Schema relazionale relativo a prenotazioni in un teatro.

```
SPETTATORI(IDSPETTATORE, NOMINATIVO, TEL, QUARTIERE, CITTA)
CONCERTI(IDCONCERTO, TITOLO, ARTISTA, DATA)
PRENOTAZIONI(IDSPETTATORE, IDCONCERTO, PREZZOBIGLIETTO)
    FK IDSPETTATORE REFERENCES SPETTATORI
    FK IDCONCERTO REFERENCES CONCERTI
```

Note
QUARTIERE contiene valori numerici interi positivo
per semplicità definire l'attributo DATA come ome esto di 8 caratteri AAAAMMGG

Scrivere in SQL le seguenti interrogazioni

a. Restituire per ogni concerto (cioè titolo) e relativa città il numero delle prenotazioni e il relativo ammontare in Euro ordinando il risultato sull'ammontare;

b. Restituire il numero di spettatori che hanno partecipato a **tutti** i concerti tra il 20020101 e il 20021231

Creazione del db e esempio di inserimento

```
CREATE TABLE sp( -- spettatori
        id_sp CHAR(16)  PRIMARY KEY,
        nom CHAR(64) NOT NULL,
        tel CHAR(16),
        quartiere CHAR(16),
        citta CHAR(64)
);

CREATE TABLE co( -- concerti
        id_co CHAR(16)  PRIMARY KEY,
        titolo CHAR(256) NOT NULL,
        artista CHAR(256),
        data CHAR(8) NOT NULL
);
CREATE TABLE pr( -- prenotazioni
 id_sp CHAR(16) NOT NULL,
 id_co CHAR(16) NOT NULL,
 prezzo numeric(8,2),
  PRIMARY KEY (id_sp,id_co),
  FOREIGN KEY (id_sp) REFERENCES sp,
  FOREIGN KEY (id_co) REFERENCES co
);
```

Inserimento dati

```
INSERT INTO sp VALUES( 'SP1',  'N1',   'TE1',  'Q1',   'C1'    );
INSERT INTO sp VALUES( 'SP2',  'N2',   'TE2',  'Q1',   'C1'    );
INSERT INTO sp VALUES( 'SP3',  'N3',   'TE3',  'Q2',   'C1'    );
INSERT INTO sp VALUES( 'SP4',  'N4',   'TE4',  'Q2',   'C1'    );
INSERT INTO sp VALUES( 'SP5',  'N5',   'TE5',  'Q3',   'C1'    );
INSERT INTO sp VALUES( 'SP6',  'N6',   'TE6',  'Q3',   'C1'    );
INSERT INTO sp VALUES( 'SP7',  'N7',   'TE7',  'Q3',   'C1'    );
INSERT INTO sp VALUES( 'SP8',  'N8',   'TE8',  'Q4',   'C1'    );

INSERT INTO co VALUES( 'S1',   'TT1',  'A1',   '20020101'     );
INSERT INTO co VALUES( 'S2',   'TT2',  'A1',   '20020601'     );
INSERT INTO co VALUES( 'S3',   'TT3',  'A2',   '20021201'     );
INSERT INTO co VALUES( 'S4',   'TT3',  'A2',   '20030601'     );
INSERT INTO co VALUES( 'S5',   'TT4',  'A3',   '20010401'     );

INSERT INTO pr VALUES( 'SP1',  'S1',   25      );
INSERT INTO pr VALUES( 'SP2',  'S2',   30      );
INSERT INTO pr VALUES( 'SP2',  'S3',   35      );
```

```
INSERT INTO pr VALUES( 'SP3',  'S3',   35     );
INSERT INTO pr VALUES( 'SP2',  'S4',   25     );
INSERT INTO pr VALUES( 'SP4',  'S4',   25     );
```

Query A)

```
-- Restituire per ogni concerto (cioè titolo)
-- il numero delle prenotazioni e il relativo ammontare in Euro
-- ordinando il risultato sull'ammontare

SELECT titolo, COUNT(*) AS numero, SUM(prezzo) AS importo
FROM co JOIN pr ON co.id_co=pr.id_co
GROUP BY titolo
ORDER BY importo DESC
```

Query B)

```
-- trova il numero di spettatori che hanno partecipato a tutti i
-- concerti in un periodo assegnato

SELECT COUNT(DISTINCT id_sp)
FROM pr p0
WHERE not EXISTS
    (SELECT * -- i concerti nel periodo cercato per cui un
              --   determinato spettatore non ha prenotato
    FROM pr p1 JOIN co ON p1.id_co=co.id_co
    WHERE co.data BETWEEN '20020501' AND '20020631'
    AND not EXISTS
              (SELECT * -- le prenotazioni di un dato spettatore
                 --  per un dato concerto
               FROM pr p2
        WHERE p2.id_sp = p0.id_sp
               AND p1.id_co=p2.id_co
        )
    );
```

Persona(cf,nome,cognome,sesso,cf_padre,cf_madre,
citta_nascita,data_nascita)
fk cf_padre references Persona
fk cf_madre references Persona
fk citta_nascita references Citta

Citta(nome, nome_provincia, cap)
fk nome_provincia references Citta

Creazione dello schema

```
CREATE TABLE Citta(
  nome CHAR(32) PRIMARY KEY,
  nome_provincia CHAR(32),
  cap CHAR(5),
  FOREIGN KEY (nome_provincia) REFERENCES Citta );

CREATE TABLE Persona(
  cf CHAR(16) PRIMARY KEY,
  nome CHAR(32),
  cognome CHAR(32),
  sesso CHAR(1),
  cf_padre CHAR(16),
  cf_madre CHAR(16),
```

```
  citta_nascita CHAR(32),
  data_nascita CHAR(8),
  FOREIGN KEY (cf_padre) REFERENCES Persona,
  FOREIGN KEY (cf_madre) REFERENCES Persona,
  FOREIGN KEY (citta_nascita) REFERENCES Citta
);
```

Inserimento dati

```
INSERT INTO citta VALUES(     'Bologna',     'Bologna',     1      );
INSERT INTO citta VALUES(     'Rimini',      'Rimini',      2      );
INSERT INTO citta VALUES(     'Riccione',    'Rimini',      3      );
INSERT INTO citta VALUES(     'Imola',       'Bologna',     4      );
INSERT INTO citta VALUES(     'Forli',       'Forli',       5      );

INSERT INTO persona
  VALUES( 1, 'ugo', 'ughi', 'm', null, null, 'Bologna', '19010101' );
INSERT INTO persona
  VALUES( 2, 'lea', 'leoni', 'f', null, null, 'Rimini', '19020202' );
INSERT INTO persona
  VALUES( 3, 'mario', 'ughi', 'm', 1, 2, 'Bologna', '19320202' );
INSERT INTO persona
  VALUES( 4, 'ada', 'ughi', 'f', 1, 2, 'Bologna', '19330303' );
INSERT INTO persona
  VALUES( 5, 'ivo', 'ivi', 'm', null, null, 'Bologna', '19320202' );
INSERT INTO persona
  VALUES( 6, 'nerio', 'ivi', 'm', 5, 4, 'Bologna', '19620202' );
```

Query a.

Restituire per ogni persona il nome, cognome, provincia di nascita e il nome e cognome del padre.

Sono necessari due join, uno per collegare la persona alla città, e ricavarne la provincia, e uno per collegare la persona al padre, e ricavarne i dati a partire dal codice fiscale

```
SELECT P.cognome, P.nome, C.nome_provincia, PD.cognome, PD.nome
FROM Persona P
JOIN Citta C
  ON P.citta_nascita=C.nome
LEFT JOIN Persona PD
  ON P.cf_padre=PD.cf
```

Se al posto di "LEFT JOIN" si specifica "JOIN" non vengono visualizzate le persone di cui non si conosce il padre, ovvero per cui cf_padre IS NULL

Query b.

Restituire per ogni provincia il numero delle persone nate escludendo dal conteggio le persone nate nella stessa città del padre e della madre

Sono necessari tre join, uno per reperire le province di nascita, su cui fare il raggruppamento, e due per reperire la città di nascita dei genitori, su cui fare la selezione

```
SELECT C.nome_provincia, COUNT(*)
FROM Persona P
  JOIN Citta C  ON P.citta_nascita=C.nome
  JOIN Persona PD  ON PD.cf=P.cf_padre
  JOIN Persona MD  ON MD.cf=P.cf_madre
WHERE P.citta_nascita<>PD.citta_nascita
  OR P.citta_nascita<>MD.citta_nascita
GROUP BY C.nome_provincia
```

Query c.

Restituire la città con il maggior numero di nati

Equivale a cercare la città il cui numero di abitanti è maggiore o uguale al numero di nati di tutte le altre città. A questo scopo una query interna troverà il numero di nati di ogni città, mentre la query esterna troverà la città il cui conteggio è maggiore a quello di tutte le altre

```
SELECT citta_nascita
FROM Persona
GROUP BY citta_nascita
HAVING COUNT(*) >= ALL (
    SELECT COUNT(*)
    FROM Persona
    GROUP BY citta_nascita
    )
```

Creare il seguente schema relazionale

```
DIPARTIMENTI(IDDIP,UNIVERSITA,STATO)
PARTECIPANO(IDDIPA,IDPROG)
  FK IDDIP REFERENCES DIPARTIMENTI
  FK IDPROG REFERENCES PROGETTI
PROGETTI(IDPROG,TITOLO,ANNOINIZIO,ANNOFINE)
```

Scrivere le istruzioni SQL che producono i seguenti risultati:

a. Calcolare per ogni dipartimento e per ogni anno il numero di progetti già terminati a cui ha partecipato, escludendo i dipartimenti che hanno partecipato in totale* a meno di 3 progetti (suggerimento: in *totale significa tutti i progetti a cui un dipartimento ha partecipato senza distinzione di anno);

b. Elencare le università che non hanno mai partecipato a progetti già terminati (suggerimento: una data università non ha mai partecipato ad alcun progetto se non appartiene all'insieme delle università che hanno partecipato almeno una volta a un progetto terminato).

Scarica file MS Access con la soluzione

Si noti che il linguaggio SQL riconosciuto da MS Access ha qualche variazione rispetto allo standard

Creazione tabelle (SQL standard)

```
CREATE TABLE Dipartimenti(
  id_dip CHAR(16) PRIMARY KEY,
  universita CHAR(64) NOT NULL,
  stato CHAR(32)
);

CREATE TABLE Progetti(
  id_prog CHAR(16) PRIMARY KEY,
  titolo CHAR(256) NOT NULL,
  anno_in INTEGER NOT NULL,
  anno_fi INTEGER
);

CREATE TABLE Partecipano(
  id_dip CHAR(16) NOT NULL,
  id_prog CHAR(16) NOT NULL,
    PRIMARY KEY (id_dip,id_prog),
    FOREIGN KEY (id_dip) REFERENCES Dipartimenti,
    FOREIGN KEY (id_prog) REFERENCES Progetti
);
```

Inserimento dati

```
INSERT INTO Dipartimenti VALUES(      'IDD1', 'UNI1', 'STATO1'       );
INSERT INTO Dipartimenti VALUES(      'IDD2', 'UNI2', 'STATO2'       );
INSERT INTO Dipartimenti VALUES(      'IDD3', 'UNI1', 'STATO1'       );

INSERT INTO Progetti VALUES(   'IDP1', 'TITOLO1',     2004,   null    );
INSERT INTO Progetti VALUES(   'IDP2', 'TITOLO2',     1998,   2000    );
INSERT INTO Progetti VALUES(   'IDP3', 'TITOLO3',     2000,   null    );
INSERT INTO Progetti VALUES(   'IDP4', 'TITOLO4',     1999,   2002    );

INSERT INTO Partecipano VALUES( 'IDD1', 'IDP1 ' );
INSERT INTO Partecipano VALUES( 'IDD1', 'IDP2' );
INSERT INTO Partecipano VALUES( 'IDD1', 'IDP3 ' );
INSERT INTO Partecipano VALUES( 'IDD2', 'IDP1' );
INSERT INTO Partecipano VALUES( 'IDD2', 'IDP3' );
INSERT INTO Partecipano VALUES( 'IDD3', 'IDP4' );
```

Query a)

```
-- la query innestata come argomento di NOT IN e' necessaria per applicare
-- il vincolo sul numero di progetti a tutta la base di dati,
-- e non separatamente sui progetti di ogni anno, come avverrebbe
-- imponendo HAVING nella query esterna

SELECT id_dip, anno_in, COUNT(*) AS num_progetti
FROM Progetti Pr JOIN Partecipano Pa ON Pr.id_prog=Pa.id_prog
WHERE anno_fi IS NOT NULL
AND id_dip NOT IN
 (SELECT id_dip
  FROM Partecipano
  GROUP BY id_dip
  HAVING COUNT(*) <3)
GROUP BY id_dip, anno_in
```

Query b)

```
-- la clausola DISTINCT e' necessaria, poiche' l'attributo
-- universita' non e' chiave di Dipartimenti
-- la query innnestata trova le università i cui dipartimenti
-- partecipano a progetti conclusi

SELECT DISTINCT universita
FROM Dipartimenti
WHERE universita NOT IN
 (SELECT universita
  FROM Dipartimenti D JOIN Partecipano Pa ON D.id_dip = Pa.id_dip
                      JOIN Progetti Pr ON Pa.id_prog=Pr.id_prog
  WHERE  anno_fi IS NOT NULL)
```

Artisti scrivono canzoni

ARTISTI(IdArtista, Nome, Localita, Regione, Sesso)
PK(idArtista)

SCRIVONO(IdArtista, IdCanzone)
PK(IdArtista, IdCanzone)
FK IdArtista References ARTISTI
FK IdCanzone References CANZONI

CANZONI(IdCanzone, Nome, Durata, AnnoEdizione, Vendite)
PK(IdCanzone)

Creazione tabelle

```
CREATE TABLE Artisti(
 idArtista CHAR(16) PRIMARY KEY,
 nome CHAR(64) NOT NULL,
 localita CHAR(64),
 regione CHAR(64),
 sesso CHAR(1)
);

CREATE TABLE Canzoni(
 idCanzone CHAR(16) PRIMARY KEY,
 nome CHAR(64) NOT NULL,
 durata INTEGER,
 annoEdizione INTEGER,
 vendite INTEGER
);

CREATE TABLE Scrivono(
 idArtista CHAR(16) NOT NULL,
 idCanzone CHAR(16) NOT NULL,
 PRIMARY KEY (idArtista,idCanzone),
 FOREIGN KEY (idArtista) REFERENCES Artisti,
 FOREIGN KEY (idCanzone) REFERENCES Canzoni
)
```

Inserimento dati

```
INSERT INTO Artisti VALUES(    'A1',   'NOME1',        'L1',   'R1',   'F'     );
INSERT INTO Artisti VALUES(    'A2',   'NOME2',        'L2',   'R2',   'F'     );
INSERT INTO Artisti VALUES(    'A4',   'NOME4',        'L4',   'R3',   'M'     );
INSERT INTO Artisti VALUES(    'A3',   'NOME3',        'L3',   'R3',   'M'     );
INSERT INTO Canzoni VALUES(    'C1',   'N1',   4,      1999,   100     );

INSERT INTO Canzoni VALUES(    'C2',   'N2',   5,      2000,   200     );
INSERT INTO Canzoni VALUES(    'C3',   'N3',   4,      2000,   150     );
INSERT INTO Canzoni VALUES(    'C5',   'N5',   5,      1998,   200     );
INSERT INTO Canzoni VALUES(    'C4',   'N4',   3,      2001,   250     );

INSERT INTO Scrivono VALUES(   'A1',   'C1'    );
INSERT INTO Scrivono VALUES(   'A2',   'C1'    );
INSERT INTO Scrivono VALUES(   'A3',   'C3'    );
INSERT INTO Scrivono VALUES(   'A4',   'C2'    );
INSERT INTO Scrivono VALUES(   'A1',   'C5'    );
INSERT INTO Scrivono VALUES(   'A2',   'C5'    );
INSERT INTO Scrivono VALUES(   'A4',   'C3'    );
INSERT INTO Scrivono VALUES(   'A4',   'C4'    );
```

Query a)

Restituire i nomi degli artisti che hanno scritto almeno 5 canzoni tra il 2000 e il 2002, con una vendita complessiva superiore a 300

```
SELECT A.Nome, SUM(Vendite) AS totale_vendite
FROM ARTISTI AS A, SCRIVONO AS S, CANZONI AS C
WHERE A.IdArtista=S.IdArtista
AND S.IdCanzone=C.IdCanzone
AND C.AnnoEdizione Between 2000 AND 2002
GROUP BY A.idartista, a.nome
HAVING count(*)>=5 AND SUM(Vendite)>300;
```

Query b)

Restituire le coppie di nomi di artisti che hanno scritto almeno due canzoni insieme

Primariamente è necessario fare un self join della tabella Scrivono sull'attributo idCanzone, per associare gli artisti che risultano co-autori di una canzone, poi, per ricavare anche i nomi degli artisti, è necessario congiungere ogni "copia" di Scrivono con una "copia" di Artisti

```
SELECT a1.idartista, a2.idartista
FROM Artisti a1, Scrivono s1, Scrivono s2, Artisti a2
WHERE a1.idartista=s1.idartista
AND s1.idcanzone=s2.idcanzone
AND s2.idartista=a2.idartista
AND a1.idartista>a2.idartista -- serve ad evitare copie duplicate
GROUP BY a1.idartista, a2.idartista
HAVING COUNT(*)>=2
```

Persona(cf,nome,cognome,sesso,cf_padre,cf_madre,
citta_nascita,data_nascita)
fk cf_padre references Persona
fk cf_madre references Persona

Creazione dello schema

```
CREATE TABLE Persona(
  cf CHAR(16) PRIMARY KEY,
  nome CHAR(32),
  cognome CHAR(32),
  sesso CHAR(1),
  cf_padre CHAR(16),
  cf_madre CHAR(16),
  citta_nascita CHAR(32),
  data_nascita CHAR(8),
  FOREIGN KEY (cf_padre) REFERENCES Persona,
  FOREIGN KEY (cf_madre) REFERENCES Persona,
);
```

Inserimento dati

```
INSERT INTO persona
  VALUES( 1, 'ugo', 'ughi', 'm', null, null, 'Bologna', '19010101' );
INSERT INTO persona
  VALUES( 2, 'lea', 'leoni', 'f', null, null, 'Rimini', '19020202' );
INSERT INTO persona
  VALUES( 3, 'mario', 'ughi', 'm', 1, 2, 'Bologna', '19320202' );
INSERT INTO persona
  VALUES( 4, 'ada', 'ughi', 'f', 1, 2, 'Bologna', '19330303' );
INSERT INTO persona
  VALUES( 5, 'ivo', 'ivi', 'm', null, null, 'Bologna', '19320202' );
INSERT INTO persona
  VALUES( 6, 'nerio', 'ivi', 'm', 5, 4, 'Bologna', '19620202' );
INSERT INTO persona
  VALUES( 7, 'carlo', 'ivi', 'm', 5, null, 'Bologna', '19630202' );
INSERT INTO Persona
  VALUES( 8, 'emma', 'lupi', 'f', null, null, 'Imola', '19650202' );
INSERT INTO Persona
  VALUES( 9, 'marco', 'ivi', 'm', 6, 8, 'Bologna', '19860101' );
INSERT INTO Persona
  VALUES( 10, 'eva', 'ivi', 'f', 6, 8, 'Bologna', '19870101' );
```

Query a.

Restituire fratelli e sorelle di ogni persona, vale a dire i figli di suo padre o sua madre.

È necessario trattare separatamente i figli del padre e della madre, quindi occorrera una "union"; per quanto riguarda, ad esempio, i figli del padre, è necessario eseguire un "self join" di Persona, imponendo l'uguaglianza del padre e la differenza delle persone (per evitare il "fratello di se stesso")

Una coppia di fratelli comparirà due volte, con posizioni scambiate. Se due fratelli hanno in comune entrambi i genitori la clausola "union" evita che una coppia con le stesse posizioni venga duplicata, mentre con "union all" si avrebbe la duplicazione.

```
SELECT p.cognome, p.nome, p1.cognome, p1.nome
FROM Persona p, Persona p1
WHERE p.cf_padre=p1.cf_padre
AND p.cf<>p1.cf

UNION

SELECT p.cognome, p.nome, p1.cognome, p1.nome
FROM Persona p, Persona p1
WHERE p.cf_padre=p1.cf_padre
AND p.cf<>p1.cf
```

Query b.

Contare i figli di ogni coppia di genitori escludendo le coppie i cui genitori sono nati in città diverse e includendo solo le coppie con meno di 3 figli

Occorre usare Persona tre volte, una in rappresentanza della coppia, una per il padre e una per la madre. I predicati di join imporranno l'uguaglianza dei cf dei genitori nella tabella della coppia ai cf di padre e madre rispettivamente. La condizione sulla città di nascita si può applicare prima del raggruppamento, quella sul conteggio ovviamente dopo, con HAVING

```
SELECT c.cf_padre, c.cf_madre, COUNT(*) AS numeroFigli
FROM Persona p, Persona c, Persona m
WHERE
  p.cf=c.cf_padre AND m.cf=c.cf_madre -- predicati di join
AND
  p.citta_nascita=m.citta_nascita
GROUP BY c.cf_padre, c.cf_madre
HAVING COUNT(*)<3
```

Query c.

Restituire le coppie di genitori che hanno figli solo con una stessa persona

Occorre selezionare le tuple per cui non esiste un'altra persona con lo stesso padre e madre diversa, né con la stessa madre e padre diverso. Occorre anche escludere la coppia di genitori NULL NULL

```
SELECT DISTINCT cf_padre, cf_madre
FROM Persona p
WHERE NOT EXISTS (
  SELECT * FROM Persona p1
  WHERE p1.cf_padre=p.cf_padre
  AND (p1.cf_madre<>p.cf_madre
       OR p1.cf_madre IS NULL)
  )
AND NOT EXISTS (
 SELECT * FROM Persona p1
 WHERE p1.cf_padre<>p.cf_padre
 AND p1.cf_madre=p.cf_madre
 )
AND cf_padre IS NOT NULL
AND cf_madre IS NOT NULL
```

DOCENTE(IDDOC,POSIZIONE)

TITOLARE(IDCORSO,IDDOC)
FK IDCORSO REFERENCES CORSO
FK IDDOC REFERENCES DOCENTE

CORSO(IDCORSO, SETTORE, CREDITI, TIPO)

a) Calcolare per ogni docente il numero di corsi insegnati e il totale di crediti

b) Trovare i settori per i cui corsi non vi sono docenti con posizione ‘s’

Creazione dello schema

```
CREATE TABLE Corso(
   idCorso CHAR(16) PRIMARY KEY,
   settore CHAR(16) NOT NULL,
   crediti INTEGER NOT NULL,
   tipo CHAR(1)
   );
CREATE TABLE Docente(
   idDocente CHAR(16) PRIMARY KEY,
   posizione CHAR(1)
);
CREATE TABLE Insegna(
   idCorso CHAR(16) NOT NULL,
   idDocente CHAR(16) NOT NULL,
   PRIMARY KEY(idCorso,idDocente),
   FOREIGN KEY (idCorso) REFERENCES Corso,
   FOREIGN KEY (idDocente) REFERENCES Docente
);
```

Inserimento dati

```
INSERT INTO Corso VALUES('c1','s1',10, null);
INSERT INTO Corso VALUES('c2','s2',5, 'A');
INSERT INTO Corso VALUES('c3','s1',10, null);
INSERT INTO Corso VALUES('c4','s3',5, null);
INSERT INTO Corso VALUES('c5','s2',12, 'A');
INSERT INTO Corso VALUES('c6','s3',7, 'B');
INSERT INTO Docente VALUES('d1','i');
INSERT INTO Docente VALUES('d1','i');
INSERT INTO Docente VALUES('d2','i');
INSERT INTO Docente VALUES('d2','i');
INSERT INTO Docente VALUES('d2','i');
INSERT INTO Docente VALUES('d3','s');
INSERT INTO Insegna VALUES('c4','d1');
INSERT INTO Insegna VALUES('c6','d1');
INSERT INTO Insegna VALUES('c1','d2');
INSERT INTO Insegna VALUES('c2','d2');
INSERT INTO Insegna VALUES('c3','d2');
INSERT INTO Insegna VALUES('c5','d3');
INSERT INTO Insegna VALUES('c2','d3');
```

Query a)

Calcolare per ogni docente il numero di corsi insegnati e il totale di crediti

È una normale interrogazione di join fra corso e insegna, con raggruppamento, somma e conteggio

```
SELECT idDocente, COUNT(*) AS numeroCorsi,
       SUM(crediti) AS totaleCrediti
FROM Corso INNER JOIN Insegna
```

```
        ON Corso.idCorso=Insegna.idCorso
GROUP BY idDocente;
```

Query b)

Trovare i settori per i cui corsi non vi sono docenti con posizione ‘s’

Occorre trovare i settori dei corsi che non sono tra quelli insegnati da docenti con posizione 's', quindi occorre una query interna per trovare i corsi che non interessano, e una query esterna per trovare i corsi che non sono in quella lista. Poiché interessano i settori, e non i corsi, che non sono chiave di corso, sarà necessario eliminare i duplicati con "distinct"

```
SELECT DISTINCT settore
FROM Corso
WHERE idCorso NOT IN (
  SELECT idCorso
  FROM Insegna i INNER JOIN Docente d ON i.idDocente=d.idDocente
  WHERE posizione='s'
  );
```

PAZIENTE(IdPaziente, Nome, AnnoNascita, Sesso)

VISITA(IdPaziente, Data, Diagnosi)
FK IdPaziente -> PAZIENTE

RICH_ACCERTAM(IdPaziente, Data, TipoRichiesta, Risultato)
FK (IdPaziente, Data) -> VW_VISITA

A. Per ogni paziente dell'archivo si vuole conoscere il numero di richieste di accertamento di tipo A (incluso 0 per chi non ne ha);

B. Trovare il numero di richieste per ogni TipoRichiesta, tralasciando le richieste il cui numero è inferiore a 2

Creazione dello schema (SQL standard)

```
CREATE TABLE paziente (
        idpaziente char (2) NOT NULL PRIMARY KEY,
        nome char (20) NOT NULL ,
        annonascita integer NULL ,
        sesso char (1) NULL
);

CREATE TABLE visita (
        idpaziente char (2) NOT NULL ,
        data char (8) NOT NULL ,
        diagnosi char (5) NULL ,
         PRIMARY KEY (idpaziente,data),
         FOREIGN KEY (idpaziente) REFERENCES paziente
);

CREATE TABLE richaccertam (
        idpaziente char (2) NOT NULL ,
        data char (8) NOT NULL ,
        tiporichiesta char (1) NOT NULL ,
        risultato char (1) NULL ,
         PRIMARY KEY (idpaziente,data,tiporichiesta),
         FOREIGN KEY (idpaziente,data) REFERENCES visita
)
```

Query a)

Per ogni paziente dell'archivo si vuole conoscere il numero di richieste di accertamento di tipo A (incluso 0 per chi non ne ha)

Poiché la richiesta si riferisce a "ogni paziente dell'archivio" è necessario trovare sia il numero di richieste in questione per tutti i pazienti che hanno richieste di tale tipo, che tutti i pazienti che non hanno richieste di tale tipo, associando ad essi il numero 0; la soluzione richiede quindi l'unione di due select

```
select idpaziente, count(*) as 'Num. richieste "A"'
from richaccertam
where tiporichiesta = 'A'
group by idpaziente  -- conteggio per tutti i pazienti che
                     -- hanno richieste di tipo "A"

union

select idpaziente, 0
from paziente
where not exists (
 select * from richaccertam
 where tiporichiesta = 'A'
 and idpaziente = paziente.idpaziente
);                   -- tutti i pazienti che non hanno
                     -- richieste di tipo "A"
```

Query b)

Trovare il numero di richieste per ogni TipoRichiesta, tralasciando le richieste il cui numero è inferiore a 2

Si risolve con un semplice raggruppamento su TipoRichiesta e conteggio, applicando "having" per la selezione a posteriori, sui gruppi

```
select tiporichiesta, count(*) as 'Num. richieste'
from richaccertam
group by tiporichiesta
having count(*)>1
```