Basi di Dati
More SQL

Michele Beretta

michele.beretta@unibg.it

Esercizio E.1

Prova in itinere 20/04/2017

```
Studente(Matr, Nome, DataNascita, Città)
Corso(Codice, Titolo, CFU, Docente)
PianoStudi(MatrStud, CodCorso, Anno)
Esame(MatrStud, CodCorso, Data, Voto, NroProgr)
```

1. Estrarre gli studenti ordinati per la media dei voti pesata per CFU, solo per chi ha acquisito almeno 60 CFU in totale.
2. Assegnare in SQL a ciascun esame il numero progressivo nella carriera dello studente in base alla data di superamento.

Esercizio E.1

Estrarre gli studenti ordinati per la media dei voti pesata per CFU, solo per chi ha acquisito almeno 60 CFU in totale.

```
SELECT MatrStud, SUM(CFU * Voto) / SUM(CFU) AS MediaPesata
FROM Esame JOIN Corso ON CodCorso = Codice
GROUP BY MatrStud
HAVING SUM(CFU) >= 60
ORDER BY MediaPesata
```

Esercizio E.1

Assegnare in SQL a ciascun esame il numero progressivo nella carriera dello studente in base alla data di superamento.

```sql
UPDATE Esame E1
SET NroProgr = 1 + (
  SELECT COUNT(*)
  FROM Esame E2
  WHERE E1.MatrStud = E2.MatrStud AND E2.Data < E1.Data
)
```

Esercizio E.2

Prova in itinere 19/04/2016

```
Studente(Matr, Nome, DataNascita, Città)
Corso(Codice, Titolo, CFU, Docente)
PianoStudi(MatrStud, CodCorso, Anno)
Esame(MatrStud, CodCorso, Data, Voto, NroProgr)
```

Per semplicità, suppongo di avere una funzione `year` che ritorna l'anno.

```sql
SELECT Matr, 110 / 30 * SUM(CFU * Voto) / SUM(CFU) as Media110
FROM Esame
  JOIN Corso ON CodCorso = Codice
  JOIN Studente ON MatrStud = Matr
WHERE year(DataNascita) = 1995
GROUP BY Matr
HAVING SUM(CFU) >= 100
```

Esercizio E.3

Tema d'esame settembre 2010

```
Aeroporto(Id, Città, Nazione, NumPiste)
Volo(NumeroVolo, GiornoSett, IdAeropPartenza, IdAeropArrivo, Compagnia,
     OraPartenza, OraArrivo, CodAereo)
Aereo(Codice, Tipo, NumPasseggeri)
```

1. Esprimere in SQL la query che estrae la coppia di città tra cui viene offerta la maggiore capacità complessiva di trasporto passeggeri (ipotizzando che la capacità sia simmetrica, come si potrebbe rendere la query più efficiente?)
2. Estrarre in SQL le compagnie che servono la città di Bergamo con più di settanta voli alla settimana, usando aerei di un solo tipo per tutti i loro voli.

Esercizio E.3

Premessa: il testo dell'esercizio parla di capacità *simmetrica* e *asimmetrica*. Un errore tipico è tentare di risolvere l'esercizio senza comprendere appieno lo schema di memorizzazione.

Lo schema propone una struttura per organizzare le rotte aeree, e guardando alla domanda fra parentesi del punto 1, se ne deduce che *per alcune città* la *capacità di trasporto è asimmetrica*.

Esercizio E.3

1. Esprimere in SQL la query che estrae la coppia di città tra cui viene offerta la maggiore capacità complessiva di trasporto passeggeri (ipotizzando che la capacità sia simmetrica, come si potrebbe rendere la query più efficiente?)

Cerchiamo di calcolare la capacità dei singoli link.

```
CREATE VIEW Trasporto(Da, A, NumPasseggeri) AS
SELECT P.Città, A.Città, SUM(NumPasseggeri)
FROM Volo
  JOIN Aereo ON CodAereo = Codice
  JOIN Aeroporto P ON IdAeropPartenza = P.Id
  JOIN Aeroporto A ON IdAeropArrivo = A.Id
GROUP BY P.Città, A.Città
```

Ora cerchiamo di calcolare la capacità per ogni coppia di città, tenendo conto del tipo di link (asimmetrico o simmetrico).

```
CREATE VIEW Totali(CityA, CityB, NumPasseggeri) AS
SELECT T1.Da, T1.A, T1.Passeggeri + T2.Passeggeri
FROM Trasporto T1
  JOIN Trasporto T2 ON T1.Da = T2.A AND T1.A = T2.Da
UNION
SELECT Da, A, Passeggeri
FROM Trasporto
WHERE (Da, A) NOT IN (SELECT A, Da FROM Trasporto)
```

Ora possiamo trovare la coppia con capacità massima.

```
SELECT DISTINCT CityA, CityB
FROM Totali
WHERE NumPasseggeri = (SELECT MAX(NumPasseggeri) FROM Totali)
```

Se la capacità fosse solo simmetrica, è immediato cambiare le query. L'esercizio viene lasciato al lettore.

Esercizio E.3

2. Estrarre in SQL le compagnie che servono la città di Bergamo con più di settanta voli alla settimana, usando aerei di un solo tipo per tutti i loro voli

Cerchiamo prima tutte le compagnie che partono da Bergamo.

```sql
CREATE VIEW BGP(Compagnia, TipoAereo) AS
SELECT Compagnia, Tipo
FROM Volo V
  JOIN Aeroporto A ON V.IdAeroportoPartenza = A.Id
  JOIN Aereo AE ON AE.Codice = V.CodAereo
WHERE A.Città = "Bergamo"
```

Ripetendo lo stesso procedimento per l'aeroporto di arrivo otteniamo `BGA`. Unendo le due (conservando i duplicati) otteniamo tutte le compagnie che servono Bergamo (in partenza o in arrivo), conservando la cardinalità che è pari al numero dei voli.

```
CREATE VIEW BGTot(Compagnia, TipoAereo) AS
SELECT * FROM BGP
UNION ALL
SELECT * FROM BGA
```

Ricaviamo quindi le compagnie che servono Bergamo usando un solo tipo di aereo.

```
CREATE VIEW BGE(Nome) AS
SELECT Compagnia
FROM BGTot A
WHERE NOT EXISTS (SELECT *
                  FROM BGTot B
                  WHERE A.Compagnia = B.Compagnia
                    AND A.TipoAereo <> B.TipoAereo)
```

Infine, selezioniamo da queste tutte le compagnia che offrono più di 70 voli e otteniamo il risultato cercato.

```
SELECT Compagnia
FROM BGTot
WHERE Compagnia IN (SELECT * FROM BGE)
GROUP BY Compagnia
HAVING COUNT(*) > 70
```

Esercizio E.4

Tema d'esame febbraio 2003

```
Cliente(Id, Cognome, Nome, NumNoleggi, Tipologia)
Film(Titolo, Regista, Genere, Durata)
Cassetta(Id, Titolo)
Dvd(Id, Titolo)
Noleggio(IdCassODvd, CodCliente, DataPrestito, DataRestituzione)
```

1. Estrarre in SQL tutti i film di cui sono stati noleggiati più volte i DVD che le videocassette.
2. Estrarre in SQL tutti i film le cui cassette sono state noleggiate più di 20 volte nel mese di gennaio, ma che non sono presenti in formato DVD. Fare in modo che vengano stampati i titoli in ordine opposto a quello alfabetico e che vengano visualizzati solo quelli in posizione 2 e 3 della lista.
3. Determinare cognome e nome dei clienti che hanno noleggiato tutti i film.
4. Determinare cognome e nome dei clienti che hanno noleggiato DVD solo di genere giallo.

Esercizio E.4

Estrarre in SQL tutti i film di cui sono stati noleggiati più volte i DVD che le videocassette.

```sql
CREATE VIEW NolDvd(Titolo, Numero) AS
SELECT Titolo, COUNT(*)
FROM Dvd JOIN Noleggio ON Id = IdCassODvd
GROUP BY Titolo

CREATE VIEW NolCassetta(Titolo, Numero) AS
SELECT Titolo, COUNT(*)
FROM Cassetta JOIN Noleggio ON Id = IdCassODvd
GROUP BY Titolo

SELECT Titolo
FROM NolDvd D JOIN NolCassetta C ON D.Titolo = C.Titolo
WHERE D.Numero > C.Numero
```

Esercizio E.4

Estrarre in SQL tutti i film le cui cassette sono state noleggiate più di 20 volte nel mese di gennaio 2003, ma che non sono presenti in formato DVD. Fare in modo che vengano stampati i titoli in ordine opposto a quello alfabetico e che vengano visualizzati solo quelli in posizione 2 e 3 della lista.

```
SELECT Titolo
FROM Cassetta JOIN Noleggio ON Id = IdCassODvd
WHERE DataPrestito BETWEEN 1/1/2003 AND 31/1/2003
  AND Titolo NOT IN (SELECT Titolo FROM Dvd)
GROUP BY Titolo
HAVING COUNT(*) > 20
ORDER BY Titolo DESC
LIMIT 2
OFFSET 1
```

Esercizio E.4

Determinare cognome e nome dei clienti che hanno noleggiato tutti i film.

Nota: `Noleggio` usa `IdCassODvd`, quindi gli ID delle cassette e gli ID dei DVD sono *diversi e riconoscibili* e si può fare l'unione tra i due senza avere conflitti.

```sql
CREATE VIEW AllCassDvd(Id, Titolo) AS
SELECT Id, Titolo
FROM Cassetta
UNION
SELECT Id, Titolo
FROM Dvd
```

```sql
SELECT Cognome, Nome
FROM Cliente C
WHERE NOT EXISTS (
  SELECT *
  FROM Film F
  WHERE NOT EXISTS (
    SELECT *
    FROM Noleggio
      JOIN AllCassDvd ON IdCassODvd = Id
    WHERE Titolo = F.Titolo
      AND CodCliente = C.Id
  )
)
```

Esercizio E.4

Determinare cognome e nome dei clienti che hanno noleggiato DVD solo di genere giallo.

```
SELECT Cognome, Nome
FROM Cliente C
  JOIN Noleggio N ON C.Id = N.CodCliente
  JOIN Dvd D ON N.IdCassODvd = D.Id
WHERE NOT EXISTS (
  SELECT *
  FROM Noleggio N2
    JOIN Dvd D2 ON N2.IdCassODvd = D2.Id
    JOIN Film F ON D2.Titolo = F.Titolo
  WHERE N2.CodCliente = C.Id
    AND F.Genere <> 'Giallo'
)
```

Query ricorsive

Ci sono alcune restrizioni sull'uso di interrogazioni ricorsive in una vista ricorsiva. Nello specifico, la query deve essere monotona, cioè il risultato sull'istanza I_1 di una vista di una relazione deve essere un superset del suo risultato sull'istanza I_2 se I_1 è un superset di I_2.

Intuitivamente, se alcune tuple vengono aggiunte ad una vista allora la query ricorsiva che viene eseguita sulla medesima vista deve almeno produrre lo stesso sottoinsieme di tuple prodotte in una precedenta invocazione, e possibilmente alcune tuple in aggiunta.

Le query ricorsive non devono usare nessuno dei seguenti costrutti, in quanto la loro presenza potrebbe rendere la query non-monotona:

- **Aggregazioni** sulla vista ricorsiva
- `NOT EXISTS` su una sotto-query che usa la vista ricorsiva
- Operatore `EXCEPT` con operando destro (sottraendo) una vista ricorsiva

Esercizio R.1

Dato il seguente schema

```
PreReq(IdCorso, IdPrereq)
```

Trovare tutti i prerequisiti del corso `ID0018`.

Soluzione

1. Esprimiamo l'interrogazione usando una vista (ricorsiva) di supporto
2. Materializziamo la vista con una seconda query, detta "base"

Esercizio R.1

Notare la separazione in **passo base** (prima di `UNION`) e **passo ricorsivo** (dopo di `UNION`).

```
WITH RECURSIVE CPrereq(IdCorso, IdPrereq) AS (
  SELECT IdCorso, IdPrereq
  FROM Prereq
  WHERE IdCorso = "ID0018"
      UNION
  SELECT CPrereq.IdCorso, CPrereq.IdPrereq
  FROM Prereq P, CPrereq
  WHERE P.IdCorso = CPrereq.IdPrereq
)

SELECT * FROM CPrereq
```

Esercizio R.2

Dato il seguente schema

```
Squadra(Nome, MediaPunti)
```

Costruire una tabella che mostri il ranking delle squadre.

Viene concesso l'utilizzo della sola funzione `COUNT()`.

Soluzione

Essenzialmente le richieste sono due:

1. Assegnare una posizione in classifica ad ogni squadra
2. Ordinare le tuple in base alla posizione

Esercizio R.2

```
SELECT Nome, (1 + (SELECT COUNT(*)
                   FROM Squadra B
                   WHERE B.MediaPunti > A.MediaPunti)
               ) AS Rank
FROM Squadra A
ORDER BY Rank
```