INFORMATICA DOCUMENTALE

Laurea in Scienze della Comunicazione – Università degli Studi di Verona

VERIFICA SCRITTA DEL 5 SETTEMBRE 2013

matricola	cognome	nome	firma

A.1 + A.2 + A3	B.1	B.2	B.3	Totale

Istruzioni

- È vietato portare all'esame libri, eserciziari, appunti e dispense. Chiunque venga trovato in possesso di documentazione relativa al corso – anche se non attinente alle domande proposte – vedrà annullata la propria prova.
- Scrivere solo sui fogli distribuiti, cancellando le parti di brutta con un tratto di penna. Non separare questi fogli.
- Tempo a disposizione: 1 ora e 45 minuti.

A. Parte prima

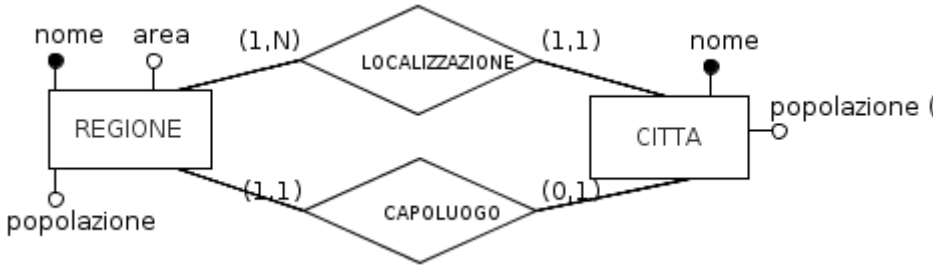
A.1. In al più 10 righe dire cosa significa che i DBMS devono poter gestire basi di dati grandi, persistenti e condivise?

A.2. Utilizzando il modello Entità-Relazione (ER) fare un esempio di relazione ricorsiva.

A.3. Nell'ambito del modello E-R, definire il concetto di identificatore esterno, spiegando quando è necessario con un esempio.

B. Parte seconda

B.1. Considerare il seguente schema ER.



B.1.a. In al più 3 righe, spiegare cosa significa (0,1) vicino all'entità CITTA.

B.1.b. Solo sulla base dell'ER sopra (cioè senza ulteriori vincoli e/o informazioni) è possibile avere due città con lo stesso nome in regioni diverse? Motivare brevemente (max 5 righe) la risposta e in caso di risposta negativa dire come si potrebbe modificare l'ER per renderlo possibile.

B.1.c. Ipotizzando che gli abitanti vivano solamente nelle città riportate, nello schema ER c'è qualche attributo ridondante? Se sì, spiegare vantaggi e svantaggi di avere o non avere tale/i attributo/i.

B.1.d. Se si volesse rappresentare anche il nome del sindaco dei capoluoghi di regione (solo dei capoluoghi) come si potrebbe modificare l'ER?

B.1.e. Tradurre lo schema ER in uno schema relazionale, indicando in quest'ultimo eventuali chiavi, vincoli di non nullità e vincoli di integrità referenziale.

B.2. Considerare il seguente schema relazionale che rappresenta una parte di un sistema per la gestione di gruppi musicali:

```
MUSICISTA(nome, cognome, data_nascita*)
GRUPPO_MUSICALE(nome, tipologia)
STRUMENTO(nome)
PARTECIPAZIONE(nome_musicista_, cognome_musicista, gruppo, strumento)
```

con vincoli di integrità referenziale tra gli attributi nome_musicista e cognome_musicista di PARTECIPAZIONE e i rispettivi attributi della relazione MUSICISTA, tra l'attributo gruppo di PARTECIPAZIONE e l'attributo nome della relazione GRUPPO_MUSICALE e tra l'attributo strumento di PARTECIPAZIONE e l'attributo nome di STRUMENTO, e la seguente istanza.

GRUPPO_MUSICALE	
nome	**tipologia**
Anaconda Fusion	Jazz
New Beatles	Rock
Rondò Veneziano	Classica

MUSICISTA		
nome	**cognome**	**data_nascita**
Claudio	Allevi	NULL
Enrico	Bentivoglio	1968-11-01
Uto	Ughi	NULL
Gianni	Compri	1954-04-24

STRUMENTO
nome
Pianoforte
Sassofono
Violino
Viola
Basso elettrico

PARTECIPAZIONE			
nome_musicista	**cognome_musicista**	**gruppo**	**strumento**
Claudio	Allevi	Rondò Veneziano	Pianoforte
Enrico	Bentivoglio	Rondò Veneziano	Sassofono
Claudio	Allevi	New Beatles	Pianoforte
Enrico	Bentivoglio	Anaconda Fusion	Sassofono
Uto	Ughi	Rondò Veneziano	Violino
Claudio	Allevi	Rondò Veneziano	Viola
Gianni	Compri	Anaconda Fusion	Basso elettrico

B.2.a. Scrivere i comandi SQL per creare le tre tabelle sopra riportate (incluse eventuali chiavi, vincoli di non nullità, vincoli di unicità e vincoli di integrità referenziale). Tenere conto che deve essere evitato di cancellare un musicista o un gruppo, a cui fa riferimento almeno una tupla di PARTECIPAZIONE, mentre se viene eliminato uno strumento in STRUMENTO vengono eliminate i record in PARTECIPAZIONE, che vi facevano riferimento. In caso di modifica del nome o del cognome di un musicista, di un orchestra oppure del nome di uno strumento, tali modifiche si devono riflettere in automatico anche in PARTECIPAZIONE.

B.2.b. Cosa restituisce la seguente interrogazione? (scrivere la tabella risultante)

```
SELECT strumento, COUNT(*) AS numero FROM partecipazione
WHERE strumento <> 'Basso elettrico' GROUP BY strumento ORDER BY strumento;
```

B.2.c. Cosa restituisce la seguente interrogazione? (scrivere la tabella risultante)

```
SELECT cognome_musicista AS musicista
FROM partecipazione INNER JOIN strumento ON partecipazione.strumento = strumento.nome
WHERE gruppo = 'Rondò Veneziano' AND nome_musicista LIKE '%o'
GROUP BY cognome_musicista HAVING COUNT(*) > 1;
```

B.2.d. Scrivere il comando SQL per aggiornare nell'intero database il nome dello strumento “Sassofono” in “Sax”.

B.2.e. Dire se il seguente comando è corretto e in tal caso scrivere quanti record avranno dopo la sua esecuzione le tabelle STRUMENTO e PARTECIPAZIONE sull'istanza sopra riportata, oppure se non lo è spiegare il motivo (max 3 righe).

```
DELETE FROM strumento WHERE nome IN (SELECT DISTINCT strumento FROM partecipazione
WHERE nome_musicista = 'Claudio');
```

B.3. Si consideri il seguente schema SQL, che rappresenta un semplice sistema di gestione ticket. Il sistema prevede degli utenti dei progetti e delle tipologie di attività/servizio. Per richiedere un intervento si apre un ticket, dove si specifica per un certo progetto l'attività richiesta. Il ticket può in seguito essere preso in carico da un utente (incaricato), che, una volta completato il compito (l'attività), chiude il ticket, impostando chiuso a TRUE.

```
CREATE TABLE utenti (
    username VARCHAR(15) PRIMARY KEY,
    password VARCHAR(15) NOT NULL
);
CREATE TABLE progetti (
    nome VARCHAR(32) PRIMARY KEY,
    citta VARCHAR(32)
);
CREATE TABLE attivita (
    descrizione VARCHAR PRIMARY KEY,
    costo INT NOT NULL
);
CREATE TABLE ticket (
    codice INT PRIMARY KEY,
    progetto VARCHAR(32),
    attivita VARCHAR,
    incaricato VARCHAR(15),
    chiuso BOOL DEFAULT FALSE,
    FOREIGN KEY(progetto) REFERENCES progetti(nome) ON UPDATE CASCADE ON DELETE SET
NULL,
    FOREIGN KEY(attivita) REFERENCES attivita(descrizione) ON UPDATE CASCADE ON DELETE
SET NULL,
    FOREIGN KEY(incaricato) REFERENCES utenti(username) ON UPDATE CASCADE ON DELETE
SET NULL
);
```

scrivere i comandi SQL che permettono di:

B.3.a. Ottenere il numero totale dei ticket chiusi dall'utente "Luigi".

B.3.b. Ottenere per ciascun incaricato la somma dei costi delle sue attività non ancora chiuse, ordinate per somma decrescente.

B.3.c. Ottenere i nomi non ripetuti di tutti gli utenti che hanno già chiuso ticket per progetti su Verona.

B.3.d. Aggiungere il ticket con codice 15 per il progetto 'Livenet Univr' per l'attività di 'Addestramento', che costa 100, tenendo conto che non sono ancora presenti nel database né tale progetto né tale attività

B.3.e. Rimuovere i ticket relativi ad attività che appaiono per più di 2 volte nella tabella ticket stessa.

SOLUZIONI

A. Parte prima

A.1. *Grandi*: le basi di dati possono avere dimensioni (molto) maggiori della memoria centrale dei sistemi di calcolo utilizzati e il limite deve essere solo quello fisico dei dispositivi.
Persistenti: le basi di dati hanno un tempo di vita indipendente dalle singole esecuzioni dei programmi, che le utilizzano.
Condivise: una base di dati è condivisa da più applicazioni ed utenti, che vi possono accedere anche contemporaneamente e concorrentemente.

A.2. Una relazione ricorsiva mette in relazione un'entità con se stessa. Come si può vedere dal libro di testo (Atzeni e altri) nella fig. 7.8, in alcuni casi è necessario stabilire anche dei, ruoli che l'entità coinvolta svolge nella relazione.

A.3. In alcuni casi, gli attributi propri di un'entità non sono sufficienti per identificare ciascuna occorrenza dell'entità stessa. Ad esempio, per l'entità STUDENTE non basta la matricola per identificare ciascuno studente, in quanto vi possono essere valori di matricola uguali in università diverse. In questi casi, utilizzo l'identificatore di un'altra entità (esterna), che sia in relazione con quella di partenza. Nell'esempio precedente, posso identificare univocamente uno studente grazie alla sua matricola e all'identificatore dell'università, a cui è iscritto.

B. Parte seconda

B.1.

B.1.a. (0,1) tra l'entità CITTA e la relazione CAPOLUOGO rappresenta la cardinalità dell'entità rispetto a questa relazione. Essa sta a significare che ogni occorrenza in CITTA (cioè ogni città) può essere in relazione CAPOLUOGO con al più una regione o anche nessuna. In altre parole ogni città può non essere capoluogo di nessuna regione o esserlo al più di una.

B.1.b. No, infatti, non solo non è possibile avere due città con lo stesso nome in regioni diverse, ma non è proprio possibile avere due città con lo stesso nome in CITTA. Nell'entità CITTA una città è identificata solo dal nome e quindi l'entità non può contenere due città diverse (e quindi, in generale, con popolazione diversa) con lo stesso nome, altrimenti quest'ultimo non potrebbe essere un identificatore valido.
Una possibile soluzione a questo problema è identificare CITTA tramite un identificatore esterno, che includa sia il nome della città sia la regione in cui essa si trova (attraverso la relazione LOCALIZZAZIONE).

B.1.c. L'attributo popolazione di regione può essere calcolato sommando le popolazioni delle città, che si trovano nella regione. Il fatto di avere l'attributo popolazione per ciascuna regione comporta un vantaggio in termini di tempo in fase di interrogazione a livello di regione, Lo svantaggio, in questo caso, a parte lo spazio di memoria in più (comunque molto ridotto), è soprattutto quello di dover tenere sempre allineata la base di dati, affinché eventuali modifiche delle popolazioni delle città siano riportate anche3 nella popolazione totale della regione relativa.

B.1.d. La soluzione più semplice è aggiungere un attributo NOME_SINDACO alla relazione (o associazione) CAPOLUOGO. Un'altra possibile soluzione è aggiungere un'entità SINDACO, con i propri attributi, e collegarla alla relazione CAPOLUOGO.

B.1.e. `CITTA(nome, popolazione*, regione)`
`REGIONE(nome, area, popolazione, capoluogo)`
con vincolo di integrità referenziale tra l'attributo `regione` e l'entità `REGIONE`.
con vincolo di integrità referenziale tra l'attributo `capoluogo` e l'entità `CITTA`.

B.2.

B.2.a.
```
CREATE TABLE musicista (
    nome varchar(32),
    cognome varchar(32),
    data_nascita date,
    PRIMARY KEY(cognome, nome)
);

CREATE TABLE gruppo_musicale (
    nome varchar(30) PRIMARY KEY,
    tipologia  varchar(30) NOT NULL
);

CREATE TABLE strumento (
    nome varchar(30) PRIMARY KEY
);

CREATE TABLE partecipazione (
    nome_musicista varchar(32),
    cognome_musicista varchar(32),
    gruppo varchar(30),
    strumento varchar(30),
    PRIMARY KEY(nome_musicista, cognome_musicista, gruppo, strumento),
    FOREIGN KEY(nome_musicista, cognome_musicista) REFERENCES musicista(nome, cognome) ON
DELETE NO ACTION ON UPDATE CASCADE,
```

```
    FOREIGN KEY(gruppo) REFERENCES gruppo_musicale(nome) ON DELETE NO ACTION ON UPDATE
CASCADE,
    FOREIGN KEY(strumento) REFERENCES strumento(nome) ON DELETE CASCADE ON UPDATE CASCADE
);
```

B.2.b.

strumento	numero
Pianoforte	2
Sassofono	2
Viola	1
Violino	1

B.2.c.

musicista
Allevi

B.2.d. `UPDATE strumento SET nome = 'Sax' WHERE nome = 'Sassofono';`

B.2.e. Il comando è corretto e dopo la sua esecuzione la tabella STRUMENTO avrà 3 record, mentre PARTECIPAZIONE ne avrà 4,

B.3.

B.3.a. `SELECT COUNT(*) AS totale FROM ticket WHERE incaricato = 'Luigi' ADN chiuso = TRUE;`

B.3.b. `SELECT incaricato, SUM(costo) AS totale_costo FROM ticket INNER JOIN attivita ON attivita = descrizione WHERE chiuso = FALSE GROUP BY incaricato ORDER BY totale_costo DESC;`

B.3.c. `SELECT DISTINCT incaricato FROM ticket INNER JOIN progetti ON progetto = nome WHERE citta = 'Verona' AND chiuso = TRUE;`

B.3.d.
```
INSERT INTO progetti VALUES ('Livenet Univr', 'Verona');
INSERT INTO attivita VALUES ('Addestramento', 100);
INSERT INTO ticket(codice, progetto, attivita) VALUES (15, 'Livenet Univr',
'Addestramento');
```

B.3.e. `DELETE FROM ticket WHERE attivita IN (SELECT attivita FROM ticket GROUP BY attivita HAVING COUNT(*) > 2);`