Universidade do Porto
Faculdade de Engenharia
FEUP

Bases de Dados, MIEIC

2010/2011

João Mendes Moreira, Válter Neto da Rocha

10 de Fevereiro de 2011

Parte de escolha múltipla: Duração de 0h30m

Nome:_____ Nº Mecanográfico_____

Seleccione uma e uma só das 4 respostas possíveis. Só há uma resposta certa por alínea. Uma resposta certa vale 0,5 valores; errada vale -0,1 valores; e omissa vale 0 valores.

i. Para que servem os elementos derivados no diagrama de classes UML? ☐

a) Para acrescentar informação que não pode ser expressa de outra forma.
b) Para representar informação que é necessária mas que é redundante.
c) Para representar restrições de integridade.
d) Nenhuma das respostas anteriores.

ii) Diga em que forma normal se encontra o seguinte esquema relacional: R(a,b,c,d) com as DFs: ab->cd; c->b. ☐

a) 1ª FN
b) 2ª FN
c) 3ª FN
d) 3ª FNBC

iii) Sobre chaves e super-chaves: ☐

a) Não há diferenças.
b) Uma chave é uma super-chave, mas uma super-chave pode não ser chave.
c) Uma super-chave é uma chave, mas uma chave pode não ser super-chave.
d) Uma chave nem contém nem está contida numa super-chave.

iv) Em álgebra relacional a instrução $\Pi_{curso,\ cnt(*)}(Cadeira)$ é equivalente à instrução em LMD-SQL: ☐

a) SELECT curso, COUNT(*) FROM Cadeira;
b) SELECT curso, COUNT(*) FROM Cadeira GROUP BY curso;
c) SELECT curso, COUNT(*) FROM Cadeira ORDER BY curso;
d) Nenhuma das respostas anteriores.

v) Em Oracle, o gatilho "*instead of*" permite: ☐

a) Substituir a execução de algumas operações sobre vistas por código PL/SQL alternativo quando certas condições são satisfeitas.
b) Substituir a execução de código PL/SQL existente por outro código PL/SQL alternativo quando certas condições são satisfeitas.
c) Optimizar *queries* complexas.
d) Nenhuma das respostas anteriores.

vi) Diga qual das seguintes afirmações sobre a regra Datalog "*R(x)* <- *S(x)* AND S(*y*) AND *x*>*y*" descreve melhor a segurança dessa regra: ☐

a) A regra não é segura porque *x* não tem limite superior.
b) A regra não é segura porque *y* não está limitada a um conjunto finito.
c) A regra não é segura porque *y* não é uma variável distinta.
d) A regra é segura.

vii) Considere a instrução LMD SQL "INSERT INTO Encomenda (idEncomenda, data, idCliente) VALUES (3, '2011-02-10',100);". Sabendo que a chave primária da tabela Encomenda é idEncomenda e que idCliente é uma chave estrangeira para a tabela Cliente, diga que índices deviam ser criados para tornar esta instrução mais rápida. ☐

a) idEncomenda.
b) idCliente.
c) idEncomenda e idCliente.
d) Nenhuma das respostas anteriores.

viii) Os índices bitmap devem ser criados sobre: ☐

a) Atributos com muitos valores distintos em tabelas com muitos registos.
b) Atributos com poucos valores distintos em tabelas com muitos registos.
c) Atributos com muitos valores distintos em tabelas com poucos registos.
d) Atributos com poucos valores distintos em tabelas com poucos registos.

ix) A cláusula "EXEC SQL" serve para: ☐

a) Executar instruções SQL de imediato.
b) Embeber instruções SQL em linguagens hospedeiras.
c) Executar instruções SQL de imediato em linguagens hospedeiras.
d) Nenhuma das respostas anteriores.

x) Uma das características dos Armazéns de Dados é a sua não volatilidade. O que é que isso significa? ☐

a) Que a informação é guardada de forma segura.
b) Que os utilizadores não podem eliminar dados do Armazém de Dados.
c) Que após a sua introdução no Armazém de Dados, os dados não podem ser alterados ou eliminados.
d) Nenhuma das respostas anteriores.

Bases de Dados, MIEIC

2010/2011

João Mendes Moreira, Válter Neto da Rocha

10 de Fevereiro de 2011

Parte prática: Duração de 2h00m

1. A entidade responsável pelos principais campeonatos de futebol de um país à beira mar plantado pretende informatizar os seus serviços. Para tal, pediu ajuda aos alunos da FEUP-MIEIC.

   Nessa entidade, designada por liga, estão inscritos clubes. Cada clube tem várias equipas e cada equipa só participa num campeonato em cada época. Um clube pode ter uma equipa por cada classe etária distinta (juvenis, juniores, etc.) e por género (masculino ou feminino). Do mesmo modo há, por época, um campeonato por classe etária e por género. As equipas só podem participar no campeonato da mesma facha etária e género.

   Sobre cada equipa é necessário saber os jogadores inscritos. Em cada época existem dois períodos para a inscrição de jogadores. Em cada momento deve ser possível saber quais os jogadores legalmente inscritos na liga e qual o clube a que cada um está vinculado. Num dado momento um jogador só pode estar vinculado a um clube. A equipa a que o jogador pertence é determinada no início da época pela sua facha etária e pelo seu género.

   No início da época é definido um calendário de jogos para cada campeonato. Sobre cada jogo deve constar: a jornada a que diz respeito, data prevista de realização, as equipas em confronto (visitada e visitante) e o estádio onde se realiza o jogo e que é, por defeito, o estádio do clube visitado, mas pode, por algum motivo, ser outro. A data do jogo pode ser alterada pela liga a pedido dos clubes.

   Sobre os jogos é necessário guardar informação sobre os diferentes eventos que ocorrem ao longo do jogo. Esses eventos estão tipificado (ex.: golo, cartão amarelo/vermelho, substituição, etc.). Obviamente que sobre: os cartões disciplinares, é necessário saber a quem foi mostrado; sobre os golos, quem o marcou e a favor de que equipa (é que há golos na própria baliza!); e sobre as substituições, quem entrou e quem saiu. Pode haver outros tipos de eventos que se assume só necessitarem de saber o jogador envolvido. É relevante saber, para todos os eventos, o minuto de jogo em que ocorreu.

   Para cada classe defina os atributos adequados. Por exemplo: os clubes têm nome, morada, ano de fundação, etc.

   a) **(5 valores)** Defina o diagrama de classes UML para a informação acima descrita. Indique todas as restrições que possam ser úteis para a construção da base de dados.

2. **(1 valor)** Considere a relação R(A,B,C,D,E) com as dependências funcionais F={AE->BC, AC->D, BD->C, C->E}. Decompondo R(A,B,C,D,E) em S(A,B,C) e outras relações, indique uma forma minimal para as dependências funcionais que se verificam em S.

3. **(1 valor)** Considere o escalonamento seguinte:

	T_1	T_2	T_3
(1)	WRITE (A)		
(2)			READ (B)
(3)		READ (B)	
(4)	WRITE (B)		
(5)		WRITE (B)	
(6)			READ(A)
(7)			WRITE (A)

Descrever como é que o controlo de concorrência por marcas temporais com a regra de escrita de Thomas lida com o escalonamento dado.

4. **(FAZER ESTE GRUPO NUMA FOLHA DE EXAME SEPARADA)** Uma base de dados de suporte aos encontros de um grupo de jogadores que se costuma reunir regularmente para jogar cartas contém as seguintes tabelas: (1) tabela com o nome e morada do jogador; (2) tabela com os tipos de jogos existentes (Sueca, Copas, …) na qual é guardado o nome do tipo de jogo e se o mesmo é de pares ou single (Multiplicidade); (3) tabela com a lista das diferentes cartas existentes; (4) tabela com todas as cartas usadas (baralho) em cada tipo de jogo; (5) tabela com o tipo de jogo e a data em que o mesmo ocorre para sistematizar os jogos que ocorrem; e a (6) tabela com os participantes em cada um dos jogos e os pontos por ele obtidos nesse jogo. Verifique o seguinte modelo relacional:

MODELO RELACIONAL

Jogador (idJogador, nomeJogador, morada)

TipoJogo (idTipoJogo, nomeTipoJogo, Multiplicidade)

Cartas (idCarta, nomeCarta)

Baralho (idBaralho, nomeBaralho)

TiposJogoBaralho (idBaralho➔Baralho, idTipoJogo➔TipoJogo)

CartasBaralho (idBaralho➔Baralho, idCartas ➔ Cartas)

Jogo(idJogo, idTipoJogo➔TipoJogo, idBaralho➔Baralho, data)

Participantes (idJogo➔Jogo, idJogador➔Jogador, pontos)

a) **(1 valor)** Indique a função PL/SQL *VencedorTipo* que, ao receber o nomeTipoJogo, devolva o idJogador que mais vezes venceu jogos desse tipo (vence um jogo o jogador que tem mais pontos nesse jogo).
b) **(1 valor)** Indique a instrução LMD SQL que tire partido da função *VencedorTipo* e retorne uma lista (nomeJogador, nomeTipoJogo) com os vencedores de cada tipo de jogo.
c) **(1 valor)** Crie uma Função *InsertPossible* que ao receber o nomeJogador e o idJogo verifique se é possível inserir um participante na base de dados, tendo em conta que esta não pode permitir inserções de mais de dois participantes num jogo de singles nem de mais de 4 num jogo de pares e que retorne 1 caso seja possível e 0 caso seja impossível.
d) **(1 valor)** Crie um procedimento *AutoParticipa* que tire partido da função InsertPossible para inserir automaticamente na tabela Participantes o jogador com mais pontos que ainda não está escalado para o jogo em questão.
e) **(1 valor)** Construa uma Vista que devolva qual o Baralho mais usado em tipos de jogos diferentes e quantas vezes foi usado para cada tipo de jogo.
f) **(1 valor)** Usando a LMD SQL diga quantos tipos de jogos diferentes foram realizados por cada um dos jogadores.
g) **(1 valor)** Usando a LMD SQL diga qual o jogador que ficou em segundo lugar quanto ao número total de pontos obtidos em cada um dos tipos de jogos.
h) **(1 valor)** Usando uma só instrução LMD SQL, diga qual foi o jogador que teve mais pontos na Sueca e qual foi o que teve menos pontos nas Copas.

Sintaxe básica de SQL

Instrução SELECT	Operadores	JOIN, UNION, INTERSECT e MINUS
SELECT * FROM tabela;	SELECT * FROM tabela WHERE col1 [NOT] BETWEEN valor1 AND valor2;	SELECT * FROM tabela1, tabela2 WHERE condições;
SELECT col1,col2 FROM tabela;	SELECT * FROM tabela WHERE col1 [NOT] IN (valor1,valor2,…) ;	SELECT * FROM tabela1 INNER JOIN tabela2 ON condições;
SELECT col1,col2 FROM tabela WHERE condições;	SELECT * FROM tabela WHERE col1 [NOT] IN (SELECT …);	SELECT * FROM tabela1 INNER JOIN tabela2 ON condições WHERE condições;
SELECT col1,col2 FROM tabela WHERE condições ORDER BY col1 ASC,col2 DESC;	SELECT * FROM tabela WHERE col1 > valor1 AND col1 < valor2;	SELECT * FROM tabela1 LEFT JOIN tabela2 ON condições;
SELECT DISTICT col1,col2 FROM tabela;	SELECT * FROM tabela WHERE col1 < valor1 OR col1 > valor2;	SELECT * FROM tabela1 RIGHT JOIN tabela2 ON condições;
SELECT col1, agregação(col2) FROM tabela GROUP BY col1;	SELECT * FROM tabela WHERE col1 = valor1;	SELECT col1 from tabela1 UNION SELECT col2 from tabela2;
SELECT col1, agregação(col2) FROM tabela GROUP BY col1 HAVING agregação(col2) > valor1;	SELECT * FROM tabela WHERE col1 = 'texto1';	SELECT col1 FROM tabela1 INTERSECT SELECT col2 FROM tabela2;
SELECT AVG(col1), SUM(col1), COUNT(*), MIN(col1), MAX(col1) FROM tabela;	SELECT * FROM tabela WHERE col1 <> valor1;	SELECT col1 from tabela1 MINUS SELECT col2 from tabela2;

Sintaxe básica de PL/SQL

Exemplos PL/SQL		

DECLARE
x tabela1.col1%TYPE;
<outras declarações opcionais>
BEGIN
x := 10;
<código PL/SQL>
END;
	CREATE OR REPLACE PROCEDURE
	<nome> (<argumentos>) AS
	<declarações opcionais>
	BEGIN
	<código PL/SQL>
	END;
		CREATE OR REPLACE FUNCTION
		<nome>(<argumentos>) RETURN
		VARCHAR2 AS
		<declarações opcionais>
		BEGIN
		RETURN <valor>;
		END <nome>;

LOOP EXIT WHEN <condição> END LOOP;	FOR <var> IN <cursor> LOOP <código PL/SQL> END LOOP;	WHILE <condição> LOOP <código PL/SQL> END LOOP;
CURSOR c IS <consulta>;	FETCH c INTO <variavel(is)>;	LOOP EXIT WHEN c%NOTFOUND END LOOP;

IF (<condição>)
THEN
<código PL/SQL>
ELSIF <condição>
THEN
<código PL/SQL>
END IF;
	CREATE OR REPLACE TRIGGER
	<nome>
	BEFORE INSERT ON Tabela
	FOR EACH ROW
	WHEN (:NEW.col1 IS NULL)
	BEGIN
	SELECT col1 INTO :NEW.col1 FROM
	Tabela2 WHERE idTabela2 =
	:NEW.idTabela2;
	END;
		BEGIN
		OPEN c;
		LOOP
		FETCH c INTO cp;
		EXIT WHEN c%NOTFOUND;
		IF (cp.col1=10) THEN
		UPDATE Tabela1 SET col1 =
		cp.col1
		WHERE col2 = 5;
		END IF;
		END LOOP;
		CLOSE c;
		END;