1)Os comandos DTL são responsáveis por gerenciar diferentes transações ocorridas dentro de um BANCO DE DADOS. Ele é dividido em três comandos, quais sejam:

A)BEGIN TRAN, COMMIT e ROLLBACK.

B)BEGIN DEVTRAN, DEFINE e BACK.

C)BEGIN LIBRARY, FIND e ROLLBACK.

D)BEGIN, DEFINE LIBRARY e SCROLL.

E)TRAN, FIND e FINISH.

RESPONDER

O subgrupo DTL (Data Transaction Language ou Linguagem de Transação de Dados) da sql é composto pelos comandos:

COMMIT - Instrução SQL usada para finalizar uma transação e tornar permanentes o resultado de uma query de modificação, inclusão ou alteração de dados.

ROLLBACK - Instrução SQL usada para desfazer o resultado de uma transação de uma query de modificação, inclusão ou alteração de dados.

BEGIN - Instrução SQL usada para indicar um bloco de transação, ou seja, indica onde a transação começa ou qual parte de uma query será controlada por transação.

A resposta CORRETA LETRA A, lembrando que o comando BEGIN sofre variações dependendo do fabricante de BANCO DE DADOS.

**Oracle** - Usa-se BEGIN ... END;

**Postgres** - Usa-se BEGIN [WORK,TRANSACTION] as palavras WORK

e TRANSACTION são opcionais e sem efeito.

**MySQL**/MariaDB - BEGIN [NOT ATOMIC] ... END; as palavras NOT ATOMIC são usadas quando estão fora de uma stored procedure.

**SQL Server** - BEGIN [TRAN, TRANSACTION]

**Informix** - BEGIN [WORK] a palavra WORK é opcional.

**DB2** - Usa-se BEGIN ... END;

***A linguagem SQL é subdivida em conjuntos de acordo com a operação que se deseja executar. Os subconjuntos principais são DML, DDL, DCL, DTL e DQL.***

2)Em uma tabela chamada Contribuinte de um banco de dados padrão SQL aberto e em condições ideais há o campo idContribuinte do tipo inteiro e chave primária. Há também o campo nomeContribuinte que é do tipo varchar. Nessa tabela, um Auditor Fiscal deseja alterar o nome do contribuinte de id 1 para 'Marcos Silva'. Para isso, terá que utilizar o comando:

A)ALTER TABLE Contribuinte SET nomeContribuinte='Marcos Silva' WHERE idContribuinte=1;

B)UPDATE Contribuinte SET nomeContribuinte='Marcos Silva' WHERE idContribuinte=1;

C)UPDATE nomeContribuinte TO 'Marcos Silva' FROM Contribuinte WHERE idContribuinte=1;

D) ALTER TABLE Contribuinte FIELD nomeContribuinte='Marcos Silva' WHERE idContribuinte=1;

E)UPDATE TABLE Contribuinte FIELD nomeContribuinte='Marcos Silva' WHERE idContribuinte=1;

RESPONDER

No enunciado do exercício foi dito que em uma tabela o usuário do BANCO DE DADOS deseja atualizar os dados de uma tupla(registro ou linha da tabela), ou seja, será usado usado o comando DML(Data Manipulation Language) UPDATE.

Como a instrução ALTER TABLE é do subgrupo DDL(Data Definition Language) da linguagem SQL, as alternativas A e D não podem ser descartadas.

Analisando-se as alternativas B, C e D sabemos que a alternativa incorreta é a **LETRA A** é a correta, pois a Letra E e C estão errada visto que a sintaxe do comando UPDATE não admite as palavras TABLE, FIELD, FROM e TO. A sintaxe correta da instrução UPDATE é:

UPDATE [nome da tabela] SET [campo=valor] WHERE [condições]; ***As instruções DML(Data Manipulation Language ou Linguagem de Manipulação de dados) são INSERT, UDPATE e DELETE, sendo que a instrução SELECT se enquadra como DQL(Data Query Language ou Linguagem de Consulta de Dados) .***

3)A expressão em SQL padrão descrita no quadro abaixo realiza uma consulta em um BANCO DE DADOS que contém, pelo menos, duas tabelas, sendo uma denominada FUNCIONARIO e outra denominada DEPENDENTE, que armazenam informações sobre funcionários de uma empresa e seus dependentes, respectivamente.

A tabela FUNCIONARIO possui pelo menos três atributos: nome\_func, sobrenome\_func e cpf\_func, que armazenam, respectivamente, o nome, o sobrenome e o número de CPF dos funcionários.

A tabela DEPENDENTE possui pelo menos quatro atributos: nome\_depen, sobrenome\_depen e cpf\_depen, que armazenam, respectivamente, o nome, o sobrenome e o número de CPF dos dependentes, e cpf\_func\_resp, que armazena o número do CPF do funcionário responsável pelos dependentes.

SELECT nome\_func, sobrenome\_func  
 FROM FUNCIONARIO  
 WHERE NOT EXISTS (   
SELECT \* FROM DEPENDENTE  
 WHERE cpf\_func = cpf\_func\_resp  
);

Com base nas informações apresentadas, a consulta em SQL padrão acima descrita recupera :

A)o nome e o sobrenome de funcionários que não têm dependentes. B)o nome e o sobrenome de todos os dependentes dos funcionários. C)o nome e o sobrenome de funcionários que têm dependentes. D)todos os atributos de funcionários que não têm dependentes. E)todos os atributos de todos os dependentes dos funcionários. RESPONDER

Para solucionar supomos que as tabelas FUNCIONARIO e DEPENDENTE estão preenchidas conforme ilustrado abaixo:

Tabela FUNCIONARIO

|  |  |  |
| --- | --- | --- |
| **nome\_func** | **sobrenome\_func** | **cpf\_func** |
| João | Alemeida | 111.111.111-11 |
| Pedro | Silva | 111.111.111-12 |
| Patricia | Alves | 111.111.111-13 |
| Pietra | Oliveira | 111.111.111-14 |

Tabela DEPENDENTE

|  |  |  |  |
| --- | --- | --- | --- |
| **nome\_depen** | **sobrenome\_depen** | **cpf\_depen** | **cpf\_func\_resp** |
| Carla | Silva | 222.222.222-21 | 111.111.111-12 |
| Ricardo | Silva | 222.222.222-22 | 111.111.111-12 |
| Emanuel | Silva | 222.222.222-23 | 111.111.111-12 |
| Enrico | Oliveira | 222.222.222-24 | 111.111.111-14 |
| Luna | Oliveira | 222.222.222-25 | 111.111.111-14 |

**CONTINUANDO...**

Apesar de a subquery SELECT \* FROM DEPENDENTE WHERE cpf\_func = cpf\_func\_resp informada no exercício, somente retornar os registros da tabela DEPENDENTE que estejam relacionados com a tabela FUNCIONARIO pela foreign key cpf\_func\_resp, a instrução externa NOT EXISTS é usada somente para retornar os registro que não foram encontrados no subselect.

**Observação:** O comando EXISTS é usado para testar se algum registro é localizado, e sendo utilizado com a instrução NOT a ideia é verificar se um registro não foi localizado na subquery.

Já pela query externa SELECT nome\_func, sobrenome\_func FROM FUNCIONARIO sabemos que o que será encontrado são os dados de nome e sobrenome dos funcionários da tabela FUNCIONARIO, podendo-se eliminar as alternativas B, D e E.

Por fim, entre as alternativas A e C, chega-se a resolução **alternativa A**, já que as restrições constantes na cláusula WHERE, conforme explicado acima e continuando na ilustração abaixo, irão retornar os dados de funcionários que não possuem dependentes.

Tabela FUNCIONARIO

|  |  |
| --- | --- |
| **nome\_func** | **sobrenome\_func** |
| João | Alemeida |
| Patricia | Alves |

***Na linguagem SQL uma subconsulta, subquery ou subselect é uma instrução SELECT utilizada dentro de outra instrução SELECT. Um subselect pode retornar somente um valor ou um conjunto de valores que serão usados por alguma cláusula do SELECT mais externo.***

4)Julgue os próximos itens, no que diz respeito a comandos SQL de consulta.

I Subqueries de múltiplas colunas retornam várias colunas em uma mesma linha.

II A função UNION apresenta as linhas que existam simultaneamente em duas ou mais tabelas.

III A função EXISTS garante que o resultado de uma subquery somente seja mostrado se retornar uma ou mais linhas.

IV A função EXTRACT retorna uma das informações de um campo do tipo data (dia, mês, ano, hora, minuto ou segundo).

Assinale a opção correta.

A)Estão certos apenas os itens I e II.

B)Estão certos apenas os itens I e IV.

C)Estão certos apenas os itens II e III.

D)Estão certos apenas os itens III e IV.

E)Todos os itens estão certos.

RESPONDER

A afirmativa do item I "Subqueries de múltiplas colunas retornam várias colunas em uma mesma linha" está **errada**, pois as subqueries de múltiplas colunas retornam várias colunas em colunas distintas, conforme demonstrado no exemplo abaixo.

Suponha que a consulta SELECT \* FROM tabelaum retorne o resultado da tabela abaixo :

|  |  |
| --- | --- |
| **campo\_um** | **campo\_dois** |
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |
| 4 | 2 |
| 5 | 1 |

Já a consulta SELECT \* FROM tabelaum WHERE (campo\_um, campo\_dois) IN (SELECT campo\_um, campo\_dois FROM tabelaum WHERE campo\_dois IN (1)); é executada corretamente e retorna uma tupla de dados fazendo uma comparação de duas colunas na subquery, conforme o resultado abaixo :

|  |  |
| --- | --- |
| **campo\_um** | **campo\_dois** |
| 1 | 1 |
| 3 | 1 |
| 5 | 1 |

**CONTINUANDO**

A afirmativa do item 2 "A função UNION apresenta as linhas que existam simultaneamente em duas ou mais tabelas." está **errada** pois a instrução UNION retorna a diferença de registros entre duas tabelas mais uma linha por registro que seja igual. O texto item 2 se refere ao comando UNION ALL. Suponha que a consulta SELECT \* FROM tabelaum retorne o resultado da tabela abaixo :

|  |  |
| --- | --- |
| **campo\_um** | **campo\_dois** |
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |

Suponha que a consulta SELECT \* FROM tabeladois retorne o resultado da tabela abaixo :

|  |  |
| --- | --- |
| **campo\_um** | **campo\_dois** |
| 1 | 1 |
| 2 | 2 |
| 4 | 2 |

Com a query select campo\_um, campo\_dois from tabelaum union select campo\_um, campo\_dois from tabeladois é feito o union entre as tabelaum e tabeladois e o resultado é demonstrada abaixo :

|  |  |
| --- | --- |
| **campo\_um** | **campo\_dois** |
| 2 | 2 |
| 1 | 1 |
| 3 | 1 |
| 4 | 2 |

Com a query select campo\_um, campo\_dois from tabelaum union all select campo\_um, campo\_dois from tabeladois é feito a união sem descarte de registros entre as tabelaum e tabeladois e o resultado é demonstrada abaixo :

|  |  |
| --- | --- |
| **campo\_um** | **campo\_dois** |
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |
| 1 | 1 |
| 2 | 2 |
| 4 | 2 |

**Observação :**Perceba que a query que usa UNION retorna 4 linhas enquanto que a query que usa UNION ALL retorna 6 linhas, ou seja, 3 linhas da tabela 1 com as 3 linhas da tabela 2.

**CONTINUANDO**

Já que os itens I e II estão errados, por eliminação sabe-se que as alternativas A, B, C e E não são a resposta, sobrando a **Letra D** como resposta correta.

***Procedimento armazenado ou Stored Procedure é uma coleção de comandos em SQL que encapsula tarefas repetitivas podendo ou não aceitar parâmetros de entrada e retornar dados de saída ou um valor de status (para indicar aceitação ou falha na execução).***

Para calcular o número de produtos com nomes que terminam com a letra B, o Auditor testou os comandos abaixo.

* SELECT COUNT(nomePro)  
  FROM Produto  
  WHERE nomePro BEGIN '%B';
* II. SELECT COUNT(\*)  
  FROM PRODUTO  
  WHERE SUBSTR(nomePro, -1)='B';
* III. SELECT COUNT(\*)  
  FROM Produto  
  WHERE nomePro LIKE '%B';
* IV. SELECT COUNT(nomePro)  
  FROM Produto  
  WHERE SUBSTR(nomePro, 0)='B';
* Mostrará o resultado desejado o que consta APENAS em:

A)III;

B)I e IV;

C)III e IV;

D)I e II;

E)II e III.

RESPONDER

Para iniciar a resolução deste exercício é importante entender que COUNT é uma das funções de agregação disponibilizadas pela linguagem SQL que tem com objetivo contabilizar a quantidade de linhas de um determinado grupo de dados. Entre as alternativas há duas formas corretas de se utilizar a instrução COUNT, o comando COUNT(\*) realizará a contabilização do número de linhas retornado pela query e COUNT(nomePro) realizará a contabilização do números de linhas retornado somente para a coluna nomePro. Supondo que a quantidade de registros seja igual em ambos os casos, COUN(\*) e COUNT(nomePro) terão o mesmo resultado.

A partir desta primeira análise da instrução COUNT ainda não é possível identificar qual a alternativa correta.

**CONTINUANDO**

Verificando a cláusula WHERE das alternativas é possível eliminar a Letra B e D pois o comando BEGIN é um tipo de comando DTL(Data Transaction Language ou Linguagem de Transação de Dados) e no exercício cabe somente comandos DML(Data Manipulation Language ou Linguagem de Manipulação de Dados)

A instrução LIKE é usada em cláusulas WHERE para localizar registros de acordo com um determinado padrão, padrão que é definido por meio dos wildcars '%' e '\_' conforme exemplos abaixo utilizando o wildcard '%':

Encontrar qualquer registro que começa com a letra B.

SELECT coluna FROM tabela WHERE coluna LIKE 'B%'

Encontrar qualquer registro que termina com a letra B.

SELECT coluna FROM tabela WHERE coluna LIKE '%B'

Encontrar qualquer registro que contenha a letra B em qualquer posição. SELECT coluna FROM tabela WHERE coluna LIKE '%B%'

Encontrar qualquer registro que contenha começa com a letra B e termina com a letra C.

SELECT coluna FROM tabela WHERE coluna LIKE 'B%C'

Com base na análise acima da instrução LIKE chega-se a conclusão que o item III está correto.

CONTINUANDO

Por fim, para se chegar a **resposta Letra E** itens II e III corretos, é necessário saber que SUBSTR é uma instrução que tem como finalidade extrair um trecho da string original tendo como parâmetro um início, um tamanho opcional e a string original. Se o tamanho que se deseja extrair não for informado como parâmetro de SUBSTR o comando terá como retorno uma substring a partir do início informado até o final da string.

Para exemplificar suponha a tabela PESSOA abaixo com os campo ID e NOME:

|  |  |
| --- | --- |
| **ID** | **NOME** |
| 1 | Afonso Pereira Najib |
| 2 | Breno da Costa Alves |

Na query SELECT NOME, SUBSTR(NOME, -1) as SUB FROM PESSOA WHERE SUBSTR(NOME, -1)='B' temos como resultado "Afonso Pereria Najib, b" ,enquanto que na query SELECT NOME, SUBSTR(nomePro, 0) as SUB FROM PESSOA WHERE SUBSTR(nomePro, 0)='B' nenhum resultado foi exibido.

Observação: O exercício não especificou qual SGBD foi utilizado para realizar a consulta, mas pode sabe-se que no ORACLE e MySQL a sintaxe é SUBSTR e no SqlServer e PostgreSQL a sintaxe é SUBSTRING.

***Gatilho ou trigger é um recurso presente nos BANCOS DE DADOS é executado sempre que um evento ocorre, como uma modificação de dados de uma tabela ou alteração de estrutura de uma tabela. É muito utilizada para ajudar a manter a consistência dos dados ou para propagar alterações em um determinado dado de uma tabela para outras. Em SQL, para se criar um trigger utiliza-se o comando CREATE TRIGGER, e para removê-lo deve-se usar DROP TRIGGER. Entre os modos de disparo de uma trigger, temos aquela que é disparada após a execução de algum comando sql(AFTER) e temos as que são executadas no lugar da query SQL(INSTEAD OF).***

6)Na linguagem SQL, considerando duas tabelas A e B, para se obter o que está na Tabela A e não está na Tabela B, assim como o que está na Tabela B e não está na Tabela A é necessário usar o comando:

A)outer excluding join.

B)right excluding join.

C)left excluding join.

D)middle excluding join.

RESPONDER

**CONTINUAÇÃO**

Para exemplificar a resolução deste exercício, vamos supor que a tabela A e B possuam somente uma coluna chamada NOME\_BANCO, que armazena nomes de BANCOS DE DADOS, conforme ilustrado abaixo: TabelaA

|  |
| --- |
| **NOME\_BANCO** |
| PostgresSQL |
| MySQL |
| Oracle |

TabelaB

|  |
| --- |
| **NOME\_BANCO** |
| Oracle |
| SQLServer |
| MongoDB |

**CONTINUANDO**

**A Letra A está correta** pois ao utilizar o instrução SQL Outer Excluding Join, o resultado serão todos os registros que estão na tabela B, mas que não estejam na tabela A, e todos os registros que estão na tabela A, mas que não estejam na tabela B.

SQL

SELECT A.NOME\_BANCO, B.NOME\_BANCO

FROM TabelaA as A FULL OUTER JOIN TabelaB as B

on A.NOME\_BANCO = B.NOME\_BANCO WHERE A.NOME\_BANCO

is null or B.NOME\_BANCO is null

Resultado

|  |  |
| --- | --- |
| **NOME\_BANCO** | **NOME\_BANCO** |
| PostgresSQL |  |
| MySQL |  |
|  | SQLServer |
|  | MongoDB |

**Observação:** O Outer Join(não confundir com Outer Excluding Join), tem como resultado todos os registros que estão na tabela A e todos os registros da tabela B.

SQL

SELECT A.NOME\_BANCO, B.NOME\_BANCO  
FROM TabelaA as A FULL OUTER JOIN TabelaB as B on A.NOME\_BANCO = B.NOME\_BANCO

Resultado

|  |  |
| --- | --- |
| **NOME\_BANCO** | **NOME\_BANCO** |
| PostgresSQL |  |
| MySQL |  |
|  |  |
| Oracle | Oracle |
|  | SQLServer |
|  | MongoDB |

**CONTINUAÇÃO**

**A Letra B está errada** pois o comando SQL Right Excluding Join, terá como resultado todos os registros que estão na tabela B e que não estejam na tabela A.

SQL

SELECT A.NOME\_BANCO, B.NOME\_BANCO  
FROM TabelaA as A RIGHT JOIN TabelaB as B on A.NOME\_BANCO = B.NOME\_BANCO  
WHERE a.NOME\_BANCO is null

Resultado

|  |  |
| --- | --- |
| **NOME\_BANCO** | **NOME\_BANCO** |
|  | SQLServer |
|  | MongoDB |

**Observação:**O Right Join(não confundir com Right Excluding Join), tem como resultado os registros que estão na tabela B e os registros da tabela A que são comuns à tabela B.

SQL

SELECT A.NOME\_BANCO, B.NOME\_BANCO FROM TabelaA as A  
RIGHT JOIN TabelaB as B on A.NOME\_BANCO = B.NOME\_BANCO

Resultado

|  |  |
| --- | --- |
| **NOME\_BANCO** | **NOME\_BANCO** |
|  |  |
| Oracle | Oracle |
|  | SQLServer |
|  | MongoDB |

**CONTINUAÇÃO**

**A Letra C está errada** pois o SQL Left Excluding Join retorna como resultado todos os registros que estão na tabela A e que não estejam na tabela B.

SQL

SELECT A.NOME\_BANCO, B.NOME\_BANCO  
FROM TabelaA as A  
LEFT JOIN TabelaB as B on A.NOME\_BANCO = B.NOME\_BANCO  
WHERE B.NOME\_BANCO is null

Resultado

|  |  |
| --- | --- |
| **NOME\_BANCO** | **NOME\_BANCO** |
| PostgresSQL |  |
| MySQL |  |

**Observação:**O Left Join(não confundir com Left Excluding Join), tem como resultado todos os registros que estão na tabela A e os registros da tabela B que são comuns à tabela A.

SQL

SELECT A.NOME\_BANCO, B.NOME\_BANCO  
FROM TabelaA as A  
LEFT JOIN TabelaB as B on A.NOME\_BANCO = B.NOME\_BANCO

Resultado

|  |  |
| --- | --- |
| **NOME\_BANCO** | **NOME\_BANCO** |
| PostgresSQL |  |
| MySQL |  |
|  |  |
| Oracle | Oracle |

**CONTINUAÇÃO**

**A Letra D está errada** pois não foi encontrado nenhuma referência ao comando Middle Excluding Join.

***Na linguagem SQL são operadores lógicos os comandos AND, OR e NOT, sendo que o operador AND indica que duas condições devem ser verdadeiras para se exibir os dados, o operador OR indica que pelo menos uma das n condições deve ser verdadeira para se exibir os dados e o operador NOT realiza a inversão do resultado de uma operação lógica. Essas operações lógicas são as mesmas presentes na montagem de tabelas verdades para análise de proposições lógicas.***

7)Considere a sintaxe SQL básica do comando para a criação de gatilhos:

CREATE TRIGGER ON ...

Nesse comando, as opções corretas do:

A)evento disparador são: Insert, Delete e Update.

B)evento disparador são: Select, Group By e Inner Join.

C)evento disparador são: Commit, Rollback e Savepoint.

D)instante de execução são: First e Last.

E)instante de execução são: One e All.

RESPONDER

O **gatilho(trigger)** é sempre associado a uma tabela e a um ou mais eventos que possam ocorrer na tabela. O gatilho pode ser criado para o **evento de inclusão(INSERT), alteração(UPDATE) e/ou exclusão(DELETE)**, o que faz a **Letra A**, a resposta correta deste exercício.

Veja abaixo a sintaxe de criação de um gatilho SQL utilizando o banco de dados Oracle.

CREATE [OR REPLACE] TRIGGER trigger\_name  
{BEFORE | AFTER } triggering\_event ON table\_name  
[FOR EACH ROW]  
[FOLLOWS | PRECEDES another\_trigger]  
[ENABLE / DISABLE ]  
[WHEN condition]  
DECLARE  
   declaration statements  
BEGIN  
   executable statements  
EXCEPTION  
   exception\_handling statements  
END;

**Observação:** na sintaxe SQL acima, em um caso real de criação de um script SQL o trecho triggering\_event seria substituído por INSERT, UPDATE, e/ou DELETE

As **Letras B, C, D e E** estão erradas pois os únicos eventos disparadores aceitáveis de uma Trigger são o INSERT, UPDATE e DELETE, ou seja, Select, Group By, Inner Join, Commit, Rollback, Savepoint, First, Last One e All não podem ser eventos disparadores de uma trigger.

***Supondo que uma tabela X que possui somente uma linha e uma coluna Y tenha três valores numéricos 25, 50, 32. A query SELECT SUM(Y) FROM X irá retornar o resultado 107 correspondente a soma de todos os valores da coluna Y.***

8)Usando a linguagem SQL, um fiscal escreveu corretamente, em uma consulta, a expressão WHERE Nome-Contribuinte LIKE 'p%o'. Com isso ele obteve nomes de contribuintes como, por exemplo:

A)Paulo ou Pedro;

B)Augusto e Jordão;

C)Paulo ou Paula;

D)Álvaro ou Augusto;

E)Paula ou Patricio.

RESPONDER

Para exemplificar, vamos imaginar uma tabela chamada 'Contribuinte', que possua somente uma coluna chamada 'Nome-Contribuinte', conforme ilustrado abaixo:

|  |
| --- |
| **Nome-Contribuinte** |
| Paulo |
| Pedro |
| Augusto |
| Jordão |
| Paula |
| Álvaro |
| Patricio |

Agora imagine a consulta SQL abaixo para complementar e exemplificar a expressão 'WHERE Nome-Contribuinte LIKE 'p%o'' que foi dada no enunciado da questão.

SELECT \*  
FROM Contribuinte  
WHERE Nome-Contribuinte LIKE 'p%o'

Veja abaixo o resultado da query acima:

|  |
| --- |
| **Nome-Contribuinte** |
| Paulo |
| Pedro |
| Patricio |

De acordo com SQL acima e seu resultado, serão localizados todos os registros da tabela cujo nome dos contribuintes começam com a letra 'p', contenham qualquer variação de letras e números no meio da palavra e terminam com a letra 'o', ou seja, os nomes Paulo, Pedro e Patrício serão localizados, o que torna a **letra A** a alternativa correta, visto que é única opção com registros válidos.

**Observação:** Para fins didáticos, ignora-se a questão das letras serem maiúsculas ou minúsculas.

A **letra B** está errada pois o nome Augusto começa com a letra ‘A’ e Jordão começa com a letra ‘J’, ou seja, os nomes Augusto e Jordão atendem a regra Nome-Contribuinte LIKE 'p%o' somente parcialmente, visto que não começam com a letra ‘p’, mas terminam com a letra ‘o’. A **letra C** está errada pois apesar do nome Paulo atender a regra Nome-Contribuinte LIKE 'p%o', o nome Paula termina com a letra ‘a’, e não com a letra ‘o’, conforme solicitado.

A **letra D** está errada pois os nomes Álvaro e Augusto, terminam com a letra ‘o’, mas começa com a letra A, ou seja, para atender a cláusula Nome-Contribuinte LIKE 'p%o' os nomes deveriam começar com a letra ‘p’.

A **letra E** está errada pois apesar do nome Patrício atender a cláusula Nome-Contribuinte LIKE 'p%o', o nome Paula termina com a letra ‘a’, e não com a letra ‘o’, conforme solicitado.

***O comando UNION é usado para combinar o resultado de duas ou mais consultas SQL, retornando somente dados distintos. Já o comando UNION ALL também é usado para combinar o resultado de duas ou mais consultas SQL, sendo que o resultado final poderá conter dados duplicados quando presentes nos resultados das queries.***

9)Considere o código SQL abaixo, que gerou a tabela ItemFatura.

CREATE TABLE ItemFatura(  
idItemFatura INT NOT NULL,  
idFatura INT NOT NULL,  
descItemFatura VARCHAR(45),  
valorItemFatura DOUBLE,  
**..I..**  
);

Considerando que a tabela ItemFatura possui chave primária composta pelos campos idItemFatura e idFatura, e que se uma fatura for excluída, automaticamente serão excluídos todos os seus itens, a lacuna I deve ser preenchida corretamente por:

A)PRIMARY KEY (idItemFatura, FOREIGN KEY(idFatura)) REFERENCES Fatura(idFatura) ON DELETE CASCADE;

B)PRIMARY KEY (idItemFatura, idFatura), FOREIGN KEY(idFatura) FROM Fatura(idFatura) WITH DELETE CASCADE;

C) PRIMARY KEY (idItemFatura, idFatura), FOREIGN KEY (idFatura) REFERENCES Fatura(idFatura) ON DELETE CASCADE;

D)PRIMARY KEY (idItemFatura, idFatura), FOREIGN KEY(idFatura) REFERENCES Fatura(idFatura);

E) PRIMARY KEY (idItemFatura, idFatura), FOREIGN KEY(idFatura) FROM Fatura(idFatura) ON DELETE CASCADE.

RESPONDER

O enunciado da questão informou que as colunas idItemFatura e idFatura compõe a chave primária(primary key), e que na deleção da chave estrangeira(foreign key) a exclusão de uma fatura será cascateada para os itens que compõe a fatura.

A **letra C é a alternativa correta** pois a sintaxe da primary key está correta, seguido da declaração da foreign key referenciado o campo idFatura da tabela Fatura e finalizando com a declaração do cascateamento da deleção de Itens no caso de deleção de uma Fatura.

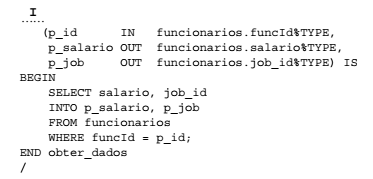
PRIMARY KEY (idItemFatura, idFatura),   
FOREIGN KEY (idFatura) REFERENCES Fatura(idFatura) ON DELETE CASCADE

A **letra A está errada** pois a sintaxe de criação da primary key está incorreta, ou seja, a declaração da foreign key deveria ser após a declaração da PK e não dentro conforme o enunciado da alternativa. A **letra B está errada** pois a sintaxe de criação da foreign key está incorreta. As instruções FROM e WITH não fazem parte da sintaxe de criação da FK.

A **letra D está errada**. Apesar da sintaxe "PRIMARY KEY (idItemFatura, idFatura), FOREIGN KEY(idFatura) REFERENCES Fatura(idFatura)" estar correta, foi dito no enunciado do exercício "que se uma fatura for excluída, automaticamente serão excluídos todos os seus itens", ou seja, a alternativa está incompleta pois está faltando a declaração da instrução de cascateamento quando houver deleção de uma fatura.

A **letra E está errada** pois a sintaxe de criação da foreign key está incorreta. A instrução FROM não faz parte da sintaxe de criação da FK, ou seja, a cláusula FROM deveria ter sido substituída pela cláusula REFERENCES para que a alternativa E fosse considerada correta.

10)Um programador deseja definir uma query SQL para retornar os valores das colunas salario e job\_id para determinado id de funcionário, de forma que ela seja executada de acordo com os parâmetros que receber. Considerando a existência da tabela e dos campos indicados, no Oracle PL/SQL digitou o bloco de comandos abaixo. Completa adequadamente a lacuna I a instrução: CREATE



A)FUNCTION obter\_dados PARAMETERS;

B)OR REPLACE VIEW obter\_dados;

C)OR REPLACE PROCEDURE obter\_dados;

D)TRIGGER obter\_dados;

E)STRUCTURE obter\_dados.