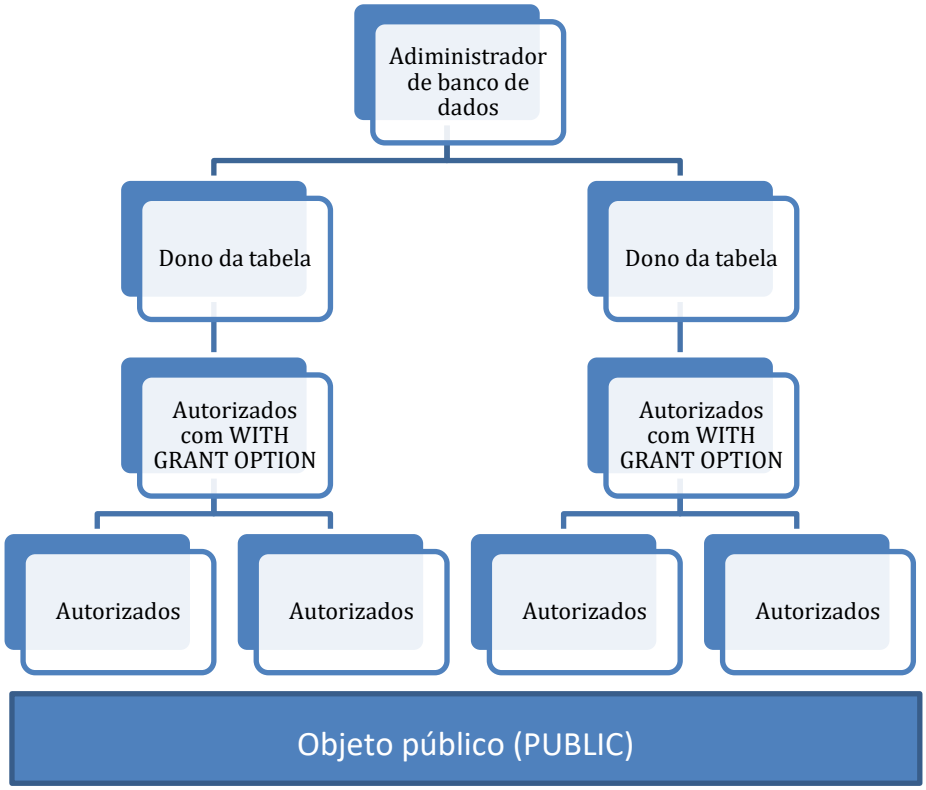


Figura 1 - Hierarquia de privilégios e permissões

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.

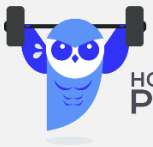
Você pode criar um papel usando uma sintaxe semelhante a seguinte:

```
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.

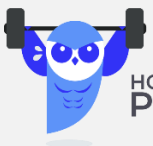
Por exemplo, suponha que você usou a cláusula WITH GRANT OPTION quando concedeu determinados privilégios a um usuário. Eventualmente, quando você deseja revogar os privilégios, você pode usar CASCADE na instrução REVOKE. Quando você revogar os privilégios de um usuário, desta forma, você também retira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

1. 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.

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

2. 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

3. **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;
```

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

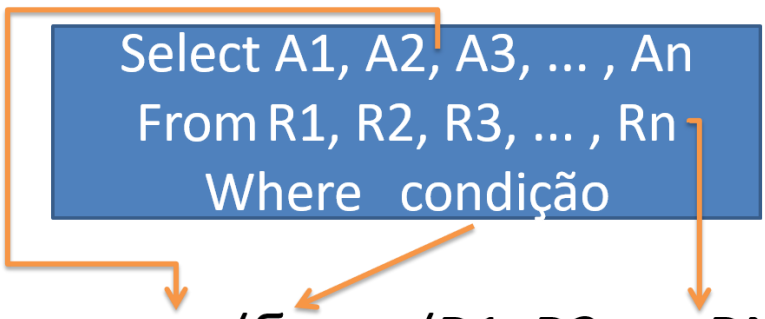
CAIU na prova!

4. 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:



Relembrando:
Π – pi - projeção
б – sigma – seleção
x – produto cartesiano

Assim, podemos observar que a afirmação está **correta**.

Gabarito: C

CAIU na prova!

5. 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 caracteres ou palavra-chave de ESCAPE. Desta forma a alternativa está incorreta.

Gabarito: C C C E

6. 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.

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

64 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 dever 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

7. 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 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

8. 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 respeitos 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 advinda 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

9. 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 transforma 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 classifica-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

10.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;

B 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.

C A alternativa C é a nossa reposta. 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.

D 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

CAIU na prova!

11. 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.

91 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.

92 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

CAIU na prova!

12. 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.

CAIU na prova!

13. **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.

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

14. **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 associa-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 de chave primária não faz o menor sentido.

Gabarito: B

15.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	

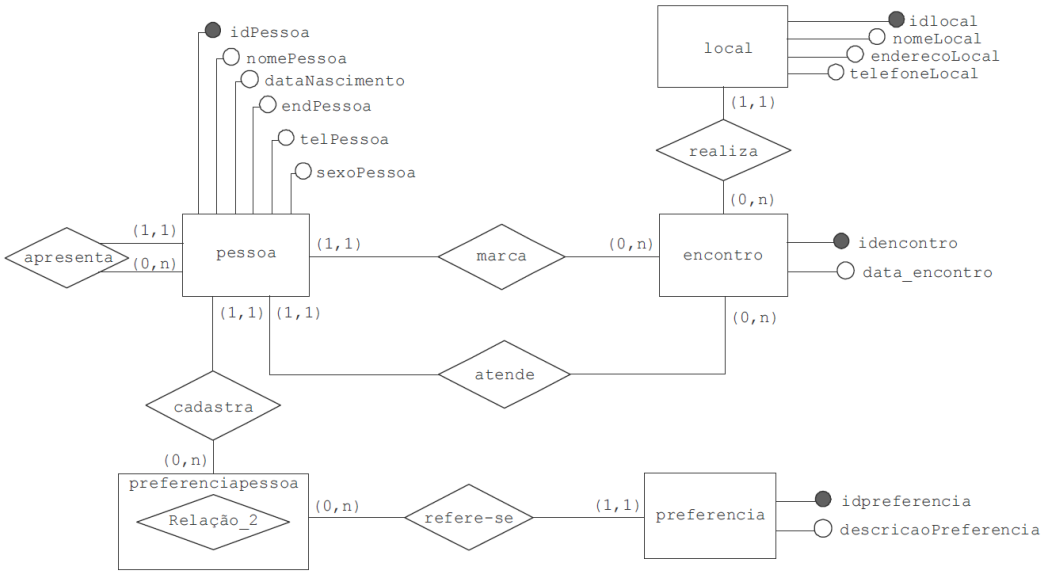
Considerando as tabelas t1 e t2, assinale a opção que apresenta a expressão SQL que gera o conteúdo da tabela resultado.

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;

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)

CAIU na prova!

16.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'));

Comentário: Vamos tentar partir do digrama. 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 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 fazer o relacionamento com pessoa, para isso precisamos utilizar a sintaxe correta da criação de CONSTRAINT.

Gabarito: C.

17. 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

18. **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 da 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

CAIU na prova!

19. **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

CAIU na prova!

20. **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:

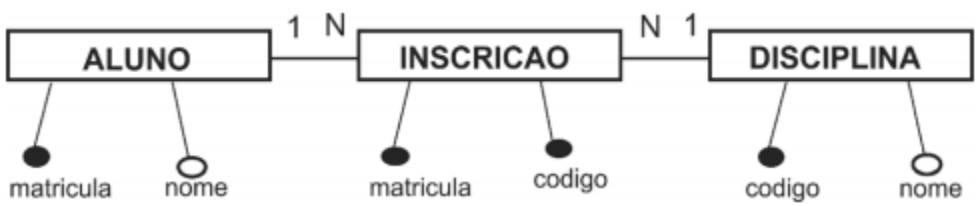
```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

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

CAIU na prova!

21.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

22.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 registros de uma tabela, no caso usamos para tal finalidade o comando DELETE FROM tabela WHERE restrição; que pode ser verificado na alternativa E.

Gabarito: E

QUESTÕES COMENTADAS CESGRANRIO

1. 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

2. 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

3. 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

4. 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

5. 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

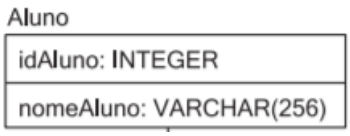
d) 1, 2, 3 e 9

e) 4, 5, 6 e 8

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

6. 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")

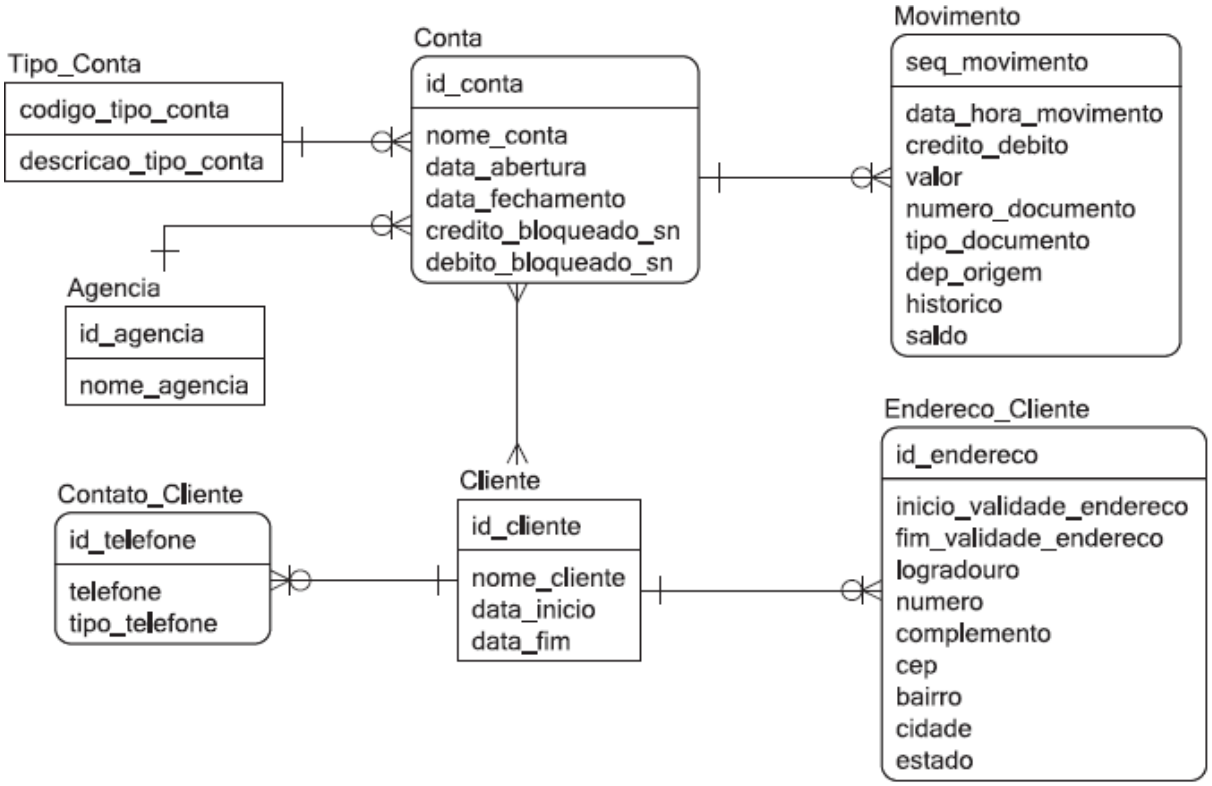
Comentário: Mas uma questão que trata do comando INSERT. Lembra-se que a sintaxe correta é INSERT INTO nome_tabela [(nome_coluna, ...)] VALUES (parâmetro1,...). Assim, podemos encontrar a sintaxe correta apenas na alternativa B.

Gabarito: B

CAIU na prova!

7. 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));

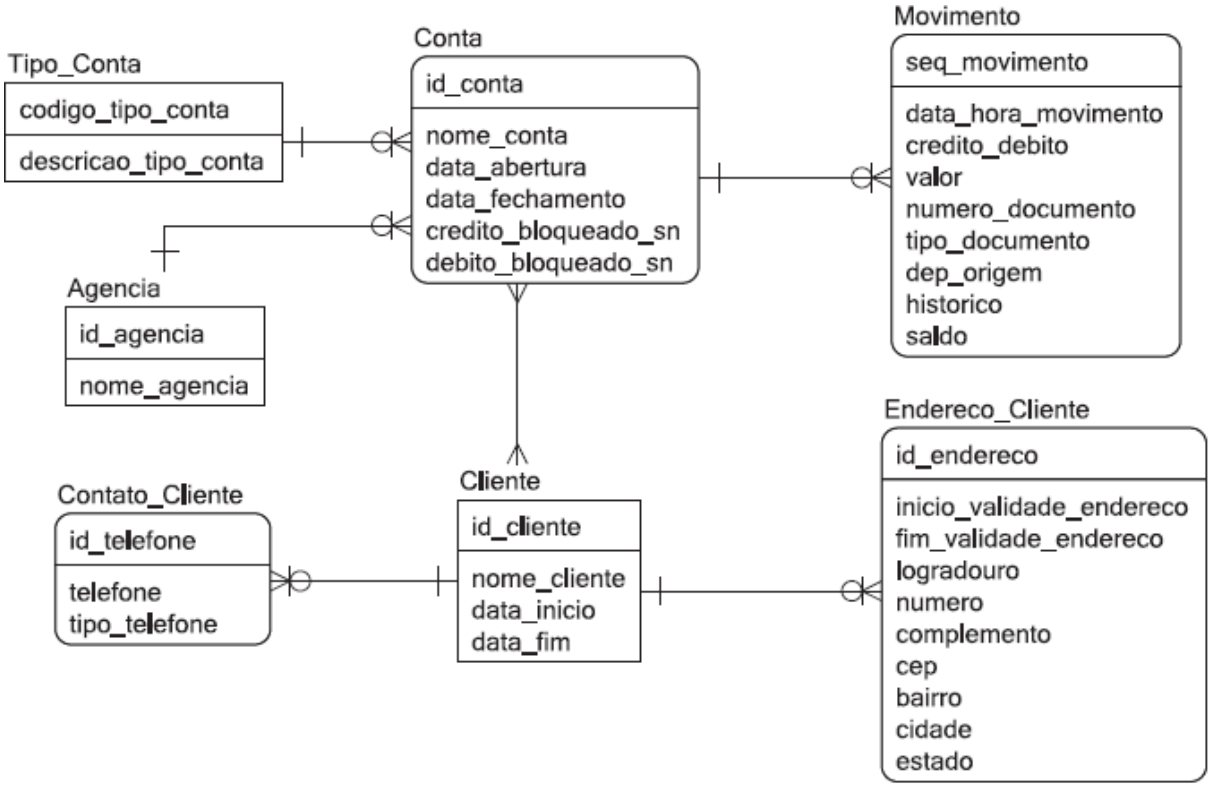
Tal comando está presente na alternativa a) que é a nossa resposta.

Gabarito: A

CAIU na prova!

8. 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

CAIU na prova!

9. 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.

CAIU na prova!

10. 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.

11. 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

12. 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.

CAIU na prova!

13. 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'

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

CAIU na prova!

14. 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.

15. 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

16. 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

17. 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

18. 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

19. 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

20. 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

21. 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

CAIU na prova!

22. 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.

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.

CAIU na prova!

23. 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

24. 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

25. 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

LISTA DE QUESTÕES – CEBRASPE (CESPE)

1. 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.

2. 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.

3. **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;
```

4. 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.

5. 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.

6. 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.

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

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

7. 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 sobrenome
```

8. 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.

9. 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.

CAIU na prova!

10.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
```

11.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.

91 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.

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

12.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.

13. **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.

14. **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.

15.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

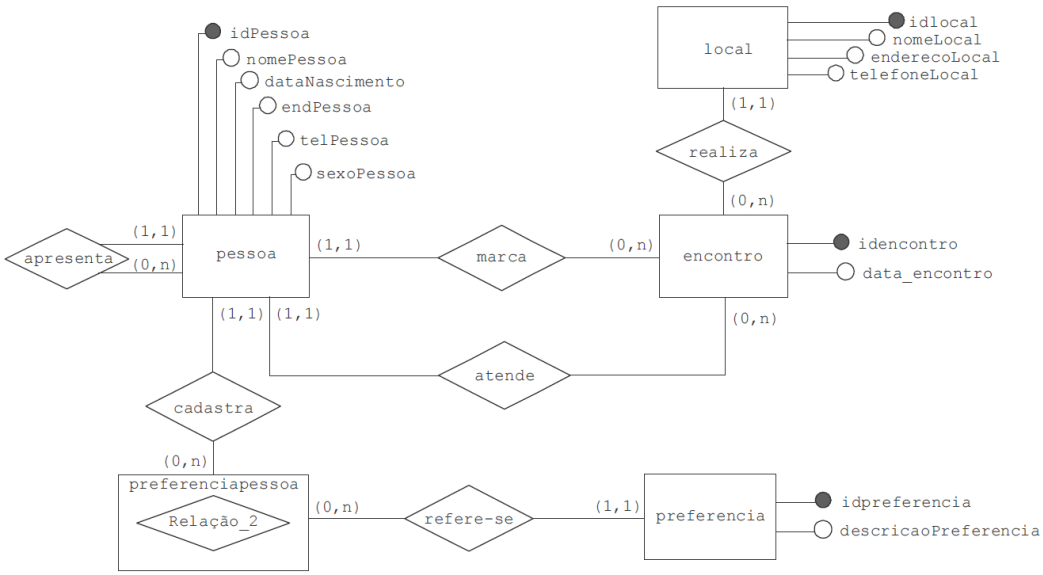
from t1 left join t2 on t1.codigo=t2.codigo

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)

16. 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'));

17. 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);

18. **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.

19. **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.

20. **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.

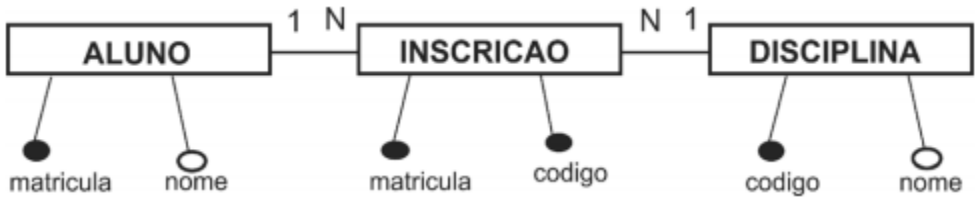
C alter table.

D update.

E insert.

CAIU na prova!

21.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

22.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

1. C E C
2. C
3. E
4. C
5. C C C E
6. C C
7. E C
8. C E C
9. A
10. C
11. E C
12. C
13. C
14. B
15. B
16. C
17. A
18. C
19. C
20. C
21. A
22. E

LISTA DE QUESTÕES – CESGRANRIO

CAIU na prova!

1. 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)

2. 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) ;

3. 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'

4. 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

5. 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

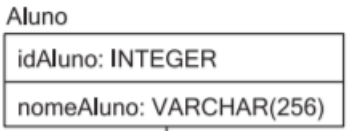
b) 4, 5 e 8

c) 3, 7 e 9

d) 1, 2, 3 e 9

e) 4, 5, 6 e 8

6. 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

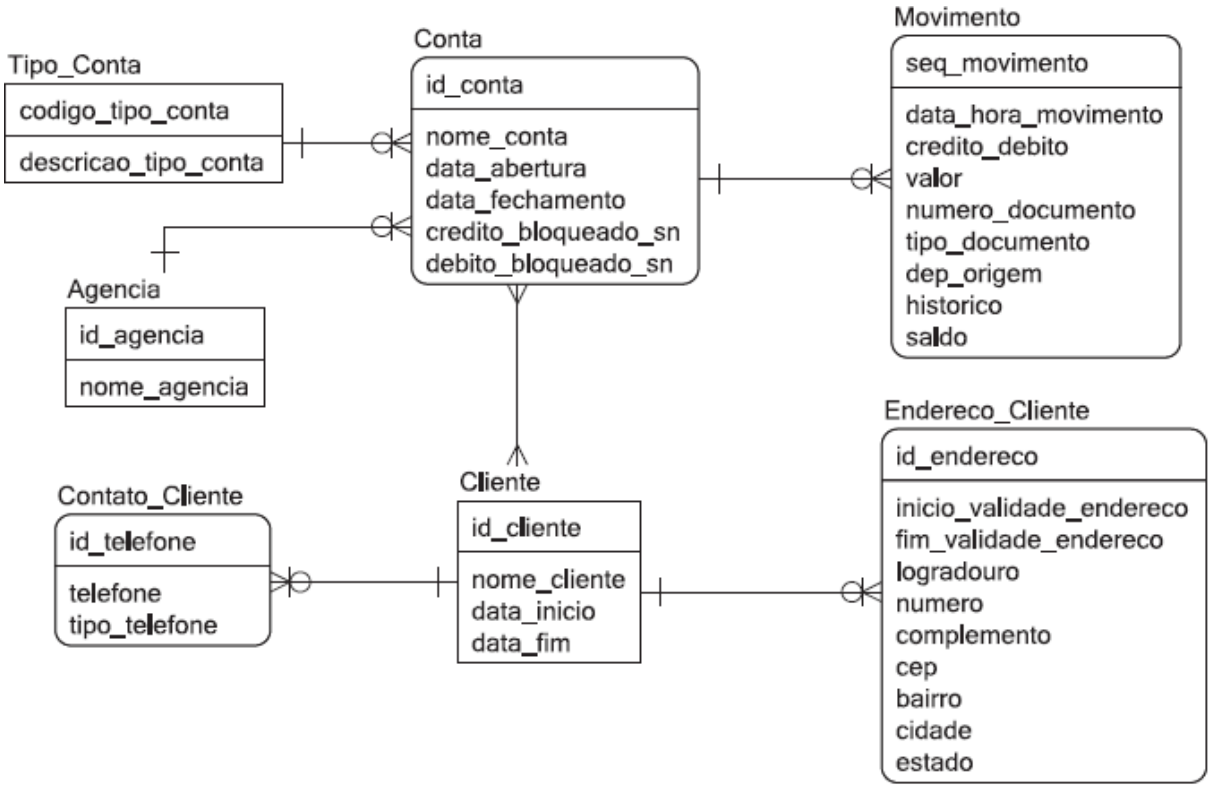
d) UPDATE Aluno SET nomeAluno="Aline" WHERE idAluno=1

e) UPDATE Aluno(idAluno, nomeAluno) SET VALUES (1,"Aline")

CAIU na prova!

7. 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))

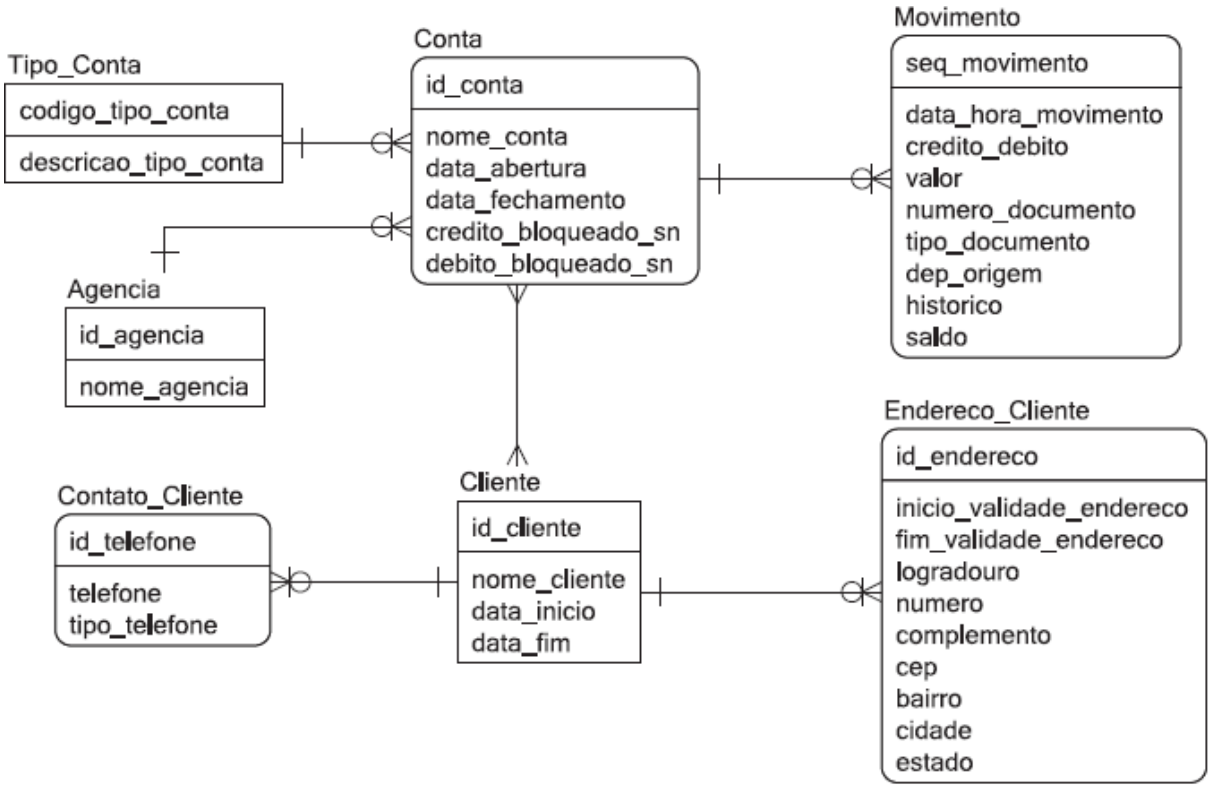
d) CREATE Tipo_Conta AS TABLE (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

e) CREATE Tipo_Conta AS TABLE WITH COLUMNS (codigo_tipo_conta NUMERIC PRIMARY KEY, descricao_tipo_conta VARCHAR(256))

CAIU na prova!

8. 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")

9. 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.

10. 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)

11. 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

CAIU na prova!

12. 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

CAIU na prova!

13. 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'

CAIU na prova!

14. 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

CAIU na prova!

15. 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
```

CAIU na prova!

16. 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)

CAIU na prova!

17. 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

CAIU na prova!

18. 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

CAIU na prova!

19. 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)

CAIU na prova!

20. 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);

21. 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;

22. 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.

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)
```

CAIU na prova!

23. 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
```

24. 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

25. 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. C
2. C
3. D
4. B
5. B
6. B
7. A
8. B
9. D
10. E
11. D
12. B
13. B
14. E
15. B
16. A
17. D
18. E
19. B
20. B
21. B
22. A
23. E
24. D
25. A

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.