1 Database universitario

1.1 Elenco tabelle

PERSONA (**ID**, is_studente, is_docente, nome, cognome)
STUDENTE (**matricola**, persona references persona(ID))
DOCENTE (**codice**, persona references persona(ID))
CORSO (**ID**, nome)
OCCORRENZA_CORSO (**ID**, corso references corso(ID), anno)
ESAME (**corso** references occorrenza_corso(ID), **docente** references docente(codice), **studente** references studente(matricola), voto, data)

1.2 Testo esercizio

1. Calcolare la media ottenuta in Informatica Generale da chi non ha superato Matematica I.

2. Identificare gli studenti che hanno ottenuto un voto superiore alla media calcolata al punto 1.

3. Calcolare il numero massimo di volte che uno studente non ha superato l'esame di Informatica Generale

4. Determinare gli studenti interessati dalla soluzione del punto 3.

5. Calcolare quando uno studente ha superato un esame con un docente con cognome uguale allo studente.

6. Determinare per ogni docente e per ogni anno quanti esami il docente ha fatto in quel particolare anno e la percentuale di esami con successo sugli esami totali.

7. Per quali corsi esiste un anno in cui ci sono stati meno esami superati rispetto all'anno precedente?

8. Calcolare la media ottenuta in Informatica Generale da chi ha fallito almeno una volta Matematica I.

1.3 Interrogazioni

1. Un primo approccio prevede di utilizzare il costrutto `NOT IN`, mentre un secondo approccio è basato sull'operatore `EXCEPT`: si noti che quest'ultimo è applicabile solo a select conformi, ovvero il cui risultato abbia stesso schema. In entrambi i casi è fondamentale incrociare le tabelle `esame`, `occorrenza_corso`, `corso` per associare ad ogni esame il nome dell'insegnamento a cui si riferisce.

```
create view esami_superati as
select matricola, nome, cognome, voto, corso
from studente, esame, occorrenza_corso, corso, persona
where persona.ID=studente.persona AND
  occorrenza_corso.corso=corso.ID AND
```

```
  esame.corso=occorrenza_corso.ID AND
  esame.studente=studente.matricola AND
  voto IS NOT NULL;

select avg(voto)
from esami_superati
where corso='Informatica Generale' AND
matricola NOT IN (
  select matricola
  from esami_superati
  where corso='Matematica I'
);
```

Nel secondo caso decido di utilizzare EXCEPT al fine di calcolare gli studenti che hanno superato Informatica Generale ma non Matematica I.

```
create view infgen_no_mate(matricola) as (
select studente
from esame
where corso='Informatica Generale' AND voto is not null

except

select studente
from esami, occorrenza_corso, corso
where nome='Matematica I' and
      esami.corso=occorrenza_corso.id and
          occorrenza_corso.corso=corso.id and
          voto is not null
)

select avg(voto)
from esami, occorrenza_corso, corso, infgen_no_mate
where nome='Informatica Generale' and
      esami.corso=occorrenza_corso.id and
          occorrenza_corso.corso=corso.id and
          esami.studente=infgen_no_mate.matricola
```

2. ```
   select nome, cognome
   from esami_superati
   where corso='Informatica Generale' AND
   voto > ALL (select avg(voto)
       from esami_superati
       where corso='Informatica Generale' AND
       matricola NOT IN (
         select matricola
   ```
   ```

```
      from esami_superati
      where corso='Matematica I'
    )
);
```

oppure

```
create view esame_infgen_no_mate(matricola, voto) as (
        select esami.studente, esami.voto
    from esami, occorrenza_corso, corso, infgen_no_mate
    where nome='Informatica Generale' and
      esami.corso=occorrenza_corso.id and
          occorrenza_corso.corso=corso.id and
          esami.studente=infgen_no_mate.matricola and
          voto is not null
)

select matricola
from esame_infgen_no_mate
where esami.voto >= all (
                            select avg(voto)
                        from esame_infgen_no_mate)
```

3.
```
create view infgen_nonsup as
select matricola, count(*) as numero
from esame, occorrenza_corso, corso
group by matricola
where occorrenza_corso.corso=corso.ID AND
  esame.corso=occorrenza_corso.ID AND
  corso.nome='Informatica Generale' AND
  voto IS NULL;

select max(numero)
from infgen_nonsup;
```

4.
```
select matricola, nome, cognome
from infgen_nonsup join persone on persone.matricola= infgen_nonsup.matricola
where numero >= ALL
    (select numero
    from infgen_nonsup
);
```

5.
```
select *
from studente, esame, occorrenza_corso, corso, persona D, persona S, docente
where S.ID=studente.persona AND
  D.ID=docente.persona AND
  esame.docente=docente.codice AND
```

```
esame.corso=occorrenza_corso.ID AND
esame.studente=studente.matricola AND
voto IS NOT NULL AND
S.cognome=D.cognome;
```

6. ```
   create view esamiripartiti as
   select codice, count(*) as totali, anno
   from esame
   group by codice. anno

   create view esamiripartitipassati as
   select codice, count(*) as totali, anno
   from esame
   where voto is not null
   group by codice. anno

   select *, passati/totali as percentuale
   from esamiripartitipassati join esamiripartiti
   where
   esamiripartitipassati.anno = esamiripartiti.anno and
   esamiripartitipassati.codice = esamiripartiti.codice
   ```

7. ```
   select distinct corso
   from esamiripartitipassati A, esamiripartitipassati B,
   where
   A.anno < B.anno and
   A.corso = B.corso and
   A.totali > B.totali
   ```

2 Biblioteca

2.1 Testo esercizio

Si desidera modellare i dati relativi alla gestione di una biblioteca. Ogni libro viene identificato univocamente dal codice ISBN, ed è caratterizzato dal fatto di avere uno o più autori (non verranno effettuate ricerche sugli autori), titolo, codice dewey e genere.

Gli utenti hanno un codice identificativo e viene conservato nome, cognome e numero di telefono (quest'ultimo dato è facoltativo). Per ogni libro preso in prestito segnamo la data di inizio prestito, la data di scadenza e la data in cui è stato effettivamente restituito (quest'ultimo dato viene posto a NULL se il libro non è ancora stato restituito).

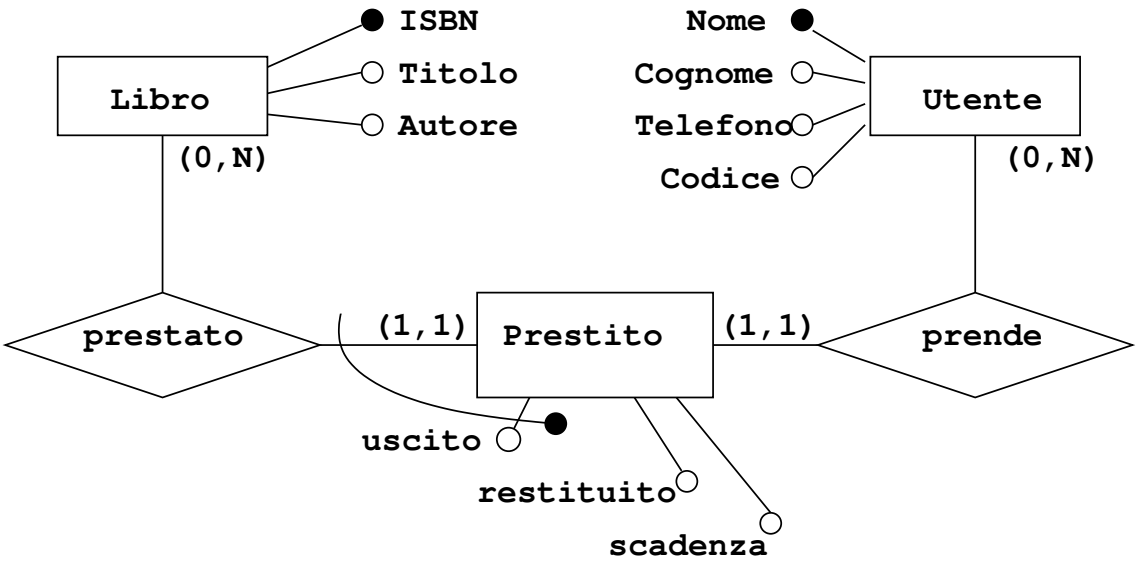
Modellare la situazione descritta con modello ER. Descrivere la query SQL, e le tabelle coinvolte in tale query, che restituisce l'elenco delle persone che, per almeno 3 volte, non hanno restituito un libro entro la scadenza. Tale esercizio deve essere risolto secondo due versioni: la prima considera soltanto il fatto che ci siano state tre restituzioni in ritardo, la seconda anche il fatto che alla data odierna ci potrebbero essere dei libri in prestito anche se la data di scadenza è già passata.

Come si modifica lo schema ER se non si ammette la possibilità che un utente prenda in prestito lo stesso libro più di una volta?

Se avessimo il vincolo che un utente può avere al massimo 5 libri in prestito contemporaneamente, come avremmo codificare tale vincolo?

Si calcoli quale libro è stato restituito in ritardo più volte.

2.2 ER



Si è deciso di utilizzare un'entità per Prestito, invece di una relazione, perchè si ritiene rilevante rappresentare i prestiti di un libro dismesso dalla libreria, ciò in quanto viene richiesto di conteggiare le persone che hanno consegnato un libro in ritardo per almeno 3 volte. Questo fatto presuppone che Prestito abbia valore a prescindere dall'esistenza di Libro, e quindi un'entità è preferibile. Si noti che il vincolo che il numero di telefono sia opzionale è rappresentabile solo con il codice SQL di creazione della tabella.

2.3 Tabelle

LIBRO (**ISBN**, Autore, Titolo, Dewey, Genere)
UTENTE (**codice**, nome, cognome, numerotel)
PRESTITO (**utente** references utente(codice), **libro** references libro(ISBN), **scadenza**, restituito, inizio)

2.4 Query

```
select codice, nome, cognome
from utente join prestito on utente(codice)=prestito(utente)
where codice in
    (select codice
    from prestito
    where restituito>scadenza
    group by codice
```

```
    having count(*)>=3);
```

La versione completa segue:

```
select utente, nome, cognome
from utente join prestito on utente(codice)=prestito(utente)
where utente in
    (select utente
    from prestito
    where (restituito>scadenza or today()>scadenza and restituito is null)
    group by utente, nome, cognome
    having count(*)>=3);
```

3 Rete ferroviaria

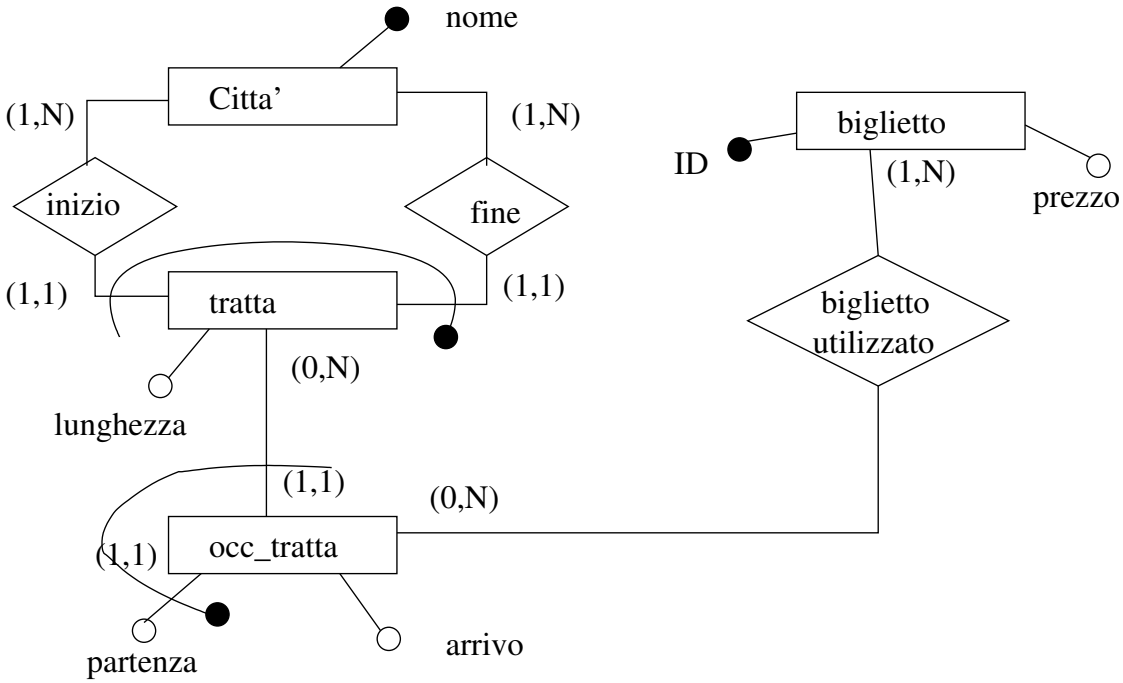
3.1 Testo

Si vuole rappresentare l'insieme di dati riguardanti una rete ferroviaria. Ogni città viene identificata univocamente tramite il proprio nome, ed ogni tratta consiste in una città di partenza, una città di arrivo ed una lunghezza epressa in km. Si suppone che non esistano stazioni intermedie fra le due città che compongono una tratta.

Ogni biglietto emesso ha un numero identificativo progressivo ed ha un prezzo, dipendente dalla massima percorrenza possibile (tale percorrenza non viene memorizzata). Per ogni volta che una tratta viene percorsa si tiene traccia dell'orario di partenza e dell'orario di arrivo. Al momento della convalida del biglietto viene immesso l'insieme delle tratte che compongono il viaggio per cui il biglietto viene utilizzato e data/ora di partenza e arrivo del viaggio.

Si scriva la query SQL che determina su quale tratta sono stati utilizzati più biglietti nel mese di Ottobre 2001.

3.2 ER



Ogni tratta viene univocamente determinata dalle città di arrivo e partenza. Inoltre l'introduzione dell'entità `occorrenza_tratta` permette di non duplicare l'informazione relativa all'orario di partenza e arrivo da associare ad ogni biglietto: infatti la relazione `biglietto_utilizzato` permette di individuare tale informazione.

3.3 Tabelle

CITTA'(**nome**)
TRATTA(**inizio** references citta'.nome, **fine** references citta'.nome, lunghezza)
BIGLIETTO(prezzo, **ID**)

OCCORRENZA_TRATTA(**inizio** references citta'.nome, **fine** references citta'.nome, **partenza**, arrivo)
BIGLIETTO_UTILIZZATO(**ID** references biglietto, **inizio** references citta'(nome), **fine** references citta'(nome), **partenza** references occorrenza_tratta)

3.4 Query

```
create view usati_ott as
select count(*) as quanti, inizio, fine
from biglietto_utilizzato
where partenza <= '31/10/2000' and partenza >= '01/10/2000'
group by partenza, arrivo;

select inizio, fine
from usati_ott
where quanti >= ALL
    (select quanti
     from usati_ott);
```

4 Campionato di calcio

4.1 Testo

Un campionato di calcio viene disputato tra un insieme di squadre. Ogni incontro è svolto fra due squadre, una detta ospitante ed una detta ospite. Per ogni incontro, oltre alle due squadre coinvolte, vengono registrati il risultato (inteso come numero di reti segnate dalle ognuna delle due squadre), la data ed i giocatori che hanno realizzato ogni rete, si memorizza anche il minuto della partita in cui è stata realizzata la rete.

In un campionato ogni squadra incontra ogni altra squadra due volte, di cui una come squadra ospite ed una come squadra ospitante.

Di ogni calciatore viene registrato nome, cognome e data di nascita. Si assume inoltre che un giocatore non possa cambiare squadra durante il campionato.

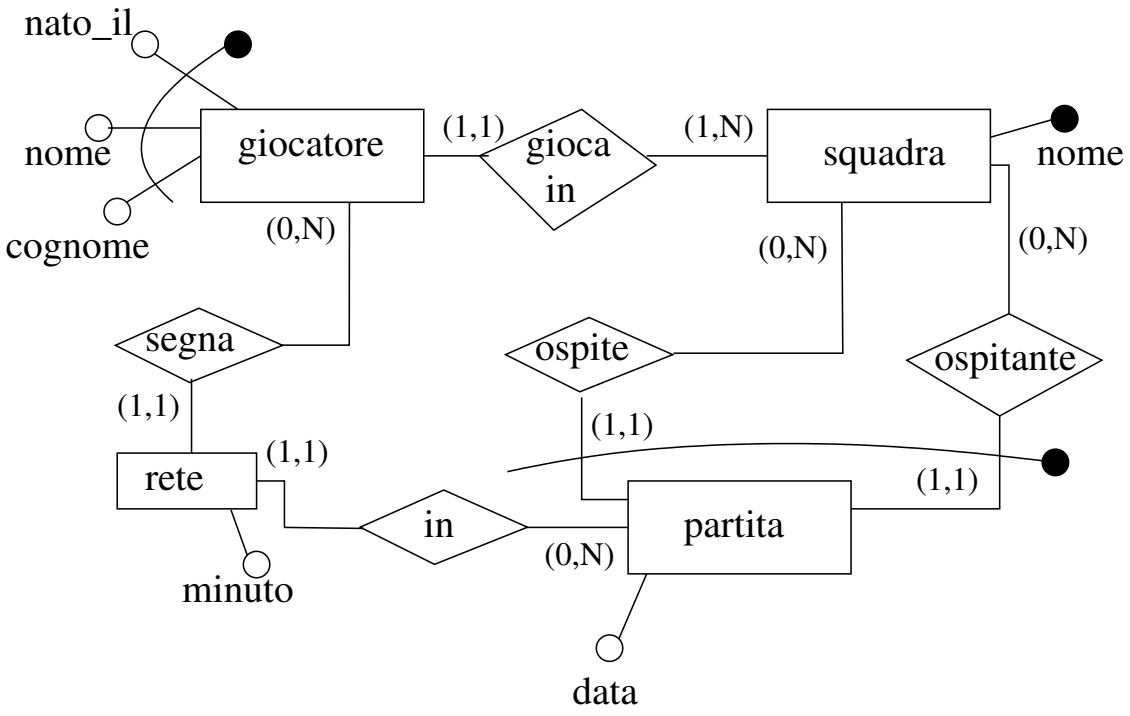
Si progetti una base di dati per rappresentare la situazione sopra descritta, sia come modello ER che come insieme di tabelle.

Si scriva una query SQL che restituisce quale squadra ha vinto più partite, una query per determinare il calciatore che ha segnato più reti ed una query per determinare in quale partita si è realizzato il numero maggiore di reti (somma fra le reti della squadra ospite e quelli della squadra ospitante).

Come viene modificato lo schema se una squadra può incontrare un'altra squadra più di due volte?

Come viene modificato lo schema se un giocatore può cambiare squadra durante il campionato?

4.2 ER



4.3 Tabelle

SQUADRA (nome)
PARTITA (ospite references squadra(nome), ospitante references squadra(nome), data)
GIOCATORE (nome, cognome, natoil, giocain references squadra(nome))
RETE (nome references giocatore, cognome references giocatore, data_di_nascita references giocatore, ospite references squadra(nome), ospitante references squadra(nome), minuto)

4.4 Query

```
create view retipartitasquadra as
  select ospite, ospitante, giocain as squadra, count(*) as quanti
  from partita, rete, giocatore
  where giocatore.nome =rete.nome and
        giocatore.cognome =rete.cognome and
        giocatore.natoil =rete.natoil and
        partita.ospite = rete.ospite and
        partita.ospitante = rete.ospitante
  group by partita.ospite, partita.ospitante, giocatore.gioca_in


create view risultati as
  select R1.quanti as retiospitante, R2.quanti as retiospite
  from retipartitasquadra R1, retipartitasquadra R2
  where R1.ospite = R2.ospite and
        R1.ospitante = R2.ospitante and
        R1.squadra <> R2.squadra
        R1.ospitante = R1.squadra



create view vittorie as
   select ospite as squadra
   from partita
   where reti_ospite>reti_ospitante
 union all
   select ospitante as squadra
   from partita
   where reti_ospite<reti_ospitante

select nome
from vittorie
group by squadra
having count(*) > ALL (
    select count(*)
    from vittorie
    group by squadra);
```

4.5 Varie

`Rete` deve essere una entità, in quanto è ragionevolmente atteso che un giocatore segni più di una rete in una partita, pertanto l'attributo `minuto` deve essere parte della chiave.

Se il numero di partite fra una coppia di squadre non è limitato, allora `data` deve fare parte della chiave dell'entità `partita`.

Se un giocatore può cambiare squadra, allora la cardinalità della relazione `gioca in` non può più essere $(1, 1)$. In questo caso deve anche diventare una tabella a sè stante. Inoltre non è più possibile inferire il risultato dalla relazione rete, in quanto non si può stabilire a favore di quale squadra sia stata realizzata una rete. Diventa quindi necessario (1) introdurre una relazione `partecipa` che lega le entità `giocatore` e `partita`, oppure (2) introdurre una relazione `a favore` che lega le entità `rete` e `squadra`. Dalle numerosità ragionevolmente attese, quest'ultima ipotesi mi sembra preferibile.

5 Servizio di fornitura di energia elettrica

5.1 Testo

Viene fornita energia elettrica ad un insieme di utenze, con la possibilità che nuove utenze vengano aggiunte nel tempo. Per ogni utenza registriamo, oltre ad un codice identificativo, nome, cognome e l'attivazione dell'utenza. Quando un contratto viene attivato, si registra un codice identificativo, la data in cui sono stati completati i lavori ed il tecnico responsabile dell'impianto.

Per ogni utenza vengono inoltre memorizzati i dati storici relativi ad ogni lettura del contatore. Dopo ogni lettura viene emessa una fattura di cui memorizziamo la lettura, prima e dopo il periodo di riferimento della fattura stessa, la data di emissione e l'intestatario. Nel caso in cui non sia stata possibile la lettura, la fattura viene emessa senza avere una lettura di riferimento.

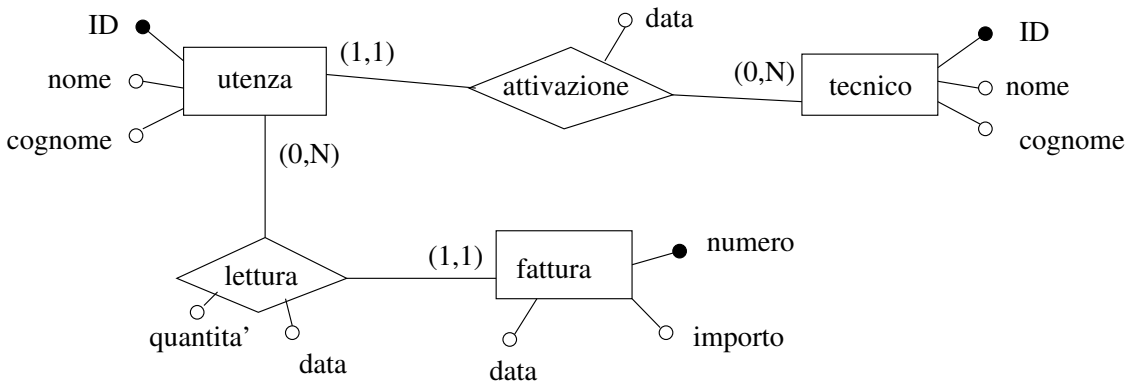
Disegnare lo schema ER e le tabelle.

Query 1: l'ultimo importo pagato da ogni utente di cognome Rossi.

Query 2: le letture da mettere in fattura del 01.07.2001 del cliente con codice 003 (tale fattura ha una lettura di riferimento).

Modificare lo schema per incorporare la seguente modifica: l'attivazione avviene tramite una sequenza di sopralluoghi effettuati presso la stessa utenza. Ogni sopralluogo viene fatto da un tecnico potenzialmente diverso. L'attivazione avviene quando l'ultimo sopralluogo verifica che sono verificate le condizioni per l'attivazione.

5.2 ER



5.3 Tabelle

UTENZA (nome, cognome, ID, attivazione unique, data_attivazione, tecnico references tecnico(ID))
TECNICO (nome, cognome, ID)
FATTURA (numero, data, importo, quantita', utente references utenza(ID) null, data_lettura null)

5.4 Query

```
create view fattrossi as
```

```
    select ID, importo, data_fattura
    from utenza join fattura on utenza.ID=fattura.utente
    where cognome='Rossi';

select importo, data
from fattrossi F
where data >= ALL
    (select data
     from fattrossi F1
     where f.numero=f1.numero);
```

```
select lettura
from fattura
where data<'01/01/2001' AND
  data >= ALL (
    select data
    from fattura
    where data<'01/01/2001');
```

6 Conferenza scientifica

6.1 Testo

Una conferenza viene organizzata nel seguente modo: uno (o più) autori sottopongono un articolo affinchè venga valutato e, alla fine del processo di valutazione, accettato oppure rifiutato alla conferenza. Ad ogni articolo viene assegnato un numero progressivo, e si tiene traccia del titolo e degli autori. Per ogni autore si memorizzano il nome, il cognome e l'indirizzo di posta elettronica che identifica univocamente ogni autore.

La conferenza ha un comitato direttivo i cui membri possono (ma non sono obbligati) sottoporre articoli. Ogni articolo viene valutato da almeno 3 ed al più 5 revisori, di cui esattamente uno deve essere membro del comitato direttivo. Si noti che non tutti i membri del comitato direttivo devono essere revisori.

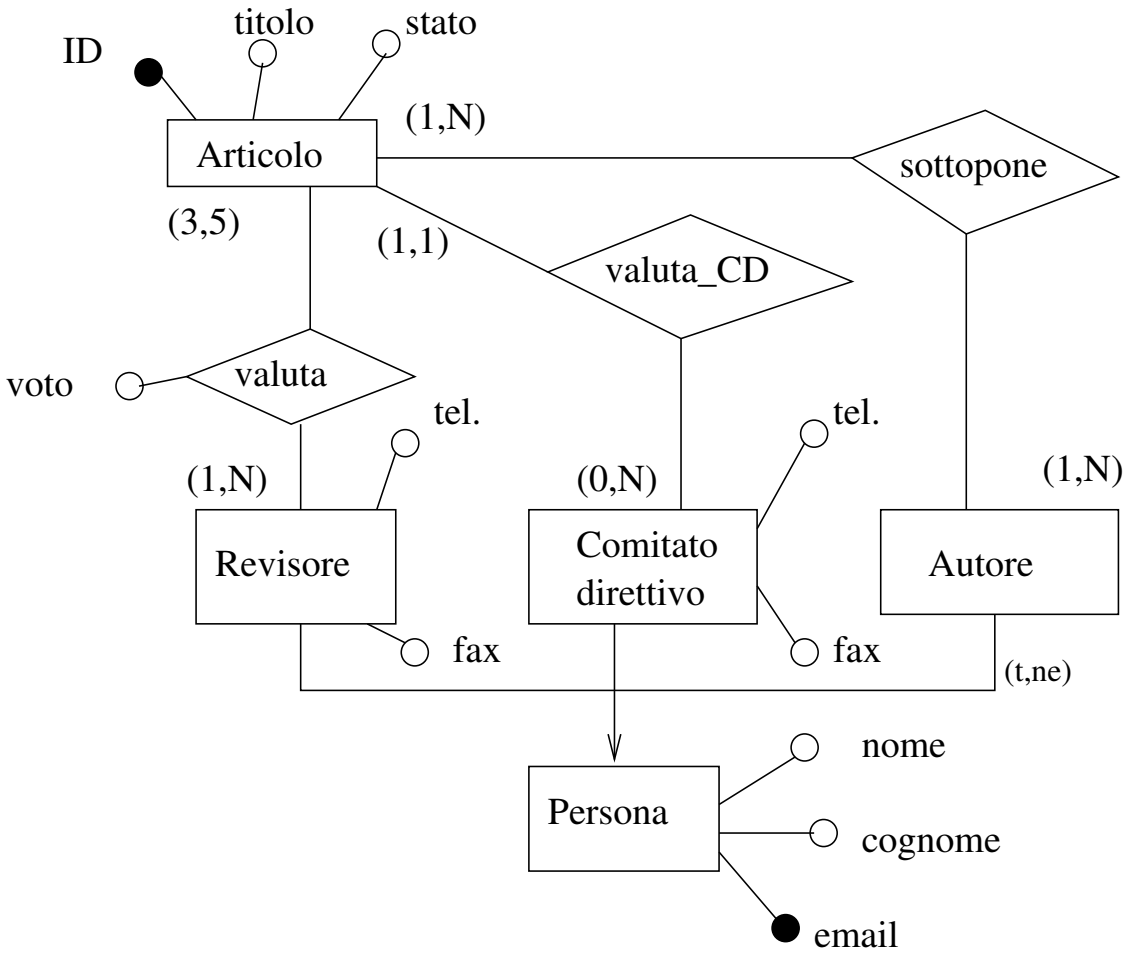
Nessuno può essere revisone di un articolo di cui è autore. Al termine della revisione, ogni revisore decide un voto da assegnare agli articoli valutati. Solo alcuni articoli vengono accettati alla conferenza, conseguentemente lo stato di ogni articolo è accettato, oppure rifiutato, oppure ancora in revisione. Di ogni revisore o membro del comitato direttivo vengono memorizzati numero di telefono e fax, oltre ai dati identificativi di un autore.

Descrivere modello ER, tabelle e codice SQL per le sequenti interrogazioni.

1. Quale autore ha avuto il massimo numero di articoli accettati?
2. Quali revisori non hanno ancora inviato la loro valutazione?
3. Visualizzare la media delle valutazioni di ogni articolo.
4. Visualizzare la media delle valutazioni degli articoli scritti da ogni singola persona.
5. Visualizzare i 20 articoli con la media più alta.

Come si modifica lo schema se i membri del comitato direttivo non possono essere autori di articoli sottoposti alla conferenza?

6.2 ER



6.3 Tabelle

ARTICOLO (<u>ID</u>, titolo, stato)
PERSONA (nome, cognome, <u>email</u>, telefono, fax, is_CD)
SOTTOPONE (lavoro REFERENCES artiolo(ID), autore REFERENCES autore(email))
VALUTA (revisore REFERENCES persona(email), lavoro REFERENCES artiolo(ID))

6.4 SQL

1.

```
create view lavori_accettati as
  select id, titolo, nome, cognome
  from articolo, sottopone, autore
  where articolo.ID=sottopone.lavoro AND
```

```
     sottopone.autore=autore.email AND
    accettato;

select email, nome, cognome
from lavori_accettati
group by email, nome, cognome
having count(*) >= ALL
  (select count(*)
  from lavori_accettati
  group by email);
```

2.

```
select distinct nome, cognome, email
from persona join valuta on persona.email=valuta.revisore
where voto is null;
```

3.

```
select avg(voto), titolo, ID
from articolo join valuta on articolo.ID=valuta.articolo
group by ID;
```

4.

```
select avg(voto), nome, cognome, email
from valuta, sottopone, articolo
where articolo.ID=sottopone.lavoro AND
    valuta.lavoro=articolo.ID
group by email;
```

7 Catena di cinema

7.1 Testo

Una catena di cinema ha sale in varie città. Di ogni cinema si memorizza la città e la via in cui si trova. Alcuni cinema sono multisala, mentre altri contengono un'unica sala. Di ogni sala si memorizza la capienza totale.

Ogni film ha un titolo (si assuma che non esistano due film con stesso titolo), l'anno di produzione un regista e vari attori. Per ogni persona esiste un codice identificativo.

Viene inoltre tenuta traccia di ogni proiezione effettuata, ovvero del film proiettato, in quale sala è avvenuta la registrazione, ed il numero di biglietti venduti.

Disegnare lo schema ER e le tabelle.

Query 1: l'elenco dei film in cui il numero dei biglietti totali venduti è stato almeno l'80% della capienza totale (sommata su tutte le proiezioni del film).

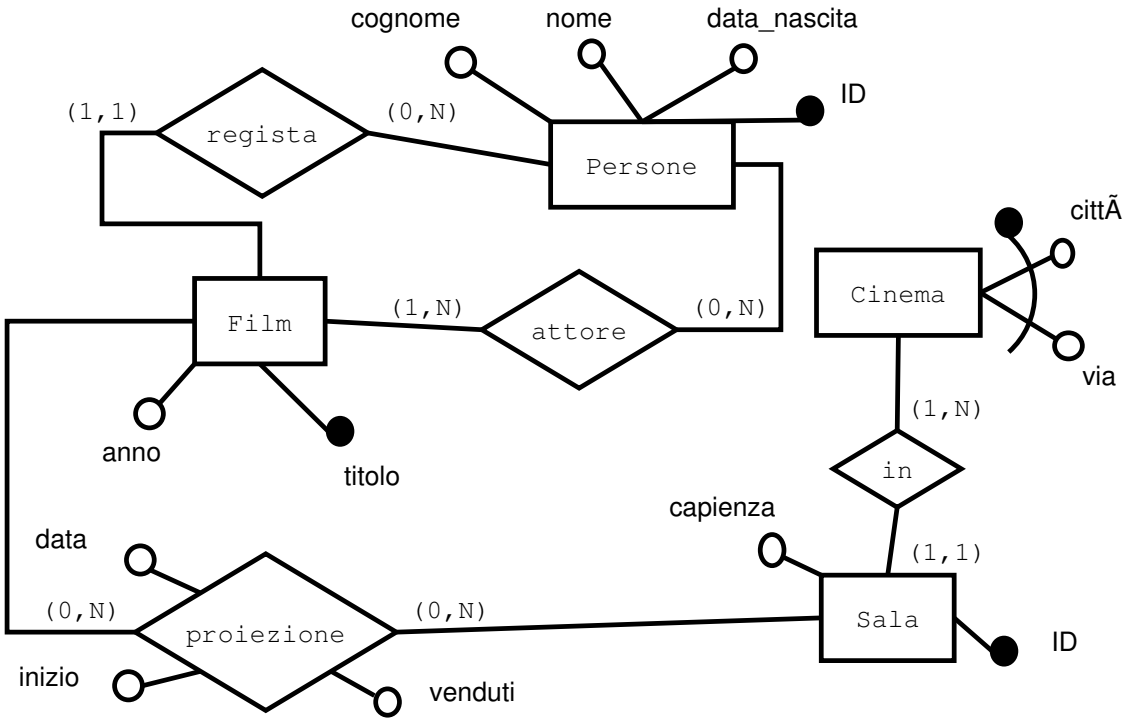
Query 2: l'elenco dei film che hanno fatto il tutto esaurito in almeno due spettacoli tenuti nella stessa città.

Query 3: l'elenco degli registi che si sono diretti in un film.

Query 4: I film che hanno venduto più biglietti.

Query 5: l'elenco dei film che hanno fatto il tutto esaurito in almeno due città.

7.2 ER



7.3 Tabelle

PERSONE (<u>ID</u>, cognome, nome, data_nascita)
FILM (<u>titolo</u>, regista references persone(ID))

CINEMA (città, via)
SALA (ID, citta references CINEMA(citta), via references CINEMA(via), capienza)
PROIEZIONE (film references FILM(titolo), citta_sala references CINEMA(citta), sala references SALA(ID), inizio, data)
ATTORE (persona references PERSONA(ID), film references FILM(titolo))

7.4 SQL

Query 1.

```
create view venditeproiezione as
  select titolo, sum(capienza) as capienzatot, sum(venduti) as vendutitot
  from proiezione, sala
  where sala.ID=proiezione.sala
  group by titolo;

select titolo
from venditeproiezione
where vendutitot/capienzatot>=0.8;
```

Query 2.

```
create view esauriti as
  select titolo, città, via, count(*) as numero
  from proiezione, sala
  where sala.città=proiezione.città AND
      sala.ID=proiezione.sala AND proiezione.venduti=sala.capienza;
  group by titolo, città

select titolo
from esauriti
where numero>=2;
```

Query 3.

```
select nome, cognome
from persone, attore, film
where film.regista=attore.persona AND
    attore.film=film.titolo AND attore.persona=persone.id;
```

Query 4.

```
select titolo
from venditeproiezione
where vendutitot >= ALL (
select vendutitot
from venditeproiezione);
```

Query 5.

```
select E1.titolo
from esauriti E1, esauriti E2
where E1.titolo=E2.titolo and
      E1.città <> E2.città
```

Licenza d'uso