Relazione Progetto Basi di Dati

Streaming di film e gestione recensioni

~

Marco Cavalli - X81000445

Indice

DESCRIZIONE DELLE SPECIFICHE DELLA BASE DI DATI

Una ditta del campo dell'intrattenimento chiede una base di dati che permetta di gestire contratti di streaming della propria piattaforma online. La ditta dispone di un catalogo dei film. Ciascun film presente sul catalogo è identificato da un id numerico. Nel catalogo sono inoltre specificati il titolo del film, gli artisti che vi hanno preso parte, ovvero il regista e gli attori principali (indicando la loro nazionalità e il personaggio interpretato nel film), l'anno di produzione e il genere. Ciascun film ha inoltre una breve descrizione della trama. I film possono essere anche in lingua originale (diversa dall'italiano). In tal caso nel catalogo è detto di quale lingua si tratta, e se il film è sottotitolato. I contratti per vedere i film in streaming devono prevedere diversi tipi di abbonamenti (mensile, trimestrale, annuale) con costi commisurati alla durata del contratto (rispettivamente 7, 20, 60 euro). Prevedere un metodo per rinnovare automaticamente i contratti in scadenza che prevedono rinnovo automatico. I tipi di contratto devono contenere la durata, un nome descrittivo e il costo. Ogni cliente può vedere qualunque film sul catalogo tranne quelli usciti negli ultimi dodici mesi. Per poter vedere i film più recenti i clienti dovranno sottoscrivere la versione premium del servizio. La versione premium comporta dei benefici come maggiore qualità dei video in streaming e la possibilità di collegarsi all'account su più dispositivi al costo di una spesa aggiuntiva pari a 3 euro per ogni mese di abbonamento. I clienti sono caratterizzati dal cognome, nome, data di nascita, e-mail ed eventualmente il telefono, indirizzo di residenza e genere (maschile, femminile). Infine deve prevedere un metodo che permetta ai soli utenti registrati di lasciare una recensione dei film presenti sul catalogo. La recensione deve essere sia testuale che grafica (con uso di stelle o altri simboli per indicare il voto riassuntivo della recensione stessa).

ANALISI DELLE SPECIFICHE

DIZIONARIO DEI TERMINI

Termine	Descrizione	Sinonimo	Termini collegati
Film	Contiene le informazioni sui film come artisti, genere, anno produzione	Prodotto	Artista, Recensione
Artista	Ha partecipato in un film		Film
Regista	Ha diretto un film		Artista
Attore	Ha recitato in un film		Artista
Film in lingua originale	Particolari film che non hanno l'audio in lingua italiana		Film
Contratto	Contiene le informazioni sulla durata e sui tipi di film che un cliente può vedere		Cliente
Cliente	Utenti registrati che fruiscono dei film in streaming	Utente registrato	Contratto, Recensione
Recensione	Descrizione qualitativa (grafica e testuale) di un cliente di un film che ha visionato		Cliente, Film

Abbiamo quindi già dato una definizione di quelli che sono i termini principali delle specifiche. Ci accingiamo ora a stilare un modello E/R attraverso l'uso di una strategia di progettazione e indicare eventuali vincoli d'integrità e derivazione.

PROGETTAZIONE CONCETTUALE

Per muoverci utilizzeremo una strategia di tipo **TOP-DOWN** partendo da uno schema scheletro e raffinandolo via via applicando le seguenti regole:

R1. Si applica quando un'entità descrive due concetti diversi legati fra loro.

R2. Si applica quando un'entità è composta da sotto-entità distinte.

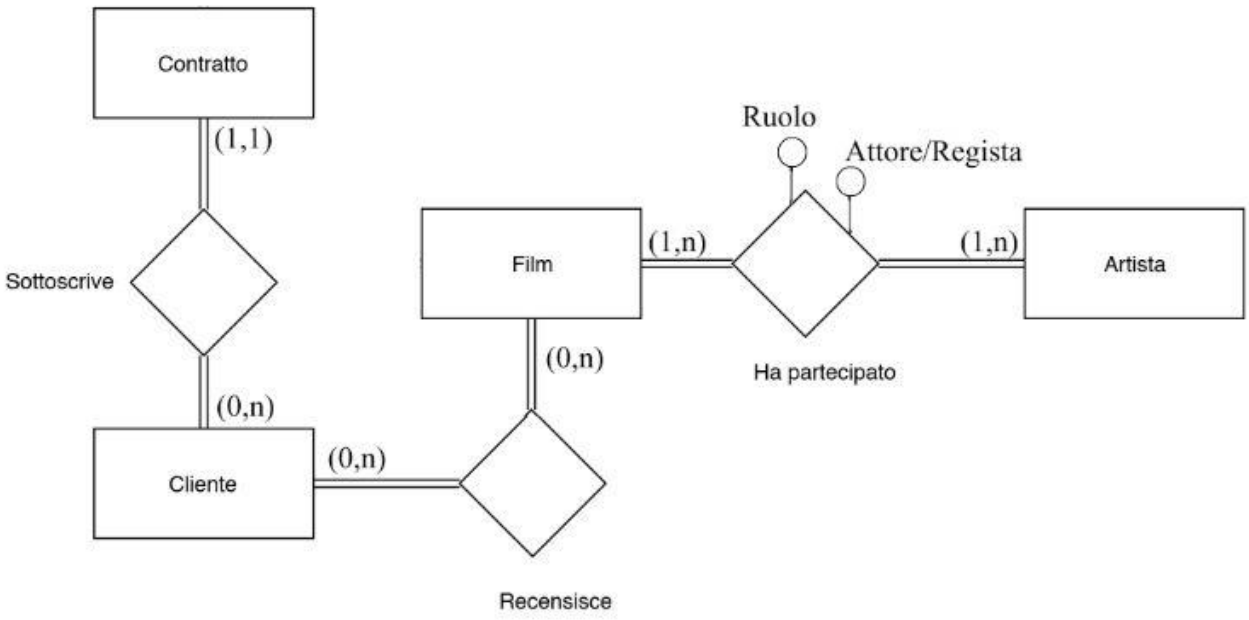
R3. Si applica quando una relazione descrive due diverse relazioni tra le stesse entità.

R4. Si applica quando una relazione esprime un concetto con esistenza autonoma.

R5. Si applica per aggiungere attributi ad entità.

R6. Si applica per per aggiungere attributi a relazioni.

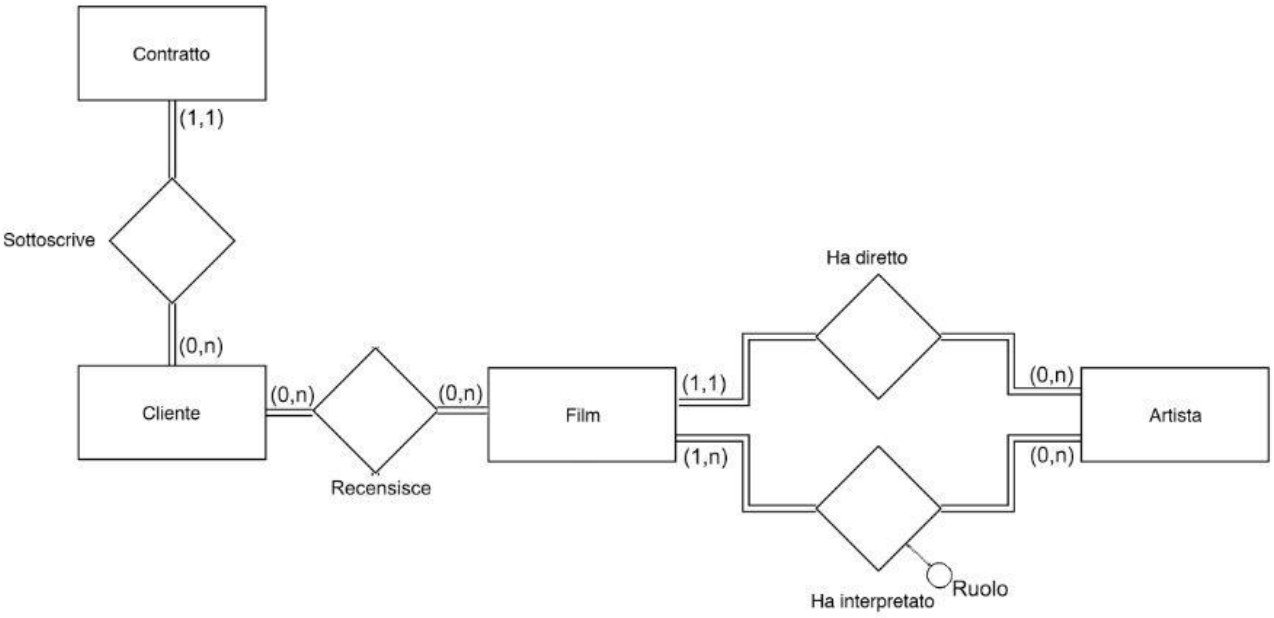
Inizieremo quindi dal seguente **schema scheletro.**



Come si può vedere questo schema ben descrive il cuore della base di dati che vogliamo raffigurare, ma vanno aggiunte diverse cose e vanno sistemati diversi problemi (sia legati ad un'astrazione non precisa dei concetti che a problemi di consistenza dei dati e preservazione delle informazioni in caso di modifica/inserimento/cancellazione).

VERSO LO SCHEMA FINALE #1

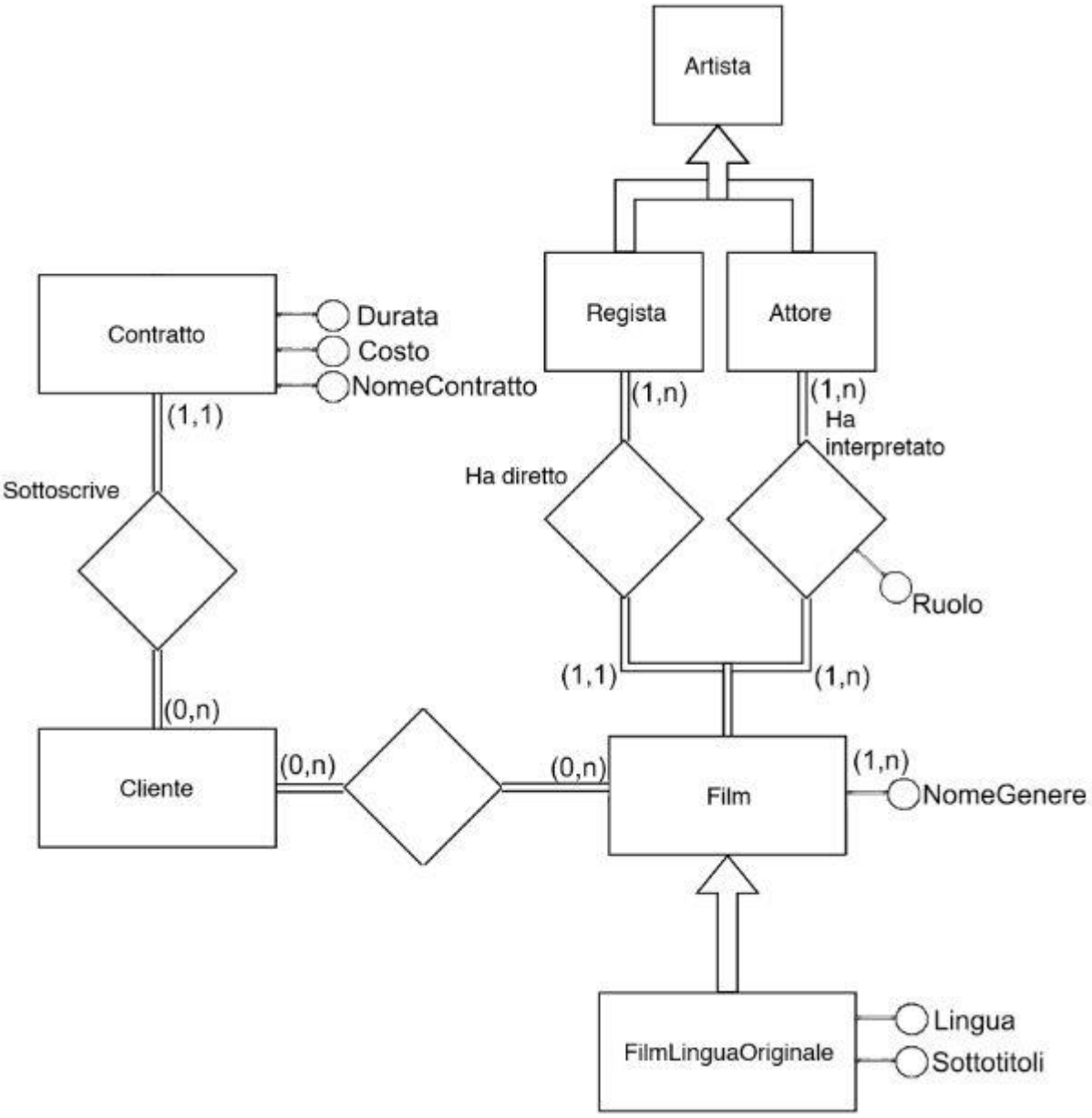
La relazione “Ha partecipato” fa uso di due attributi che permettono di capire se l’artista che prende parte al film è un attore (e in caso dirne il ruolo) o un regista. Evidentemente la relazione può essere divisa in due relazioni che svolgono un ruolo distinto nella nostra base di dati. Applichiamo quindi la R3 e dividiamo “Ha partecipato” in due relazioni chiamate “Ha diretto” e “Ha interpretato”.



“Ha interpretato” mantiene l’attributo “Ruolo” così da mantenere la specifica di indicare il ruolo interpretato da un attore in un dato film, ma siamo riuscito a distinguere universalmente gli attori dai registi.

VERSO LO SCHEMA FINALE #2

Un'altra modifica che ci teniamo a fare allo schema è legata alle due entità "Film" e "Artista". In particolare nelle specifiche viene richiesto di rappresentare anche Film in lingua originale (ovvero particolari istanze dell'entità Film che non hanno audio in italiano) e di differenziare gli artisti tra registi e attori. Ci rendiamo quindi conto che esistono due gerarchie da inserire nello schema e lo facciamo usando la R2.



La gerarchia di "Film" è **esclusiva** e **non totale** (non tutti i film sono in lingua originale).

La gerarchia di "Artista" è **totale** e **non esclusiva (**alcuni attori sono anche registi).

VERSO LO SCHEMA FINALE #3

Lo schema ricavato nella precedente immagine non è comunque ancora accettabile perché presenta dei problemi legati alla **consistenza e corretta rappresentazione** dei dati.

Ho aggiunto solo una parte degli attributi in alcune entità. Per esempio l'entità "Film" ha un attributo (**NomeGenere**) che ha cardinalità multipla. Questo potrebbe darci problemi nel rappresentarlo in fase di progettazione fisica e in particolare crea un problema di consistenza dei dati (che fare se indichiamo un genere usando nomi diversi o nomi composti?).

Anche l'entità "Contratto" così non può andare bene. Gli attributi (**Durata, Costo, NomeContratto**) potrebbero variare in funzione di particolari inserimenti di n-uple che, seppur indicando lo stesso tipo di contratto, usano *combinazioni di valori diversi e imprecisi*. Ipotizzando che i dati fossero sempre consistenti tra loro, **non potremmo impedire** la creazione di nuovi tipi di contratto e se dovessimo cancellare tutti i contratti di un certo tipo, **ne perderemmo la traccia e il formato per sempre**.

Evidentemente dobbiamo creare delle entità apposite per gestire i generi di film e i tipi di contratto che secondo le specifiche devono essere esattamente 3: mensile, trimestrale e annuale.

Usiamo quindi la regola R1 su entrambe le entità e creiamo le due nuove entità "TipoContratto" e "Genere". Inoltre usiamo le regole R5 e R6 per aggiungere tutti gli attributi che non avevamo ancora preso in considerazione, ma che ci vengono richiesti dalle specifiche.

Lo schema finale a cui arriviamo è rappresentato nella pagina seguente.

SCHEMA FINALE

Questo è quindi lo schema finale. Ho evidenziato a parte le entità “Cliente” e “Film” per mantenere una buona chiarezza a livello di schema generale, ma non perdere le informazioni legate agli attributi delle due entità.

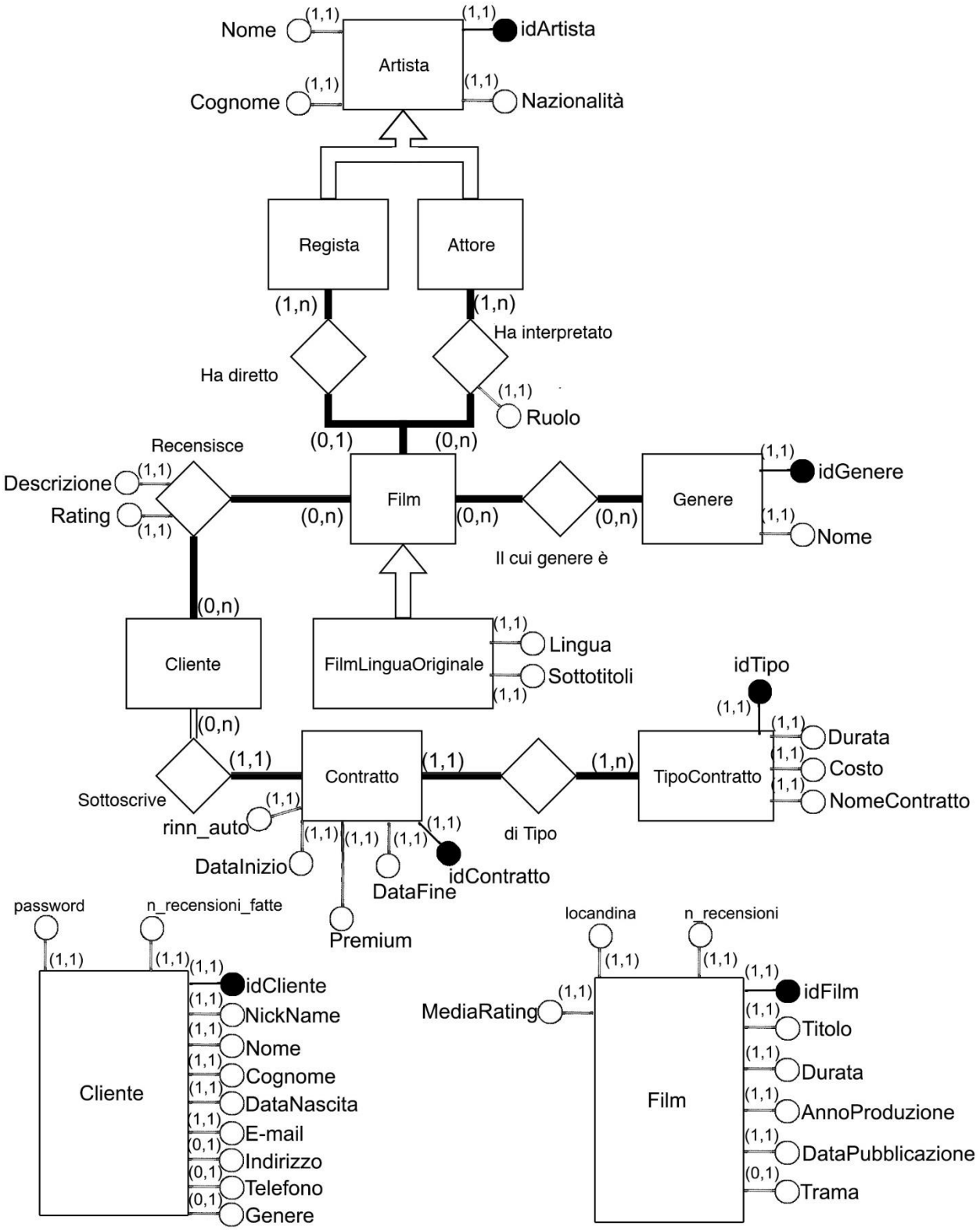


TABELLE DELLE ENTITA' E DELLE RELAZIONI

DIZIONARIO ENTITA'

Entità	Descrizione	Attributi	Identificatore
Artista	Attore/Regista che prende parte ad un film	Nome, Cognome, Nazionalità	idArtista
Regista	Artista che dirige film	Vedi Artista	Vedi Artista
Attore	Artista che recita nei film	Vedi Artista	Vedi Artista
Film	Film presente sul catalogo	Titolo, Durata, AnnoProduzione, DataPubblicazione, Trama, n_recensioni, mediarating, locandina	idFilm
FilmLinguaOriginale	Film che non ha l'audio in lingua italiana, ma potrebbe avere i sottotitoli in italiano	Vedi Film, Lingua, Sottotitoli	Vedi Film
Genere	Caratterizza i generi dei film	Nome	idGenere
Cliente	Utente iscritto al sito	NickName, Nome, Cognome, DataNascita, E-mail, Indirizzo, Telefono, Genere, n_recensioni_fatte, password	idCliente
Contratto	Stipulabile per usufruire del servizio di streaming	DataInizio, DataFine, Premium, rinn_auto	idContratto
TipoContratto	Identifica quale contratto si sottoscrive	Nome, Durata, Costo	IdTipo

DIZIONARIO RELAZIONI

Relazione	Entità partecipanti	Descrizione	Attributi
Ha diretto	Regista, Film	Associa un regista ad uno o più film. Ogni film ha un regista	
Ha interpretato	Attore, Film	Associa uno o più attori ad uno o più film	Ruolo
Il cui genere è	Film, Genere	Associa uno o più generi ad uno o più film	
Recensisce	Film, Cliente	Associa ai film una recensione compilata da uno o più clienti	Descrizione, Rating
Sottoscrive	Cliente, Contratto	Associa ad ogni cliente che ne fa richiesta un contratto	
Di Tipo	Contratto, TipoContratto	Associa ad ogni contratto il tipo di contratto che lo caratterizza	

VINCOLI DI INTEGRITA' E VALORI DERIVABILI

I **vincoli** che vanno rispettati sono i seguenti:

- Possono esistere utenti che non hanno mai recensito e film mai recensiti, ma ogni recensione deve avere un utente e un film
- Devono esserci esattamente 3 tipi di contratti (andrà impedito di aggiungere nuove tipologie da utenti normali)
- Per ogni attore che viene associato ad un film dove recita si deve indicare il ruolo che ha impersonato
- Sottotitoli dell'entità "FilmLinguaOriginale" è un valore booleano
- Premium dell'entità "Film" è un valore booleano
- Rinn_auto dell'entità "Contratto" è un valore booleano. Se è impostato a True, andrà creato un nuovo contratto con le stesse modalità non appena si conclude il contratto aperto dal cliente interessato
- Un contratto è considerato "stipulato" e viene associato ad un cliente se e solo se il cliente ha pagato la spesa pattuita (si paga una tantum l'intero importo a prescindere del tipo di contratto nel momento della stipulazione)
- Il premium può essere ottenuto solo durante la stipulazione del contratto (quindi solo in fase di inserimento e non è possibile aggiornarlo)
- Rating deve essere un valore numerico con virgola compreso tra 1 e 5

I **valori derivabili** nel nostro schema sono tre e sono così calcolati:

- N_recensioni_fatte dell'entità "Film" si calcola tenendo conto del numero di clienti che partecipano alla relazione "Ha recensito" con la particolare istanza di Film.
- Media_Rating dell'entità "Film" indica il numero di recensioni fatte per quell'istanza di film ed è calcolato tenendo conto del numero di recensioni presenti nella relazione "Ha recensito" e i valori di rating presenti nella suddetta relazione.
- N_recensioni_fatte dell'entità "Clienti" si calcola tenendo conto del numero di film che partecipano alla relazione "Recensisce" con la particolare istanza di Clienti.

PROGETTAZIONE LOGICA

Iniziamo la progettazione logica studiando il carico applicativo dei record che popoleranno la base di dati.

Indichiamo quindi la seguente tabella dei volumi:

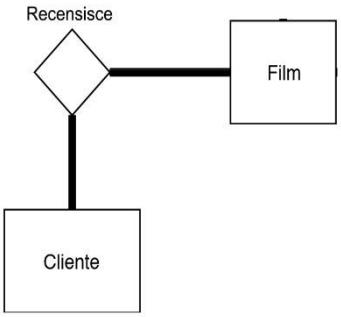
CONCETTO	TIPO	VOLUME
ARTISTA	E	12000
FILM	E	4000
GENERE	E	200
CLIENTE	E	1000000
CONTRATTO	E	4000000
HA DIRETTO	R	4000
HA INTERPRETATO	R	60000
IL CUI GENERE E'	R	12000
HA RECENSITO	R	15000

Non ho indicato alcune entità e relazioni perché hanno un volume scontato (per esempio l'entità "TipoContratto" ha esattamente 3 record come indicato dal vincolo).

Le operazioni che si devono rendere possibili sono le seguenti:

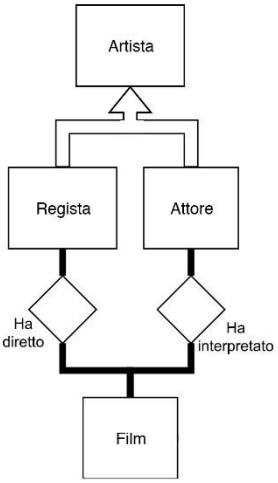
OPERAZIONE	DESCRIZIONE	TIPO	FREQUENZA
O1	Sottoscrizione di un nuovo contratto	I	2000 / giorno
O2	Chiusura di un contratto e rinnovo	I	2000 / giorno
O3	Ricerca di un film	I	30000 / giorno
O4	Aggiunta di un film	I	200 / anno
O5	Aggiunta di un regista	I	50 / anno
O6	Aggiunta di un attore	I	400 / anno
O7	Visionare la spesa totale di un cliente	I	2500 / giorno
O8	Calcolo del bilancio mensile	B	1 / mese
O9	Invio della newsletter agli utenti iscritti	B	1 / settimana
O10	Iscrizione nuovo utente	I	2000 / giorno
O11	Aggiungere una nuova recensione	I	900 / giorno
O12	Visionare le recensioni totali di un film	I	20000 / giorno
O13	Visionare le recensioni totali di un utente	I	2500 / giorno
O14	Visionare la scheda di un film	I	25000 / giorno

CONTROLLO DELLE RIDONDANZE: n_recensioni



Controlliamo per primo l'attributo n_recensioni dell'entità "Film". Le operazioni legate a questo parametro sono la O11, O12 e la O14. La parte di schema interessata per le operazioni O11 e O12 è indicata dalla foto alla sinistra di questo paragrafo. Mentre quella per O14 è indicata a destra.

Tenendo conto che n_recensioni è un intero di 4B in base alla tabella di volumi dovremo tenere 4KB di dati per mantenere la ridondanza. Ne vale la pena?



Il costo delle operazioni senza ridondanza è indicato dalla seguente tabella (S = 2L):

O11		O12		O14	
Cliente	L	**Film**	L	**Film**	L
Film	L	**Recensisce**	L	**Recensisce**	2L
Recensisce	S			**Ha diretto**	L
Film	S			**Ha interpretato**	L
				Attore	L
				Regista	L

Mentre il costo con la ridondanza è indicato dalla seguente tabella:

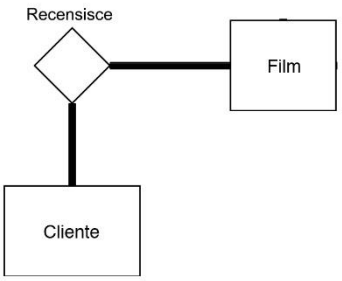
O11		O12		O14	
Cliente	L	**Film**	L	**Film**	L
Film	L			**Recensisce**	L
Recensisce	S			**Ha diretto**	L
Film	S			**Ha interpretato**	L
				Attore	L
				Regista	L

Senza ridondanza avremo un costo di 6*900 + 2*20000 + 7*25000 = 220400 operazioni al giorno.

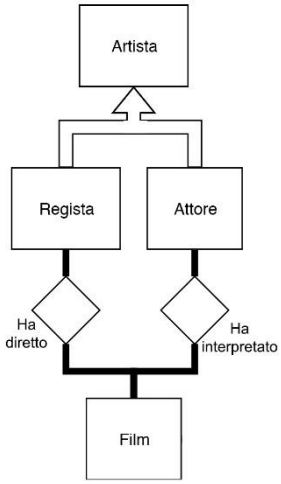
Con ridondanza avremo un costo di 6*900 + 1*20000 + 6*25000 = 175400 operazioni al giorno.

Manterremo quindi la ridondanza in quanto ci fa risparmiare un numero di operazioni consistente e ci permetterà di semplificare alcune query legate alle operazioni indicate.

CONTROLLO DELLE RIDONDANZE: MediaRating



Faremo lo stesso lavoro anche con MediaRating. Le operazioni interessate sono O3, O11, O14. O3 e O11 utilizzano lo schema riportato a sinistra in alto, mentre O14 quello a sinistra in basso. Vediamo adesso i costi per ogni operazione senza e con ridondanza. MediaRating sarà un valore double (8B) che richiederà 8KB di spazio aggiuntivo.



Senza:

O3		O11		O14	
Film	L	**Cliente**	L	**Film**	L
Recensisce	L	**Film**	L	**Recensisce**	2L
		Recensisce	S	**Ha diretto**	L
		Film	S	**Ha interpretato**	L
				Attore	L
				Regista	L

Con:

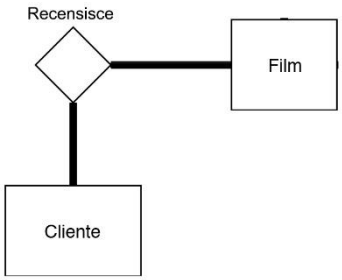
O3		O11		O14	
Film	L	**Cliente**	L	**Film**	L
		Film	L	**Recensisce**	2L
		Recensisce	S	**Ha diretto**	L
		Film	S	**Ha interpretato**	L
				Attore	L
				Regista	L

In definitiva i costi senza e con delle 3 operazioni sono:

- Senza: 2*30000 + 6*5400 + 7*25000 = 240400 operazioni al giorno.
- Con: 1*30000 + 6*5400 + 6*25000 = 185400 operazioni al giorno.

Anche in questo caso manterremo la ridondanza a fronte di un minor costo di operazioni e per semplificare alcune di queste query.

CONTROLLO DELLE RIDONDANZE: n_recensioni_fatte



Infine controlliamo l'attributo n_recensioni_fatte. Quest'attributo verrebbe usato dalle operazioni O11 e O13. Entrambe coinvolgono una parte dello schema proposto nell'immagine a sinistra.

Mantenere questa ridondanza comporterebbe un integer aggiuntivo dentro Cliente con un costo totale di 4MB.

I costi senza e con ridondanza delle operazioni sono calcolati di seguito come per gli altri due casi.

Senza:

O11		**O13**	
Cliente	L	**Cliente**	L
Film	L	**Recensito**	L
Recensisce	S		
Film	S		

Con:

O11		**O13**	
Cliente	L	**Cliente**	L
Film	L		
Recensisce	S		
Film	S		

I costi delle operazioni sono:

- Senza: 6*900 + 2*2500 = 10800 operazioni al giorno.
- Con: 6*900 + 1*2500 = 7500 operazioni al giorno.

Le operazioni sono ridotte di una piccola quantità e non giustificano quindi la presenza di un dato di ridondanza che peraltro, nel suo complesso, utilizza 4MB. Decidiamo quindi di eliminare questo attributo ridondante dallo schema.

VERSO IL MODELLO LOGICO

Prima di passare alla normalizzazione e alla finale rappresentazione fisica del nostro schema E/R bisogna decidere come tradurre le due gerarchie e come raffigurare le relazioni (in particolare le due relazioni molti a molti).

Diciamo subito che per entrambe le gerarchie semplicemente faremo un "collasso verso l'alto".

FilmLinguaOriginale è un sottoinsieme dei Film del catalogo. Utilizzeremo l'attributo "Lingua" come attributo selettore: se infatti questo attributo sarà NULL, allora il Film sarà inteso come in lingua italiana.

Artisti ha le due sottoclassi **Regista** e **Attore**. Non essendo una gerarchia con esclusività e considerato che entrambe le entità figlie non aggiungono nuovi attributi, preferiamo accorpare le due entità. Sarà ancora possibile comprendere se un artista è un attore o un regista per un dato film percorrendo le due relazioni ed inoltre dentro Film sarà presente un id indicante il regista di riferimento, ma essendo comune che diversi attori siano anche stati registi (e viceversa), non vogliamo appesantire lo schema dividendo le due entità.

La relazione "Recensisce" verrà tradotta in una entità chiamata **Recensione**. Questa avrà come chiave primaria le chiavi esterne di un film e un cliente e terrà traccia delle recensioni dei clienti mantenendo i due attributi della precedente relazione.

La relazione "Ha interpretato" verrà tradotta in una entità chiamata **AttoreFilm** e utilizzerà come chiave primaria la coppia di chiavi esterne <idArtista,idFilm>.

La relazione "Il cui genere è" verrà tradotta in una entità chiamata **GenereFilm** e utilizzerà come chiave primaria la coppia di chiavi esterne <idGenere,idFilm>.

Le altre relazioni (ovvero tutte le relazioni 1 a molti) verranno semplicemente eliminate aggiungendo una chiave esterna (di una delle due entità partecipanti nella relazione) dentro l'altra entità.

Cosi discorrendo avremo che:

- Film conterrà la chiave esterna di Artista (in riferimento ad un regista). Eliminiamo quindi la relazione "Ha diretto"
- Contratto conterrà le chiavi esterne di Cliente e TipoContratto, eliminando le relazioni "Sottoscrive" e "di Tipo"

Possiamo quindi tradurre lo schema nel modello logico e attuare la normalizzazione.

MODELLO LOGICO

Artista(idartista, nome, cognome, nazionalità**)**

AttoreFilm(idattore, idfilm, ruolo**)**

Film(idfilm, titolo, durata, annoprod, datapub, trama, idregista, n_recensioni, mediarating, lingua, sottotitoli, locandina**)**

GenereFIlm(idgenere, idfilm**)**

Genere(idgenere, nome**)**

Recensione(idcliente, idfilm, descrizione, rating**)**

Cliente(idcliente, nickname, password, nome, cognome, datanascita, email, indirizzo, telefono, genere, password**)**

Contratto(idcontratto, idcliente, idtipo, datainizio, datafine, premium, rinn_auto**)**

TipoContratto(idtipo, durata, costo, nomecontratto**)**

NORMALIZZAZIONE

Le dipendenze funzionali del nostro schema sono le seguenti:

Artista: idartista-> nome, cognome, nazionalità

AttoreFilm: idattore, idfilm -> ruolo

Film: idfilm -> titolo, durata, annoprod, datapub, trama, idregista, n_recensioni, mediarating, lingua, sottotitoli, locandina

Genere: idgenere -> nome

Recensione: idcliente, idfilm -> descrizione, rating

Cliente: idcliente -> nickname, password, nome, cognome, datanascita, email, indirizzo, telefono, genere, password

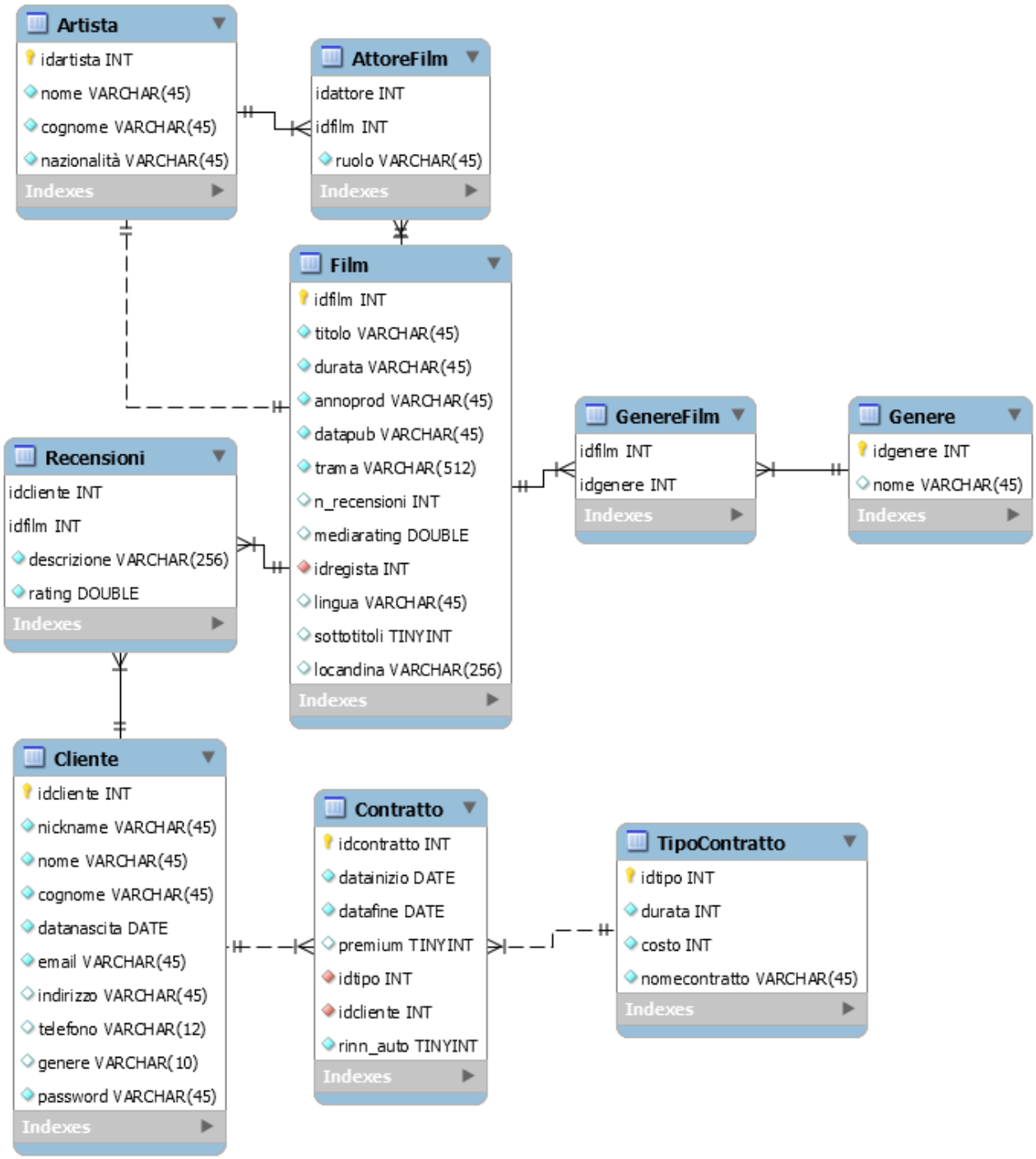
Contratto: idcontratto -> idcliente, idtipo, datainizio, datafine, premium, rinn_auto

TipoContratto: idtipo -> durata, costo, nomecontratto

Possiamo tranquillamente notare che tutte le dipendenze sono del formato **chiave** -> **attributo** e quindi la decomposizione del nostro schema soddisfa la condizione per essere in **BCNF**.

MODELLO FISICO

Possiamo adesso iniziare a creare il modello fisico. Lo schema dovrà avere la seguente struttura (stilizzata usando MySql Workbench):



IMPLEMENTAZIONE SCHEMA E TABELLE

Utilizzando come DBMS MySql, daremo adesso il codice per creare l'intero schema della base di dati.

```
-- -----------------------------------------------------
-- Schema streaming_film
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `streaming_film` DEFAULT CHARACTER SET utf8 ;
USE `streaming_film` ;

-- -----------------------------------------------------
-- Table `streaming_film`.`Artista`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`Artista` (
  `idartista` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `nazionalità` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idartista`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `streaming_film`.`GenereFilm`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`GenereFilm` (
  `idfilm` INT NOT NULL,
  `idgenere` INT NOT NULL,
  INDEX `fk_GenereFilm_Film1_idx` (`idfilm` ASC) VISIBLE,
  INDEX `fk_GenereFilm_Genere1_idx` (`idgenere` ASC) VISIBLE,
  PRIMARY KEY (`idfilm`, `idgenere`),
  CONSTRAINT `fk_GenereFilm_Film1`
    FOREIGN KEY (`idfilm`)
    REFERENCES `streaming_film`.`Film` (`idfilm`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_GenereFilm_Genere1`
    FOREIGN KEY (`idgenere`)
    REFERENCES `streaming_film`.`Genere` (`idgenere`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `streaming_film`.`Film`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`Film` (
  `idfilm` INT NOT NULL AUTO_INCREMENT,
  `titolo` VARCHAR(45) NOT NULL,
  `durata` VARCHAR(45) NOT NULL,
  `annoprod` VARCHAR(45) NOT NULL,
  `datapub` DATE NULL,
  `trama` VARCHAR(512) NULL,
  `n_recensioni` INT NULL,
  `mediarating` DOUBLE NULL,
  `idregista` INT NOT NULL,
  `lingua` VARCHAR(45) NULL,
  `sottotitoli` TINYINT NULL,
  `locandina` VARCHAR(256) NULL,
  PRIMARY KEY (`idfilm`),
  INDEX `fk_Film_Artista1_idx` (`idregista` ASC) VISIBLE,
  CONSTRAINT `fk_Film_Artista1`
    FOREIGN KEY (`idregista`)
    REFERENCES `streaming_film`.`Artista` (`idartista`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `streaming_film`.`Genere`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`Genere` (
  `idgenere` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NULL,
  PRIMARY KEY (`idgenere`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `streaming_film`.`AttoreFilm`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`AttoreFilm` (
  `idattore` INT NOT NULL,
  `idfilm` INT NOT NULL,
  `ruolo` VARCHAR(45) NOT NULL,
  INDEX `fk_AttoreFilm_Artista_idx` (`idattore` ASC) VISIBLE,
  INDEX `fk_AttoreFilm_Film1_idx` (`idfilm` ASC) VISIBLE,
  PRIMARY KEY (`idattore`, `idfilm`),
  CONSTRAINT `fk_AttoreFilm_Artista`
    FOREIGN KEY (`idattore`)
    REFERENCES `streaming_film`.`Artista` (`idartista`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_AttoreFilm_Film1`
    FOREIGN KEY (`idfilm`)
    REFERENCES `streaming_film`.`Film` (`idfilm`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `streaming_film`.`Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`Cliente` (
  `idcliente` INT NOT NULL AUTO_INCREMENT,
  `nickname` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `datanascita` DATE NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `indirizzo` VARCHAR(45) NULL,
  `telefono` VARCHAR(12) NULL,
  `genere` VARCHAR(10) NULL,
  `password` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idcliente`),
  UNIQUE INDEX `nickname_UNIQUE` (`nickname` ASC) VISIBLE,
  UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `streaming_film`.`Recensioni`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`Recensioni` (
  `idcliente` INT NOT NULL,
  `idfilm` INT NOT NULL,
  `descrizione` VARCHAR(256) NOT NULL,
  `rating` DOUBLE NOT NULL,
  INDEX `fk_Recensioni_Cliente1_idx` (`idcliente` ASC) VISIBLE,
  INDEX `fk_Recensioni_Film1_idx` (`idfilm` ASC) VISIBLE,
  PRIMARY KEY (`idcliente`, `idfilm`),
  CONSTRAINT `fk_Recensioni_Cliente1`
    FOREIGN KEY (`idcliente`)
    REFERENCES `streaming_film`.`Cliente` (`idcliente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Recensioni_Film1`
    FOREIGN KEY (`idfilm`)
    REFERENCES `streaming_film`.`Film` (`idfilm`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `streaming_film`.`TipoContratto`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`TipoContratto` (
  `idtipo` INT NOT NULL,
  `durata` INT NOT NULL,
  `costo` INT NOT NULL,
  `nomecontratto` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idtipo`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `streaming_film`.`Contratto`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `streaming_film`.`Contratto` (
  `idcontratto` INT NOT NULL,
  `datainizio` DATE NOT NULL,
  `datafine` DATE NOT NULL,
  `premium` TINYINT NULL DEFAULT 0,
  `idtipo` INT NOT NULL,
  `idcliente` INT NOT NULL,
  `rinn_auto` TINYINT NOT NULL,
  PRIMARY KEY (`idcontratto`),
  INDEX `fk_Contratto_TipoContratto1_idx` (`idtipo` ASC) VISIBLE,
  INDEX `fk_Contratto_Cliente1_idx` (`idcliente` ASC) VISIBLE,
  CONSTRAINT `fk_Contratto_TipoContratto1`
    FOREIGN KEY (`idtipo`)
    REFERENCES `streaming_film`.`TipoContratto` (`idtipo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Contratto_Cliente1`
    FOREIGN KEY (`idcliente`)
    REFERENCES `streaming_film`.`Cliente` (`idcliente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

IMPLEMENTAZIONE TRIGGER

Per poter gestire le variabili derivabili e alcuni vincoli di integrità introduciamo dei trigger. In particolare introdurremo i seguenti trigger:

```
/*
	Trigger che aumenta di uno il numero di recensioni per un film quando ne viene inserita una
*/

CREATE TRIGGER aumenta_n_recensione
after insert on streaming_film.recensioni
for each row
UPDATE film
set n_recensioni = n_recensioni + 1
where film.idfilm = new.idfilm

/*
	Trigger che riduce di uno il numero di recensioni per un film quando ne viene eliminata una
*/

CREATE TRIGGER riduci_n_recensione
after delete on streaming_film.recensioni
for each row
UPDATE film
set n_recensioni = n_recensioni - 1
where film.idfilm = old.idfilm
```

```
/*
	Calcola la media dei rating ogni volta che viene aggiunto un nuovo rating
*/

DELIMITER //

CREATE TRIGGER calcola_mediarating_aumento
after insert on streaming_film.recensioni
for each row
BEGIN
	DECLARE somma DOUBLE;
	DECLARE n INTEGER;

	select count(*), sum(rating) into n,somma
	from streaming_film.recensioni
	where recensioni.idfilm = new.idfilm
	group by idfilm;

	UPDATE streaming_film.film
	set film.mediarating = somma/n
	where film.idfilm = new.idfilm;
END//

DELIMITER ;
```

```
/*
	Calcola la media dei rating ogni volta che viene modificato un rating esistente
*/

DELIMITER //

CREATE TRIGGER calcola_mediarating_modifica
after update on streaming_film.recensioni
for each row
BEGIN
	DECLARE somma DOUBLE;
	DECLARE n INTEGER;

	select count(*), sum(rating) into n,somma
	from streaming_film.recensioni
	where recensioni.idfilm = new.idfilm
	group by idfilm;

	UPDATE streaming_film.film
	set film.mediarating = somma/n
	where film.idfilm = new.idfilm;
END//

DELIMITER ;


/*
	Evita che si modifichi il valore di premium una volta stipulato un contratto
*/

CREATE TRIGGER restore_premium
before update on streaming_film.contratto
for each row
	set new.premium = old.premium
```

```
/*
	Calcola la media dei rating ogni volta che viene eliminato un rating esistente
*/

DELIMITER //

CREATE TRIGGER calcola_mediarating_riduci
after delete on streaming_film.recensioni
for each row
BEGIN
	DECLARE somma DOUBLE;
	DECLARE n INTEGER;
	DECLARE media DOUBLE;

	select count(*), sum(rating) into n,somma
	from streaming_film.recensioni
	where recensioni.idfilm = old.idfilm
	group by idfilm;

	if n > 0 then
		SET media = somma/n;
	else
		SET media = 0;
	end if;

	UPDATE streaming_film.film
	set film.mediarating = media
	where film.idfilm = old.idfilm;
END//

DELIMITER ;
```

```
/*
	Indica la data esatta della fine di un contratto non appena creato correggendo eventuali errori o dimenticanze.
*/

DELIMITER //

CREATE TRIGGER check_date
before insert on streaming_film.contratto
for each row
begin
	DECLARE n integer;
	select (new.datainizio + INTERVAL durata*30 DAY) into n
	from tipocontratto
	where idtipo = new.idtipo;

   set new.datafine = n;
end//

DELIMITER ;

/*
	Evita di eliminare un utente che è disattivato
*/

DELIMITER //

CREATE TRIGGER no_eliminazione_cliente_disattivato
before delete on streaming_film.cliente
for each row
begin
	if old.disattivato = 1 THEN
	SIGNAL sqlstate '45001' set message_text = "Il cliente selezionato è attualmente disattivo. Non è possibile cancellarlo senza la sua autorizzazione!";
   end if;
end//

DELIMITER ;
```

IMPLEMENTAZIONE PROCEDURA RINNOVO AUTOMATICO DEI CONTRATTI

Proponiamo anche una procedura che permette di essere schedulata (usando la funzionalità degli eventi di MySql) per la mezzanotte di ogni giorno. Questa procedura dovrà semplicemente controllare se vi sono contratti che terminano esattamente quel giorno e rinnovarli automaticamente se è previsto rinnovo automatico.

```
DELIMITER //
CREATE PROCEDURE crea_nuovi_contratti ()
BEGIN
   DECLARE finished INTEGER DEFAULT 0;
   DECLARE idc integer DEFAULT 0;
   DECLARE idt integer DEFAULT 0;
   -- cursore per clienti premium e non
   DECLARE curClientiP
      CURSOR FOR
         SELECT idcliente, idtipo FROM contratto where premium = 1 && datafine = date(now());

          DECLARE curClienti
      CURSOR FOR
         SELECT idcliente, idtipo FROM contratto where premium = 0 && datafine = date(now());

   -- dichiara un handler per NOT FOUND
   DECLARE CONTINUE HANDLER
      FOR NOT FOUND SET finished = 1;

   OPEN curClientiP;

   clientiP: LOOP
      FETCH curClientiP INTO idc, idt;
      IF finished = 1 THEN
         LEAVE clientiP;
      END IF;
      -- crea un nuovo record
      insert into contratto(datainizio,datafine,premium,idcliente,idtipo)
                  values (date(now()),date(now()),1,idc,idt);
   END LOOP clientiP;

          CLOSE curClientiP;
          OPEN curClienti;

          clientiP: LOOP
      FETCH curClienti INTO idc, idt;
      IF finished = 1 THEN
         LEAVE clientiP;
      END IF;
      -- crea un nuovo record
      insert into contratto(datainizio,datafine,premium,idcliente,idtipo)
                  values (date(now()),date(now()),0,idc,idt);
   END LOOP clientiP;
   CLOSE curClienti;
END//
DELIMITER ;
```

IMPLEMENTAZIONE OPERAZIONI

Diamo di seguito il codice in SQL delle operazioni indicate nella pagina #12 di questo elaborato.

O1. Sottoscrizione di un nuovo contratto

INSERT INTO Contratto(idcliente, idtipo, datainizio,datafine,premium,rinn_auto)

VALUES (1435,2,'2019-09-15','2019-12-15',1,0);

O2. Chiusura di un contratto e rinnovo

Gestito dalla procedura.

O3. Ricerca di un film

SELECT titolo,durata,datapub,trama,n_recensioni,mediarating,nome as "Nome Regista", cognome as "Cognome Regista"

FROM Film JOIN Artista on idregista = idartista

WHERE Titolo LIKE "%Alien%"

O4. Aggiunta di un film

INSERT INTO Film(idfilm,titolo, durata, annoprod,datapub,idregista)

VALUES (0,"The Matrix",136,1998,'1999-03-31',7);

O5/O6. Aggiunta di un regista/attore

INSERT INTO Artista(nome, cognome, nazionalita)

VALUES ("Leonardo", "Di Caprio", "USA");

O7. Visionare la spesa totale di un cliente

```
CREATE VIEW mesi_premium(idcliente,tot_mesi) AS
SELECT idcliente, sum(durata)
FROM Contratto NATURAL JOIN TipoContratto
WHERE premium = 1
GROUP BY idcliente;

CREATE VIEW somma_costi_mensilità (idcliente,costo_totale_parziale) AS
SELECT idcliente, SUM(costo)
FROM Contratto NATURAL JOIN TipoContratto
GROUP BY idcliente;

SELECT s.idcliente, COALESCE(tot_mesi,0)*3+costo_totale_parziale as “SpesaTotale”
FROM somma_costi_mensilità s LEFT JOIN mesi_premium m
ON s.idcliente = m.idcliente;

DROP VIEW mesi_premium;

DROP VIEW somma_costi_mensilità;
```

O8. Calcolo del bilancio mensile

```
CREATE VIEW contratti_stipulati(idcontratto,durata,premium,costo) AS
SELECT idcontratto,durata,premium,costo
FROM Contratto NATURAL JOIN TipoContratto
WHERE datafine >= '2019-09-01' AND datafine <= '2019-09-30';

SELECT SUM(costo)+SUM(3*premium*durata) as BilancioMensile
FROM contratti_stipulati;

DROP VIEW contratti_stipulati;
```

O9. Calcolo del bilancio mensile

```
SELECT email
FROM Cliente c
WHERE EXISTS    (SELECT *
                 FROM Contratto NATURAL JOIN TipoContratto
                 WHERE c.idcliente = idcliente
                 AND datainizio + INTERVAL durata MONTH > DATE(NOW()) )
```

O11. Iscrizione nuovo utente

INSERT INTO
Cliente(idcliente,nickname,nome,cognome,datanascita,email,indirizzo,telefono,genere,password)
VALUES(0,“Chrono”,”Marco”,”Cavalli”,’1993-03-12’,”cavallimrc@gmail.com”,”Via Carmelina Naselli, 45”,”3931597016”,”maschio”,TOP-SECRET);

O11. Aggiungere una nuova recensione

INSERT INTO Recensioni(idcliente,idfilm,descrizione,rating)
VALUES (0,0,”Film bellissimo. Riprendetevelo se non lo avete ancora visto!”,5.0);

O12. Visionare le recensioni totali di un film

SELECT idfilm, n_recensioni
FROM Film
WHERE idfilm = 45;

O13. Visionare le recensioni totali di un utente

SELECT idcliente, count(*) as Totale_recensioni
FROM Cliente NATURAL JOIN Recensioni
WHERE idcliente = 0
GROUP BY idcliente;

O14. Visionare la scheda di un film

SELECT *
FROM FILM NATURAL JOIN GenereFilm
NATURAL JOIN Genere
NATURAL JOIN Recensioni
NATURAL JOIN AttoreFilm JOIN Artista a1
ON a1.idartista = idattore
JOIN Artista a2
ON a2.idartista = idregista;