Matricola: _____
Cognome:_____
Nome:_____
Insegnamento: Basi di dati (DM 270) ☐
Basi di dati e Web (V.O.) ☐
Basi di dati e MM (V.O.) ☐

Basi di Dati/Web/Multimedia
Prova intermedia del 8 giugno 2010

Avvertenze: e' severamente vietato consultare libri e appunti.

Durata 2h

DOMANDE TEORIA COMUNI

a) (3) Lo studente illustri il concetto di VIEW-serializzabilità (VSR), mostrando anche un esempio di schedule VSR (non seriale) e uno di schedule non VSR.

View-serializzabilità

Uno schedule S è view-serializzabile (VSR) se esiste uno schedule seriale S' tale che S' $\approx_V$ S.

View-equivalenza

Due schedule S_1 e S_2 sono view-equivalenti ($S_1 \approx_V S_2$) se possiedono le stesse relazioni LEGGE_DA e le stesse scritture finali.

RELAZIONE "LEGGE_DA"

Dato uno schedule S si dice che un'operazioni di lettura $\mathbf{r_i(x)}$, che compare in S, *LEGGE_DA* un'operazione di scrittura $\mathbf{w_j(x)}$, che compare in S, se $\mathbf{w_j(x)}$ precede $\mathbf{r_i(x)}$ in S e non vi è alcuna operazione $\mathbf{w_k(x)}$ tra le due.

SCRITTURE FINALI

Dato uno schedule S si dice che un'operazione di scrittura $\mathbf{w_i(x)}$, che compare in S, è una SCRITTURA FINALE se è l'ultima operazione di scrittura della risorsa x in S.

Schedule VSR

Schedule VSR: r1(x) r2(x) w1(x) w2(z) r1(z) w2(y) w1(y)

LEGGE_DA={(r1(z),w2(z))} SCRITTURE_FINALI={w1(y), w2(z), w1(x)}

Schedule seriale: r2(x) w2(z) w2(y) r1(x) w1(x) r1(z) w1(y)

LEGGE_DA={(r1(z),w2(z))} SCRITTURE_FINALI={w1(y), w2(z), w1(x)}

Schedule non VSR

(anomalia: perdita di update)

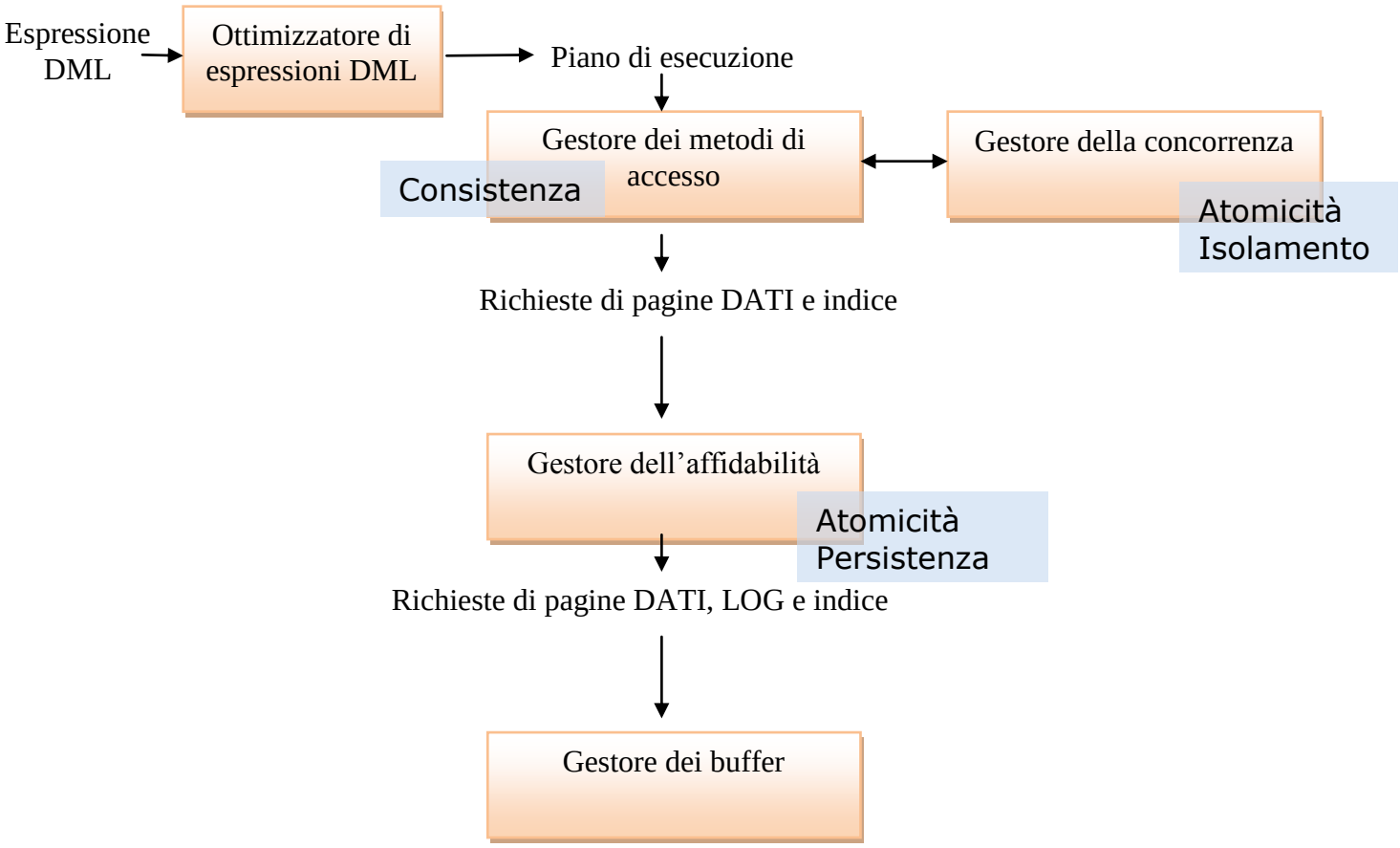
Schedule non VSR: S=r1(x) r2(x) w2(x) w1(x): LEGGE_DA(S)=∅ SCRITTURE_FINALI(S)={w1(X)}

S1=r1(x) w1(x) r2(x) w2(x): LEGGE_DA(S1)={(r2(x),w1(x))} SCRITTURE_FINALI(S1)={(w2(x))}

S2=r2(x) w2(x) r1(x) w1(x): LEGGE_DA(S1)={(r1(x),w2(x))} SCRITTURE_FINALI(S1)={(w1(x))}

S non è VSR

b) (3) Illustrare nello spazio seguente l'architettura di un DBMS indicando quali proprietà delle transazioni vengono garantite da ciascun modulo dell'architettura.



c) (2) Lo studente illustri la differenza tra indici primari e secondari su strutture sequenziali ordinate.

Gli indici sono strutture di accesso ai dati.

Tali strutture velocizzano l'accesso casuale via chiave di ricerca. La chiave di ricerca è un insieme di attributi utilizzati dall'indice nella ricerca.

Indici su file sequenziali

- INDICE PRIMARIO: in questo caso la chiave di ordinamento del file sequenziale coincide con la chiave di ricerca dell'indice.
- INDICE SECONDARIO: in questo caso invece la chiave di ordinamento e la chiave di ricerca sono diverse.

d) (3) Lo studente descriva l'algoritmo di ricerca di una tupla con valore di chiave K in una tabella utilizzando una struttura ad accesso calcolato (Hashing).

RICERCA

- Dato un valore di chiave K trovare la corrispondente tupla
  - Calcolare b=h(f(K)) (costo zero)
  - Accedere al bucket b (costo: 1 accesso a pagina)
  - Accedere alle n tuple attraverso i puntatori del bucket (costo: m accessi a pagina con m<=n)

DOMANDE TEORIA PER BASI DI DATI E WEB

e) (4) Lo studente illustri le caratteristiche fondamentali della tecnologia Java Server Pages (JSP) e il suo ruolo nell'architettura MVC-2 servlet centric.

Una pagina JSP può essere vista come uno "schema di pagina Web" dove:

- le parti statiche sono scritte in HTML e
- le parti dinamiche sono generate attraverso porzioni di codice Java.

Le pagine JSP vengono "gestite" da un componente operante sul web server chiamato *JSP container*

Codice sorgente JSP: è scritto dal programmatore dell'applicazione web, con la collaborazione di altri esperti per la parte di presentazione (grafici, …) .

Si tratta di un file con estensione .jsp contenente:

- **Codice HTML**: parti statiche e grafica
- **Istruzioni Java (scripting JSP)**: parti dinamiche
- **Marcatori speciali JSP**: descrivono il modo in cui generare la servlet associata alla JSP e consentono di gestire oggetti speciali (java data beans, ecc…).

Il "linguaggio" JSP fornisce 4 gruppi principali di marcatori speciali:

- Direttive (directives): …
- Scripting:
  - Dichiarazione (declarations): …
  - Espressione (expressions): …
  - Scriptlet: …
- Azioni (Actions): …
- Commenti: …

DOMANDE TEORIA PER BASI DI DATI E MULTIMEDIA

f) (4) Lo studente illustri le caratteristiche della tecnica di compressione JPEG.

Codifica JPEG sequenziale su singola componente

JPEG encoder

- ***F****orward* **D**escrete **C**osine **T**ransform (**FDCT**): formula
- *Quantizer idea*
- *Entropy Encoder: idea*

JPEG decoder

- *Entropy Decoder: idea*
- *Dequantizer idea*
- ***I****nverse* **DCT** (**IDCT**): formula

DOMANDE TEORIA PER BASI DI DATI

Una domanda a scelta tra e) e f).

ESERCIZI COMUNI

1. (4) Si generi la struttura di un B+-tree con fan-out=5 e contenente 4 nodi foglia con i seguenti valori della chiave di ricerca: {A,B,C,H,L,M,N,O,P,Q,S,T,U,W,Z}

Svolto alla lavagna

2. (3) Si mostri la struttura dell'albero ottenuto al primo esercizio dopo l'inserimento del valore E e dopo l'inserimento del valore F.

Svolto alla lavagna

3. (6) Dato il seguente file XML e i seguenti requisiti si produca il file XML schema che ne descrive la struttura.

XML

```
<?xml version="1.0"?>
<ReteStradale xmlns ="http://www.reteStradale.org"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.reteStradale.org
                                        reteStradale.xsd">
   <Strada id=”STR00002”>
         <tipo>statale</tipo>
         <codice>11</codice>
         <lunghezza unitàMisura=”Km”>855</lunghezza>
   </Strada>
   <Strada id=”STR00023”>
         <tipo>provinciale</tipo>
         <codice>131</codice>
         <lunghezza unitàMisura=”Km”>125</lunghezza>
         <numeroCorsie>2</numeroCorsie>
   </Strada>
…
</ReteStradale>
```

Requisiti

Il tipo può assumere solo uno dei seguenti valori: {tangenziale, statale, provinciale, comunale}.
L’attributo id è obbligatorio.

XMLSchema reteStradale.xsd

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" …
            targetNamespace=“http://www.reteStradale.org“
            xmlns=“http://www.reteStradale.org“ />

<xsd:element name="ReteStradale">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:element ref="Strada" minOccors=”1” maxOccurs="unbounded"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:simpleType name=”tipoStrada” >
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="tangenziale"/>
      <xsd:enumeration value="statale"/>
      <xsd:enumeration value="provinciale"/>
      <xsd:enumeration value="comunale"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name=”tipoCorsia” >
   <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="1"/>
        <xsd:maxInclusive value="10"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name=”tipoLunghezza” >
   <xsd:simpleContent>
        <xsd:extension base=”xsd:integer”>
             <xsd:attribute name=”unitàMisura”
                       type=”xsd:string” use=”required”/>
        </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>

<xsd:element name=”Strada”>
   <xsd:complexType>
        <xsd:sequence>
             <xsd:element name=”tipo” type=”tipoStrada”/>
             <xsd:element name=”codice” type=” xsd:unsignedInt”/>
             <xsd:element name=”lunghezza” type=”tipoLunghezza”/>
             <xsd:element name=”numeroCorsie” type=”tipoCorsie”/>
        </xsd:sequence>
        <xsd:attribute name="id" type=”xsd:ID” use="required" />
   </xsd:complexType>
</xsd:element>
```

ESERCIZI PER BASI DI DATI E WEB

4. (6) Data la seguente base di dati “TeatriVR”, contenente informazioni sugli spettacoli offerti nei teatri della provincia di Verona:

   TEATRO (<u>Codice,</u> Nome, Indirizzo, Comune, Capienza)
   inSCENA (<u>CodT, NomeS, Data</u>, Ora)
   SPETTACOLO (<u>Nome</u>, Descrizione, Protagonista, Genere:{prosa, lirica, concerto musica classica, concerto rock/pop})
   Vincoli di integrità referenziale:

   inSCENA.CodT → TEATRO
   inSCENA.NomeS → SPETTACOLO

   Progettare, secondo la metodologia basata sulla specifica di `page-schema`, lo schema logico di un sito web che presenti le informazioni contenute nella base di dati “TeatriVR”. In particolare:

   a. nella **homePage** si presenti l’elenco dei teatri gestiti (secondo l’ordine alfabetico del loro nome) riportando: il nome e il comune dove si trova il teatro. Il nome del teatro è un link verso lo schema di pagina **teatroPage**. Si aggiunga inoltre un link alla pagina che presenta gli spettacoli di oggi (**oggiPage**).
   b. nello schema di pagina **teatroPage** si presentino tutti i dati che descrivono un teatro: nome, codice, indirizzo, comune e capienza. Inoltre si mostri il numero di spettacoli diversi messi in scena presso il teatro fino a oggi (oggi escluso). Si aggiunga l’elenco degli spettacoli dei prossimi 7 giorni (oggi incluso) indicando: il nome dello spettacolo, il genere e la data di messa in scena. Il nome dello spettacolo è un link verso lo schema di pagina **spettacoloPage**.
   c. nello schema di pagina **spettacoloPage** si presentino tutte le informazioni di uno specifico spettacolo: nome, genere, protagonista e descrizione. Si mostri inoltre il numero (quantità) di messe in scena dello spettacolo previste nel futuro (da oggi in poi).
   d. oggiPage mostra l’elenco degli spettacoli messi in scena oggi con tutte le informazioni disponibili incluse quelle del teatro dove si svolge lo spettacolo.

   Lo studente produca sia gli schemi di pagina che le interrogazioni SQL (`DB to page schema`) che li alimentano. Si supponga presente sul DBMS relazionale, che esegue le interrogazioni SQL, una funzione `current_date` che restituisce la data di oggi.

```
page-schema HomePage unique
( teatri: list-of(teatro: link(nome: String; *teatroPage); comune: String);
  SpettacoliOggi: link(“spettacoli di oggi”, *oggiPage);
)

page-schema teatroPage
( nome: String; codice: String; indirizzo: String; comune: String; capienza: Integer;
  numeroSpettacoli: Integer;
  prossimi7giorni: list-of(spettacolo: link(nome: String; *spettacoloPage);
                           genere: String; data: Date;);
)
page-schema spettacoloPage
( nome: String; genere: String; protagonista: String; descrizione: String;
  numeroMesseInScena: Integer;);

page-schema oggiPage unique
( spettacoli: list-of(nome: String; genere: String;
                      protagonista: String; descrizione: String;
                      teatro: String; codice: String; indirizzo: String;
                      comune: String; capienza: Integer;);
);
```

```
DB to page-schema HomePage
Parameter()
( teatri: SELECT Codice, Nome, Comune FROM TEATRO ORDER BY Nome;
)

DB to page-schema teatroPage
Parameter(codTR)
( nome, codice, indirizzo, comune, capienza, numeroSpettacoli:
       SELECT Nome, Codice, Indirizzo, Comune, Capienza, count(*) as numeroSpettacoli
       FROM TEATRO JOIN inSCENA ON Codice=CodTR
       WHERE Data<current_date AND Codice=?codT?
       GROUP BY Nome, Codice, Indirizzo, Comune, Capienza;

 prossimi7giorni: SELECT Nome, Genere FROM SPETTACOLO JOIN inSCENA ON Nome=NomeS
                  WHERE CodT=?CodTR? AND Data >= current_date
                        AND Data <= current_date+7
)

DB to page-schema spettacoloPage
Parameter(nomeSP)
( nome, genere, protagonista, descrizione, numeroMesseInScena:
       SELECT Nome, Genere, Protagonista, Descrizione, count(*) as numeroMesseInScena
       FROM SPETTACOLO JOIN inSCENA ON Nome=NomeS
       WHERE Data>=current_date AND Nome=?nomeSP?
       GROUP BY Nome, Genere, Protagonista, Descrizione;
)

DB to page-schema oggiPage
Parameter()
( spettacoli: SELECT S.Nome, S.Genere, S.Protagonista, S.Descrizione,
                     T.Nome, T.Codice, T.Indirizzo, T.Comune, T.Capienza
              FROM SPETTACOLO S JOIN inSCENA ON S.Nome=NomeS
                                JOIN TEATRO T ON T.Codice=CodT
              WHERE Data = current_date;
)
```

ESERCIZI PER BASI DI DATI E MULTIMEDIA

5. (2) Si descrivano le caratteristiche della classe PreparedStatement della libreria JDBC.

E' possibile ottimizzare l'esecuzione di una interrogazione che deve essere rifatta più volte usando la classe PreparedStatement.
Tale classe consente di inserire parametri nell'interrogazione (attraverso il simbolo '?') e di valorizzarli, usando specifici metodi (setInt(pos,valoreInt), setString(pos,valoreString), …), prima dell'effettiva esecuzione dell'interrogazione stessa. In questo modo si lascia ai metodi della classe PreparedStatement l'onere di convertire i valori dai tipi Java ai tipi SQL.

Esempio:

```
Connection con = DriverManager.getConnection(URL,
    user, passwd);
String q = "SELECT Nome, Cognome"+
        " FROM Persona"+
        " WHERE id = ?";
PreparedStatement pstat = con.prepareStatement(q);
pstat.setInt(1, 15);
ResultSet res = pstat.executeQuery();
```

6. (4) Data una sorgente di informazione con le seguenti caratteristiche:
   - Alfabeto = {A,B,C,D,E}
   - P(A)=P(B)=P(E)
   - P(A)+P(B)+P(E)=0.75
   - P(C)=0.125

   6.a (1) Calcolare l'informazione (o entropia) della sorgente.

   P(A) = 0.25
   P(B) = 0.25
   P(C) = 0.125
   P(E) = 0.25

   Entropia sorgente = $3^* (0.25 * -\log_2(0.25)) + 2 * (0.125 * -\log_2(0.125))$
   $= 3^* (0.25 * 2) + 2^* (0.125 * 3) = 18/8 = 2{,}25$

   6.b (2) Calcolare una possibile codifica in bit secondo Huffman per i simboli dell'alfabeto.

   A = 10
   B = 11
   E = 00
   C = 010
   D = 011

   6.c (1) Codificare il messaggio: AAABEECB

   10101011000001011

ESERCIZI PER BASI DI DATI

Esercizio 4) senza gli schemi di pagina c) e d)
Esercizio 6).