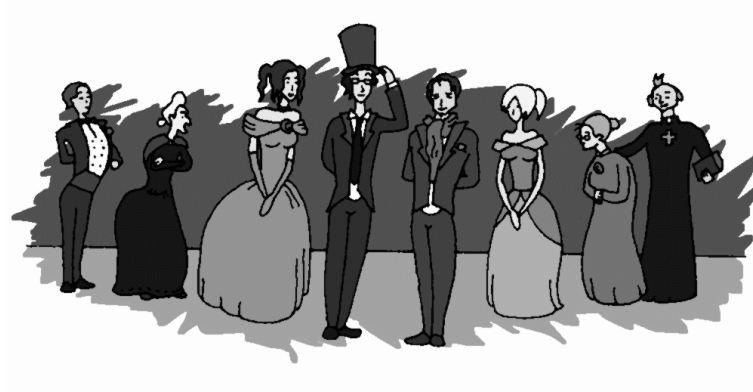


Database Nugae Teatrali

Bruni Alessandro

19 giugno 2007



Indice

1	Analisi dei requisiti	2
1.1	Più nel dettaglio...	2
2	Schema concettuale ad oggetti	3
2.1	Modello senza attributi:	3
2.2	Modello con attributi:	4
3	Modello dei dati relazionale	4
4	Implementazione MySQL	6
4.1	DDL: La definizione del modello relazionale in SQL	6
4.2	DML: Le interrogazioni al database	9

1 Analisi dei requisiti

Il database delle *Nugae Teatrali*, compagnia teatrale stabile, vuole gestire le informazioni della compagnia riguardanti:

- le opere messe in scena;
- gli attori e i membri della compagnia;
- gli oggetti di scena necessari alle rappresentazioni;
- i ricavi delle rappresentazioni;
- le spese per l'acquisto o il noleggio di oggetti di scena, costumi, infrastrutture.

La compagnia è un'associazione di persone senza scopo di lucro, riunitesi dall'interesse per il teatro. Essa è composta da individui classificati nelle seguenti tre categorie: membri, membri fondatori (coloro che hanno partecipato fin dall'inizio alle attività della compagnia), collaboratori esterni.

Membri e membri fondatori sono soggetti all'obbligo di adempimento di determinati compiti all'interno della compagnia, e si dividono in *attori* e persone che hanno un *ruolo amministrativo/di supporto* all'interno del gruppo, anche se alcuni attori possono ricoprire cariche amministrative.

Degli attori interessa il profilo, mentre di chi ha un ruolo all'interno della compagnia si vuole annotare la descrizione del ruolo che svolge.

Di ogni *opera* che viene messa in scena, la compagnia vuole tener traccia dei dati relativi all'opera, compresi il *titolo*, il nome dell'*autore*, i *personaggi* presenti, gli *oggetti di scena* e i rispettivi proprietari, una *descrizione* dell'opera e un link al file pdf del copione, se disponibile in rete. Un'opera può essere rappresentata in luoghi e momenti differenti, e di ogni *rappresentazione* si vuole poter registrare il luogo e la data, quali attori hanno preso parte e in quale/i ruolo/i, chi è stato addetto alla gestione delle luci, chi ha curato la colonna sonora, chi si è preso cura degli oggetti di scena e così via.

Una volta avvenuta una rappresentazione, il gruppo vuole poter registrare l'*incasso* di tale rappresentazione e, se possibile, il numero di persone che hanno assistito a tale messa in scena (*pubblico*).

Dal momento che si tratta di un'associazione senza scopo di lucro, il gruppo intende, con gli incassi, pagare le *spese* per l'*acquisto* degli oggetti di scena, il *noleggio* e/o l'*acquisto* di luci e impianto sonoro e conservare, se possibile, dei *preventivi* per eventuali futuri acquisti di apparecchiature utili allo scopo della compagnia.

1.1 Più nel dettaglio...

Il database conterrà un'insieme di *persone*, sia membri che persone esterne al gruppo teatrale, di cui si vuole sapere il nome, il cognome, la data di nascita e se appartengono o meno al gruppo (Gli oggetti di scena, ad esempio, possono appartenere a persone esterne al gruppo, e se ne vuole memorizzare il proprietario). Ogni persona che è membro del gruppo teatrale, può essere o meno un *attore*, o avere uno o più *ruoli* amministrativi/di supporto all'interno del gruppo. Non è necessario che tutti i membri siano anche attori, e nulla vieta agli attori di ricoprire altri ruoli all'interno del gruppo, ma chi è membro e non è attore, deve necessariamente avere un ruolo specifico.¹

¹Nello schema ad oggetti gli attori saranno rappresentati come una sottoclasse delle persone, mentre i ruoli amministrativi/di supporto come una nuova entità a parte, ruoli, che indicherà quali persone hanno un ruolo.

Per quanto riguarda il controllo d'integrità referenziale nel database, sarà implementata una procedura di correzione che controllerà che ogni membro sia un attore o abbia almeno un ruolo e, nel caso ci siano membri che non hanno né un ruolo né sono attori, renda attori tali persone, nel caso ci siano persone che hanno un ruolo ma non sono membri, li renda automaticamente membri. Attenzione: non necessariamente un attore deve avere già preso parte ad una rappresentazione del gruppo: può essere semplicemente un nuovo membro.

Di ogni *opera* messa in scena dalla compagnia teatrale, si vuole sapere il titolo, il nome dell'autore, una descrizione e un link al copione di riferimento (se presente in internet). In un'opera possono comparire più personaggi, che hanno un nome e un ruolo, e si dividono in principali e secondari. Un'opera viene messa in scena (rappresentata) in luoghi e date differenti, e di ogni *rappresentazione* si vuole sapere la data, il *luogo* (composto da nome della località, via, città e provincia) e, una volta che una rappresentazione è avvenuta, si vuole poter memorizzare il numero di persone che l'hanno vista (pubblico) ed eventuali incassi e/o spese riguardanti la messa in scena.

Dato che possono avvenire delle modifiche alle *parti* assegnate agli attori fra una rappresentazione e l'altra della stessa opera, si vuole inoltre poter memorizzare, per ogni rappresentazione, quale attore ha interpretato quale personaggio di tale opera. Inoltre, sempre per ogni rappresentazione, si deve registrare il nome di chi ha svolto ruoli di supporto pratico (tecnico audio, luci etc.) necessari alla messa in scena.

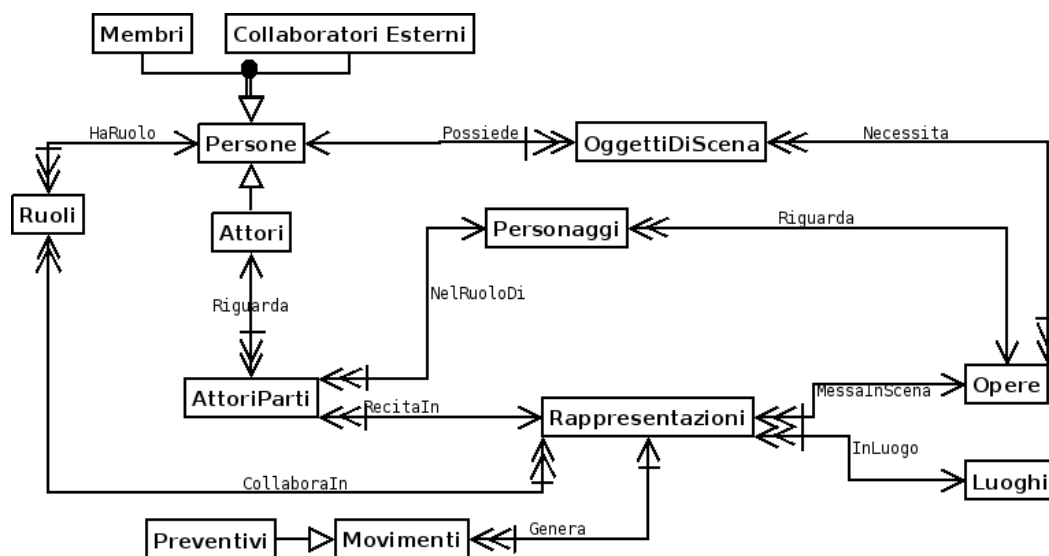
Per quanto riguarda i ricavati e le spese necessarie per la messa in scena, si vuole sapere a quanto ammonta l'importo di denaro speso/guadagnato (negativo se si tratta di una spesa, ovvero denaro uscente, positivo se si tratta di un introito), la data e il motivo (descrizione) per cui tale denaro è entrato/uscito dalla cassa del teatro. Possono inoltre essere memorizzati dei preventivi, che non influenzano però il saldo della cassa della compagnia.

Ultima informazione importante è costituita dagli *oggetti di scena*, che sono posseduti da persone registrate nel database, e che si vuole poter contattare nel caso questi oggetti servano per una nuova rappresentazione. Di ogni oggetto si vuole sapere il nome, una descrizione ed il proprietario, che è unico. Un oggetto di scena può essere utile per più opere e, viceversa, un'opera richiede certamente più oggetti di scena.

2 Schema concettuale ad oggetti

Gli schemi presentati qui sotto seguono il formalismo visto a lezione²:

2.1 Modello senza attributi:



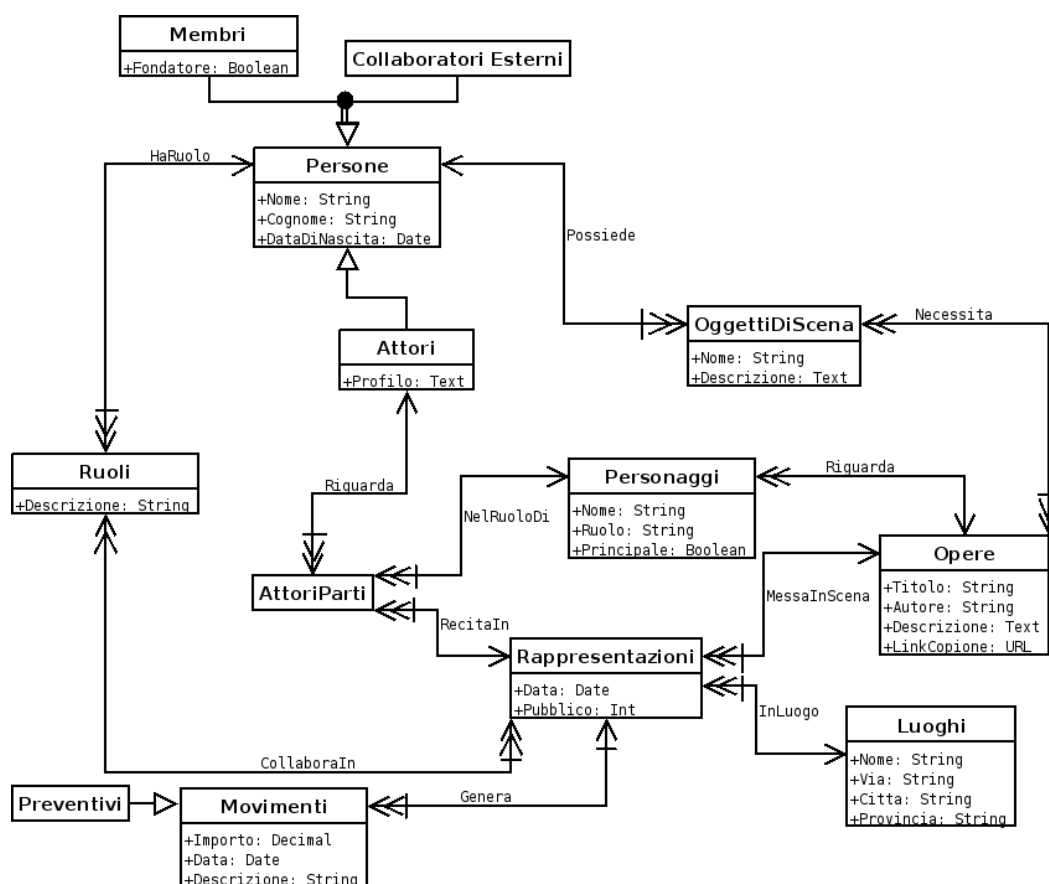
²Nel progetto vi è una relazione ternaria, quella tra Attori, Personaggi e Rappresentazioni, che non è stata trattata a lezione. Per la sua rappresentazione ho scelto di creare una nuova entità, AttoriParti, che mette in relazione le tre classi.

Sulle relazioni parziali. Lo schema non presenta relazioni particolari, come relazioni ricorsive, relazioni totali uno a uno, o altre relazioni non rappresentabili a meno di perdite di vincoli nel modello relazionale: principalmente l'unico vincolo che può risultare difficile da gestire è il subclassing degli attori da persone e la sua relazione con le persone che hanno ruoli (rispettando le richieste dell'analisi dei requisiti), ma tale vincolo sarà gestito da una procedura di correzione, come si vedrà più avanti.

La gran parte delle relazioni è uno a molti, con la parzialità nel senso in cui la relazione è molteplice, e ciò si realizza inserendo delle chiavi esterne dalla parte della molteplicità con attributo not null. Dove non c'è nemmeno questo vincolo, ovvero tra Movimenti e Rappresentazioni, è sufficiente tralasciare l'attributo not null del campo della chiave esterna.

Un altro vincolo che non è rappresentabile è quello totale in entrambi i sensi tra Personaggi e Opere, ma ciò non rappresenta un problema, anzi risulta comodo per l'inserimento di nuove opere.

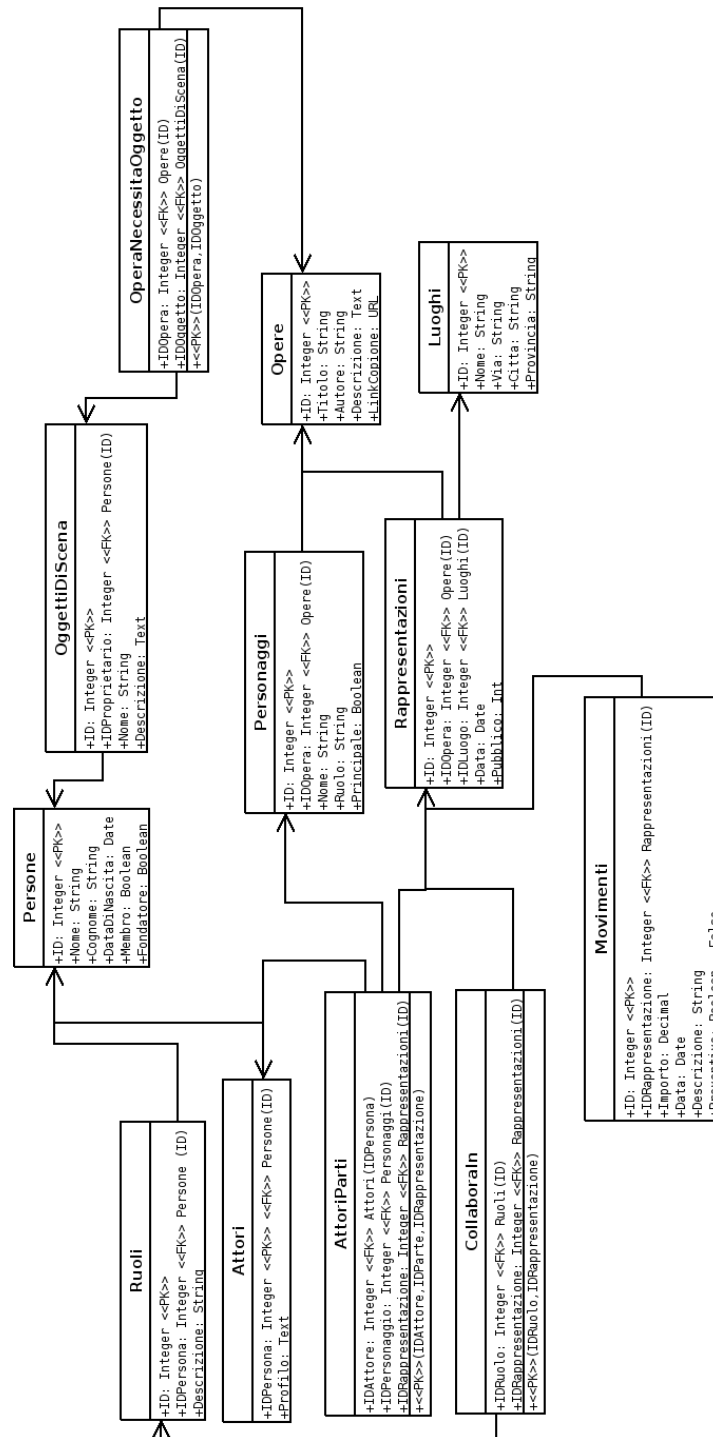
2.2 Modello con attributi:



3 Modello dei dati relazionale

Per la traduzione dal modello concettuale ad oggetti al modello relazionale si è scelto di inserire, come chiave primaria, un codice identificativo di ogni entità e, dove necessario, assegnargli l'attributo auto_increment. Le relazioni molti a molti del modello ad oggetti sono state tradotte con nuove tabelle

del modello relazionale che hanno per chiave primaria l'insieme degli attributi, e per chiavi esterne degli attributi che si riferiscono ai codici identificativi dei record nelle tabelle da mettere in relazione.



4 Implementazione MySQL

4.1 DDL: La definizione del modello relazionale in SQL

Il seguente blocco di codice definisce, in MySQL, il modello relazionale appena prodotto³:

```
drop table if exists Movimenti, OperaNecessitaOggetto,
  OggettiDiScena, CollaboraIn, AttoriParti, Rappresentazioni
  , Luoghi, Personaggi, Opere, Attori, Ruoli, Persone;
drop view if exists Attori_v, Mansioni, AttoriSenzaMansioni,
  AttoriConMansioni;

create table Persone (
  ID integer primary key auto_increment,
  Nome varchar(20) not null,
  Cognome varchar(20) not null,
  DataDiNascita date,
  Membro boolean not null,
  Fondatore boolean not null check (not Fondatore or
    Membro),
  unique (Nome, Cognome)
) engine=InnoDB;

create table Attori (
  IDPersona integer primary key,
  Profilo text,
  foreign key fk_Persone (IDPersona) references Persone
    (ID)
) engine=InnoDB;

create view Attori_v as
select p.*, a.Profilo
from Persone p join Attori a on a.IDPersona = p.ID;

create table Ruoli (
  ID integer primary key auto_increment,
  IDPersona integer not null,
  Descrizione varchar(40) not null,
  foreign key (IDPersona) references Persone(ID)
) engine=InnoDB;

create view AttoriSenzaMansioni as
select a.*
from Attori_v a
```

³Nota bene: dato che lo scopo del database è quello di gestire lo storico dello sviluppo dell'attività del gruppo teatrale, piuttosto che trattarne semplicemente lo stato attuale, i vincoli di chiave esterna non hanno azioni specificate per la modifica o l'eliminazione di record all'interno del database, in quanto si assume che, una volta creati, essi non vengano né cancellati né modificati, almeno nei loro attributi di chiave primaria, che per scelta di progettazione sono degli interi con auto incremento.

```

where a.ID not in (select r.IDPersona from Ruoli r);

create view AttoriConMansioni as
select a.*, r.Descrizione
from Attori_v a, Ruoli r
where a.ID = r.IDPersona;

create table Opere (
  ID integer primary key auto_increment,
  Titolo varchar(100) not null,
  Autore varchar(100) not null,
  Descrizione longtext,
  LinkCopione varchar(255) check (LinkCopione regexp '
    (^[http://]{1}([^\s:]*):[a-z]{1}[:
    alnum:]+\(\)/[[:alnum]#?/&=]$\)')
) engine=InnoDB;

create table Personaggi (
  ID integer primary key auto_increment,
  IDOpera integer not null,
  Nome varchar(40) not null,
  Ruolo varchar(40),
  Principale boolean not null,
  unique (IDOpera, Nome),
  foreign key fk_Opera (IDOpera) references Opere (ID)
) engine=InnoDB;

create table Luoghi (
  ID integer primary key auto_increment,
  Nome varchar(100) not null,
  Via varchar(100),
  Citta varchar(40) not null,
  Provincia varchar(10) not null
) engine=InnoDB;

create table Rappresentazioni (
  ID integer primary key auto_increment,
  IDOpera integer not null,
  IDLuogo integer not null,
  Data date not null,
  Pubblico integer check ((Data > Now() and Pubblico is
    null) or
    (Data <= Now() and Pubblico is not null)),
  foreign key fk_Opera (IDOpera) references Opere (ID),
  foreign key fk_Luogo (IDLuogo) references Luoghi (ID)
) engine=InnoDB;

create table CollaboraIn (
  IDRuolo integer,

```

```

        IDRappresentazione integer,
        primary key (IDRuolo, IDRappresentazione),
        foreign key (IDRappresentazione) references
            Rappresentazioni(ID),
        foreign key (IDRuolo) references Ruoli(ID)
    ) engine=InnoDB;

create table AttoriParti (
    IDAttore integer,
    IDPersonaggio integer,
    IDRappresentazione integer,
    primary key (IDAttore, IDRappresentazione,
        IDPersonaggio),
    foreign key (IDAttore) references Attori (IDPersona),
    foreign key (IDPersonaggio) references Personaggi (ID
    ),
    foreign key (IDRappresentazione) references
        Rappresentazioni (ID)
    ) engine=InnoDB;

create table OggettiDiScena (
    ID integer primary key auto_increment,
    IDProprietario integer not null,
    Nome varchar(100) not null,
    Descrizione text,
    foreign key (IDProprietario) references Persone(ID)
    ) engine=InnoDB;

create table OperaNecessitaOggetto (
    IDOpera integer not null,
    IDOggetto integer not null,
    primary key (IDOpera, IDOggetto),
    foreign key (IDOpera) references Opere (ID),
    foreign key (IDOggetto) references OggettiDiScena(ID)
    ) engine=InnoDB;

create table Movimenti (
    ID integer primary key auto_increment,
    IDRappresentazione integer,
    Importo decimal not null,
    Data date not null,
    Descrizione text not null,
    Preventivo boolean not null,
    foreign key (IDRappresentazione) references
        Rappresentazioni(ID)
    ) engine=InnoDB;

```

4.2 DML: Le interrogazioni al database

Prima di tutto avviene la popolazione del database. A tal proposito ho creato il file "popolazione.sql", che inserisce nel database delle istanze valide per il progetto, in gran parte prese dalle informazioni reali della compagnia.

Di seguito invece sono riportate alcune interrogazioni utili da fare al database:

1. Seleziona le coppie di attori (Nome, Cognome, Data in cui ha rappresentato) che hanno recitato la stessa parte in due rappresentazioni diverse della stessa opera, di cui si vuole sapere il titolo e l'autore:

```
SELECT a1.Nome, a1.Cognome, r1.Data AS DataRappresentazione, a2.
  Nome, a2.Cognome, r2.Data AS DataRappresentazione, p1.Nome AS
  Personaggio, o.Titolo, o.Autore
FROM
```

```
Attori_v a1
JOIN AttoriParti ap1 ON a1.ID = ap1.IDAttore
JOIN Personaggi p1 ON ap1.IDPersonaggio = p1.ID
JOIN Rappresentazioni r1 ON ap1.IDRappresentazione = r1.ID,
```

```
Attori_v a2
JOIN AttoriParti ap2 ON a2.ID = ap2.IDAttore
JOIN Personaggi p2 ON ap2.IDPersonaggio = p2.ID
JOIN Rappresentazioni r2 ON ap2.IDRappresentazione = r2.ID,
```

```
Opere o
```

```
WHERE o.ID = p1.IDOpera
AND o.ID = p2.IDOpera
AND p1.ID = p2.ID
AND a1.ID > a2.ID;
```

2. Seleziona i nomi degli attori di ogni prima rappresentazione, in ennuple della forma (Nome Cognome, Personaggio, Nome Opera) ordinando tali ennuple per nome dell'opera e mettendo per primi i personaggi principali di ogni opera:

```
SELECT a.Nome, a.Cognome, p.Nome AS Personaggio, o.Titolo AS
  TitoloOpera
```

```
FROM Attori_v a
JOIN AttoriParti ap ON a.ID = ap.IDAttore
JOIN Personaggi p ON ap.IDPersonaggio = p.ID
JOIN Opere o ON p.IDOpera = o.ID
```

```
WHERE ap.IDRappresentazione <= ALL (
  SELECT r.ID
  FROM Rappresentazioni r
  WHERE IDOpera = o.ID
) ORDER BY o.Titolo, p.Principale DESC;
```

3. Ottieni una lista degli oggetti di scena che servono per la prossima rappresentazione, con i rispettivi proprietari da contattare e la data per cui questi oggetti devono essere a disposizione del gruppo:

```
SELECT p.Cognome, p.Nome, os.Nome AS NomeOggetto, os.Descrizione,
        o.Titolo, r.Data AS DataDisponibilita
```

```
FROM Rappresentazioni r
JOIN Opere o ON r.IDOpera = o.ID
JOIN OperaNecessitaOggetto ono ON o.ID = ono.IDOpera
JOIN OggettiDiScena os ON ono.IDOggetto = os.ID
JOIN Persone p ON os.IDProprietario = p.ID
```

```
WHERE r.Data >= Now( )
AND r.Data <= ALL (
    SELECT DATA FROM Rappresentazioni
    WHERE DATA >= Now( )
) ORDER BY p.Cognome, p.Nome;
```

4. Ottieni nome dell'opera, bilancio, data e luogo della rappresentazione che e' stata più redditizia, ovvero il cui bilancio tra spese e introiti relativi a tale rappresentazione è maggiore di tutte le altre:

```
DROP VIEW IF EXISTS BilancioRappresentazioni;
```

```
CREATE VIEW BilancioRappresentazioni AS
SELECT Sum( m.Importo ) AS Bilancio, r.ID AS IDRappresentazione
FROM Movimenti m
RIGHT JOIN Rappresentazioni r ON m.IDRappresentazione = r.ID
WHERE m.Preventivo = False GROUP BY r.ID;
```

```
SELECT br.Bilancio, r.Data, l.Nome AS NomeLuogo, l.Via, l.Citta,
        l.Provincia, o.Titolo AS NomeOpera
FROM BilancioRappresentazioni br
JOIN Rappresentazioni r ON br.IDRappresentazione = r.ID
JOIN Luoghi l ON r.IDLuogo = l.ID
JOIN Opere o ON r.IDOpera = o.ID
WHERE br.Bilancio >= ALL (
    SELECT br2.Bilancio
    FROM BilancioRappresentazioni br2
);
```

5. Seleziona la media di soldi donati per persona (si assume che gli introiti del gruppo collegati alle rappresentazioni siano derivati solo dalle donazioni libere del pubblico), dove sono stati donati soldi:

```
SELECT m.Importo/r.Pubblico as DonazioneMedia, r.Data, l.Nome AS
        NomeLuogo, l.Via, l.Citta, l.Provincia, o.Titolo AS NomeOpera
FROM Movimenti m
JOIN Rappresentazioni r ON m.IDRappresentazione = r.ID
JOIN Luoghi l ON r.IDLuogo = l.ID
JOIN Opere o ON r.IDOpera = o.ID
WHERE r.Data<Now() AND m.Importo>0;
```

6. Seleziona gli attori che hanno recitato solo in parti secondarie (Attenzione: qui possono apparire anche attori che non hanno ancora recitato):

```

SELECT a.*
FROM Attori_v a
WHERE NOT EXISTS (
    SELECT *
    FROM AttoriParti ap
    JOIN Personaggi p ON ap.IDPersonaggio = p.ID
    WHERE p.Principale = True
    AND ap.IDAttore = a.ID
);

```

7. Versione dell'interrogazione precedente con solo gli attori che hanno recitato (NB: questa modifica non produce alcun effetto nell'istanza del progetto, per il solo fatto che sono stati inseriti solo attori che hanno recitato, e non attori nuovi):

```

SELECT a.*
FROM Attori_v a
WHERE NOT EXISTS (
    SELECT *
    FROM AttoriParti ap
    JOIN Personaggi p ON ap.IDPersonaggio = p.ID
    WHERE p.Principale = True AND ap.IDAttore = a.ID
) AND EXISTS (
    SELECT *
    FROM AttoriParti ap
    JOIN Personaggi p ON ap.IDPersonaggio = p.ID
    WHERE p.Principale = False AND ap.IDAttore = a.ID
);

```

8. Seleziona i luoghi nei quali e' stata fatta o e' in programma più di una rappresentazione

```

SELECT l.* , Count(*) AS NumRappresentazioni
FROM Luoghi l
JOIN Rappresentazioni r ON l.ID = r.IDLuogo
GROUP BY l.ID
HAVING Count(*)>1;

```

9. Ottieni ID, Titolo, Autore dell'opera che richiede il maggior numero di oggetti di scena.

```

SELECT o.ID, o.Titolo, o.Autore
FROM Opere o JOIN OperaNecessitaOggetto ono ON o.ID = ono.IDOpera
GROUP BY o.ID
HAVING Count(*)>=ALL (
    SELECT Count(*)
    FROM OperaNecessitaOggetto oo
    GROUP BY oo.IDOpera
);

```

10. Procedura per correggere il problema di integrità referenziale di cui si è parlato nell'analisi dei requisiti⁴:

```
DROP PROCEDURE IF EXISTS correggi ;

DELIMITER $$
CREATE PROCEDURE correggi ()
BEGIN
    DECLARE idMembro INT;
    DECLARE numIterazioni INT;

    /* Rende attori tutti i membri che non hanno alcun ruolo */
    SELECT Count(*) INTO numIterazioni FROM Persone p
    WHERE p.Membro = True AND NOT EXISTS (
        SELECT * FROM Attori a WHERE a.IDPersona = p.ID
    ) AND NOT EXISTS (
        SELECT * FROM Ruoli r WHERE r.IDPersona = p.ID
    );

    WHILE numIterazioni > 0 DO
        SELECT p.ID INTO idMembro FROM Persone p
        WHERE p.Membro = True AND NOT EXISTS (
            SELECT * FROM Attori a WHERE a.IDPersona = p.ID
        ) AND NOT EXISTS (
            SELECT * FROM Ruoli r WHERE r.IDPersona = p.ID
        ) LIMIT 1;

        INSERT INTO Attori VALUES (idMembro, '.');
        SET numIterazioni = numIterazioni - 1;
    END WHILE;

    /* Rende membri le persone che hanno un ruolo all'interno
    della compagnia */
    SELECT Count(*) INTO numIterazioni FROM Persone p
    WHERE p.Membro = False AND p.ID IN (SELECT DISTINCT r.
        IDPersona FROM Ruoli r);

    WHILE numIterazioni > 0 DO
        SELECT p.ID INTO idMembro FROM Persone p
        WHERE p.Membro = False AND p.ID IN (SELECT DISTINCT r.
            IDPersona FROM Ruoli r) LIMIT 1;

        UPDATE Persone p SET p.Membro = True WHERE p.ID = idMembro;
        SET numIterazioni = numIterazioni - 1;
    END WHILE;
END$$
DELIMITER ;
```

⁴Attenzione: la procedura non si carica se richiamata da file, va copiato il codice e incollato nella shell MySQL.