Basi di Dati

Progetto A.A. 2023/2024

SISTEMA PER LA GESTIONE DI NOLEGGIO DI VIDEOCASSETTE E DVD

0306888

Francesco Masci

Indice

1. Descrizione del Minimondo

Si vuole realizzare un sistema informativo per la gestione di un negozio per il noleggio di videocassette e DVD, tenendo conto delle seguenti informazioni.

Il sistema mantiene le informazioni relative a tutte le persone impiegate presso il negozio. Per ciascun impiegato sono noti il codice fiscale, il nome, ed un recapito. Inoltre si vuole tenere traccia della carica che ha rivestito in un determinato periodo (per esempio, cassiere o commesso) considerando che tale qualifica può cambiare nel tempo. Le informazioni sul personale sono gestite dal proprietario del negozio, che è in grado di inserire nuovo personale nel sistema e di visualizzare report mensili ed annuali sulle ore lavorate dai dipendenti. I turni di lavoro sono inseriti, su base mensile, sempre dal proprietario.

I film disponibili presso il negozio sono identificati dal titolo e dal nome del regista; inoltre sono noti l'anno in cui il film è stato girato, l'elenco degli attori principali del film, il costo corrente di noleggio della videocassetta ed eventualmente i film disponibili presso la catena di cui il film in questione rappresenta la versione "remake".

Per ogni film è nota la collocazione all'interno del negozio. In particolare, sono noti il settore, la posizione all'interno del settore ed il numero di copie in cui il film è disponibile. Ciascun settore è identificato attraverso un codice numerico univoco all'interno del negozio.

I clienti, al momento della registrazione al negozio, ricevono una tessera cliente. Per ciascun cliente devono essere mantenute tutte le informazioni anagrafiche e viene associato anche un numero arbitrario di recapiti (telefono, email, cellulare) a cui possono essere contattati. Quando un cliente effettua un noleggio, viene registrata la data entro cui il film dovrà essere restituito. Il personale di cassa può gestire l'anagrafica dei clienti e gestire i noleggi. Inoltre, può visualizzare in ogni momento quali titoli sono associati ad un noleggio scaduto e quali sono i clienti che hanno effettuato tali noleggi.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
12	Catena	Negozio	Il Sistema gestisce le informazioni di un singolo negozio non di una catena di negozi.
21	Film	Copia di film	L'affermazione non si riferisce al concetto astratto di film ma alla specifica copia di film che viene noleggiata.
15	Disponibile	Noleggiabile	In questo caso il termine disponibile fa riferimento al numero di copie di un film presenti nel negozio e che possono essere effettivamente noleggiate.
12	Videocassetta	Videocassetta e DVD	Il prezzo è noto per entrambi i tipi di copia di un film.
23	Titoli	Copie di film	Il termine si riferisce alle copie di un film che sono state noleggiate.

Specifica disambiguata

Si vuole realizzare un sistema informativo per la gestione di un negozio per il noleggio di videocassette e DVD, tenendo conto delle seguenti informazioni.

Il sistema mantiene le informazioni relative a tutte le persone impiegate presso il negozio. Per ciascun impiegato sono noti il codice fiscale, il nome, ed un recapito (telefono). Inoltre si vuole tenere traccia della carica che ha rivestito in un determinato periodo (per esempio, cassiere o commesso) considerando che tale qualifica può cambiare nel tempo. Le informazioni sul personale sono gestite dal proprietario del negozio, che è in grado di inserire nuovo personale nel sistema e di visualizzare report mensili ed annuali sulle ore lavorate dai dipendenti. I turni di lavoro sono inseriti, su base mensile, sempre dal proprietario.

I film disponibili presso il negozio sono identificati dal titolo e dal nome del regista; inoltre sono noti l'anno in cui il film è stato girato, l'elenco degli attori principali del film, il costo corrente di noleggio della videocassetta e del DVD ed eventualmente i film disponibili presso il negozio di cui il film in questione rappresenta la versione "remake".

Per ogni copia del film è nota la collocazione all'interno del negozio. In particolare, sono noti il settore, la posizione all'interno del settore ed il numero di copie in cui il film è noleggiabile. Ciascun settore è identificato attraverso un codice numerico univoco all'interno del negozio.

I clienti, al momento della registrazione al negozio, ricevono una tessera cliente. Per ciascun cliente devono essere mantenute tutte le informazioni anagrafiche (nome, cognome, data di nascita, codice fiscale e indirizzo di residenza) e viene associato anche un numero arbitrario di recapiti (telefono,

email, cellulare) a cui possono essere contattati. Quando un cliente effettua un noleggio, viene registrata la data entro cui la copia del film dovrà essere restituita. Il personale di cassa può gestire l'anagrafica dei clienti e gestire i noleggi. Inoltre, può visualizzare in ogni momento quali copie di film sono associate ad un noleggio scaduto e quali sono i clienti che hanno effettuato tali noleggi.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Impiegato	Persona che lavora all'interno del negozio. Ha una carica che può cambiare nel tempo. Gestisce i clienti e i noleggi se ha la carica di cassiere. Il proprietario del negozio lo inserisce nel sistema e gli assegna mensilmente dei turni di lavoro.	Personale, dipendente	Copia di film, Cliente
Film	Concetto astratto di film disponibile nel catalogo del negozio. Può essere la versione remake di un altro film.		Copia di film
Copia di film	Copia fisica di un film presente nel catalogo del negozio. Può essere noleggiata da un cliente. Si trova in un settore all'interno del negozio.		Impiegato, Settore, Cliente
Settore	Contiene le copie fisiche dei film. È identificato da un codice unico.		Copia di film
Cliente	Persona che noleggia le copie dei film. Gli viene data una tessera cliente alla registrazione presso il negozio.		Impiegato, Copia di film

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Impiegato
Per ciascun impiegato sono noti il codice fiscale, il nome, ed un recapito. Inoltre si vuole tenere traccia della carica che ha rivestito in un determinato periodo (per esempio, cassiere o commesso) considerando che tale qualifica può cambiare nel tempo. Le informazioni sul personale sono gestite dal proprietario del negozio, che è in grado di inserire nuovo personale nel sistema e di visualizzare report mensili ed annuali sulle ore lavorate dai dipendenti. I turni di lavoro sono inseriti, su base mensile, sempre dal proprietario.

Frasi relative a Film
I film disponibili presso il negozio sono identificati dal titolo e dal nome del regista; inoltre sono noti l'anno in cui il film è stato girato, l'elenco degli attori principali del film, il costo corrente di noleggio della videocassetta e del DVD ed eventualmente i film disponibili presso il negozio di cui il film in questione rappresenta la versione "remake". […] sono noti […] il numero di copie in cui il film è noleggiabile.

Frasi relative a Copia di film
Per ogni copia del film è nota la collocazione all'interno del negozio. In particolare, sono noti il settore, la posizione all'interno del settore […]. Quando un cliente effettua un noleggio, viene registrata la data entro cui la copia del film dovrà essere restituita.

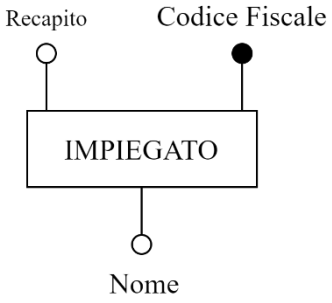
Frasi relative a Settore
Ciascun settore è identificato attraverso un codice numerico univoco all'interno del negozio.

Frasi relative a Cliente
I clienti, al momento della registrazione al negozio, ricevono una tessera cliente. Per ciascun cliente devono essere mantenute tutte le informazioni anagrafiche e viene associato anche un numero arbitrario di recapiti (telefono, email, cellulare) a cui possono essere contattati. Il personale di cassa può gestire l'anagrafica dei clienti […]. Inoltre, può visualizzare in ogni momento quali titoli sono associati ad un noleggio scaduto e quali sono i clienti che hanno effettuato tali noleggi.

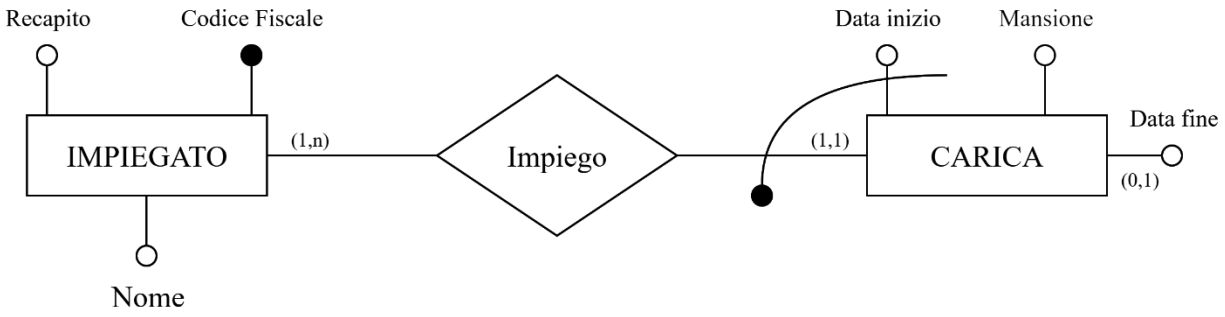
3. Progettazione concettuale

Costruzione dello schema E-R

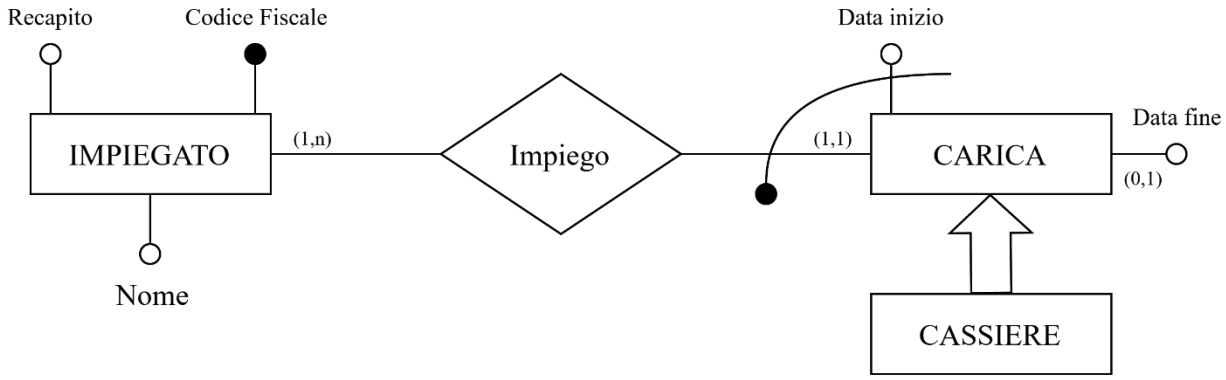
Si sviluppa lo schema ER a partire dal concetto di *Impiegato*. Questo viene realizzato attraverso un'entità a cui sono collegati gli attributi richiesti nelle specifiche.



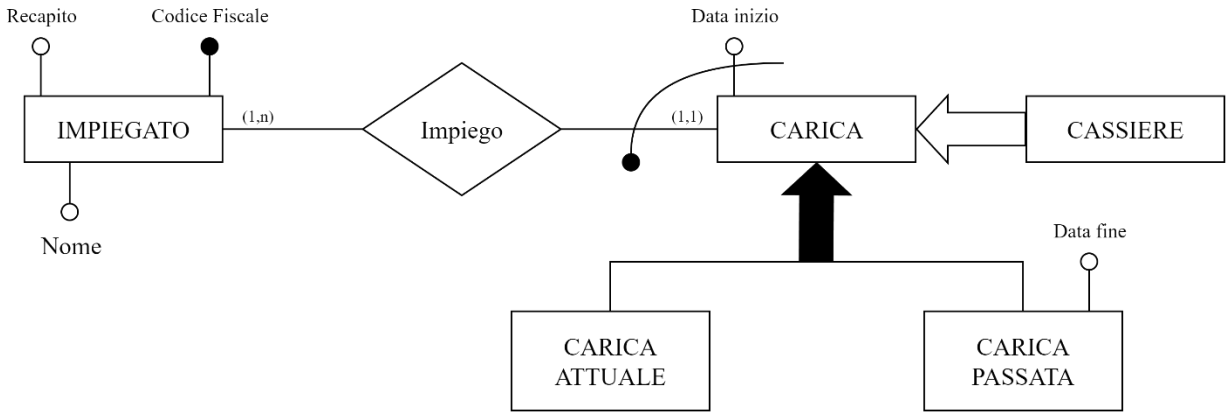
Si prosegue con la rappresentazione della carica di un impiegato. La prima realizzazione trovata è la seguente:



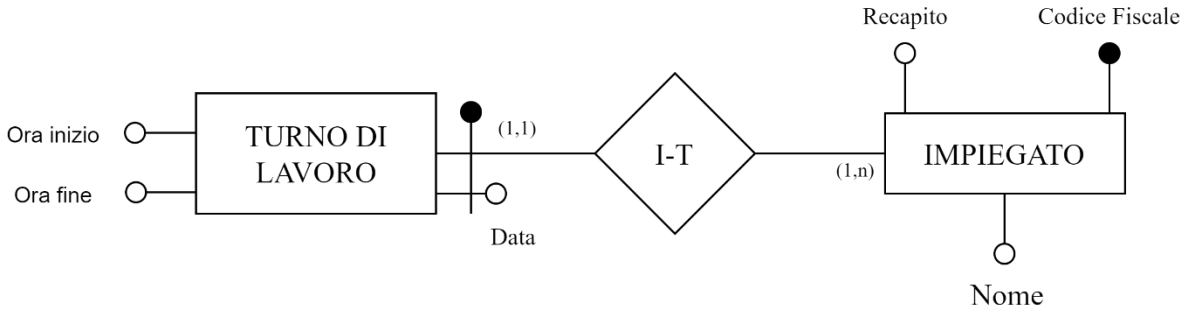
Dalle specifiche si evince che gli *Impiegati* con il ruolo di cassiere abbiano la possibilità di eseguire delle azioni di particolare interesse per il negozio (es. la registrazione dei noleggi e delle restituzioni). Per questo motivo si è deciso di utilizzare una generalizzazione parziale sull'entità *Carica* per enfatizzare proprio il diverso ruolo che hanno nel sistema tali impiegati. Non viene rappresentato tramite generalizzazione totale poiché gli altri ruoli del negozio non sono di particolare interesse.



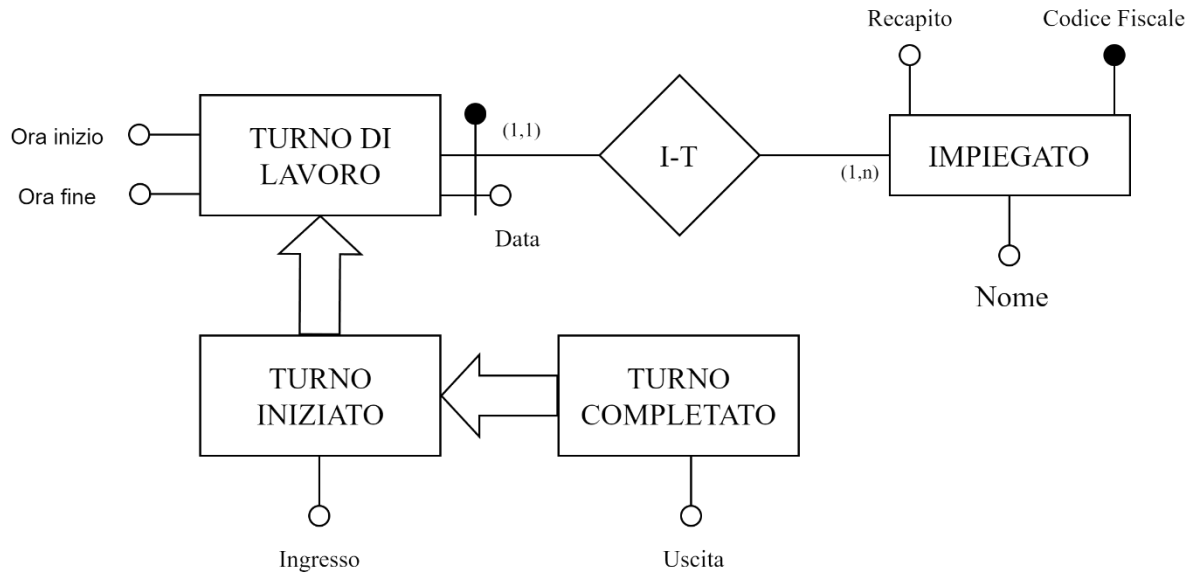
L'opzionalità sull'attributo *Data fine* però può causare ambiguità sul suo significato a questo livello della progettazione. Perciò si introduce una generalizzazione applicando il pattern di storicizzazione sull'entità *Carica* e si aggiunge una regola aziendale per impedire ad un *Impiegato* di avere più di una carica attuale all'interno del negozio.



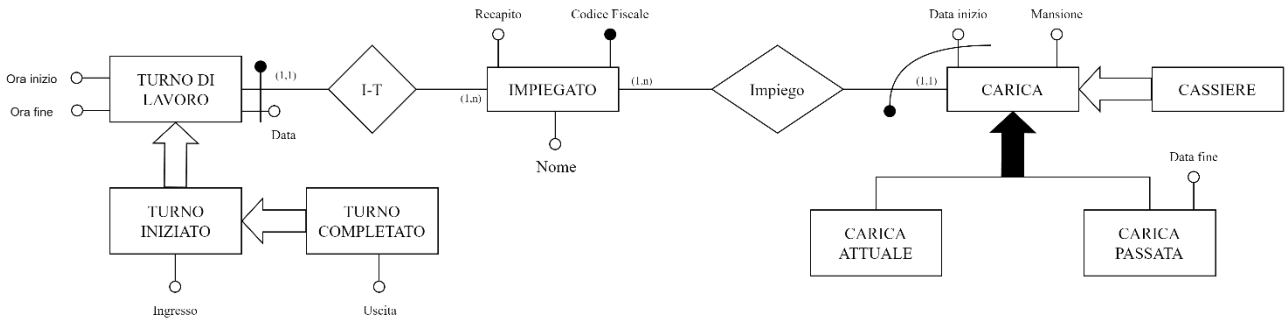
Per poter mantenere le informazioni circa i turni di lavoro dell'impiegato viene aggiunta un'entità identificata proprio dall'impiegato che deve svolgere il turno e la data a cui esso fa riferimento. Inoltre, vengono rappresentati anche l'orario di inizio e di fine del turno.



La rappresentazione fatta fino ad ora non permette di salvare le informazioni relative alle ore effettivamente svolte da un impiegato durante il turno di lavoro. Per farlo si introduce una generalizzazione parziale sull'entità *Turno di lavoro* per catturare i turni completati dagli impiegati con i relativi orari di entrata e uscita dal lavoro. Tale generalizzazione è fatta su due livelli, il primo livello cattura i turni iniziati man non ancora terminati, il secondo livello quelli completati. Si noti che in questa rappresentazione si è applicato il pattern di evoluzione di un'entità.

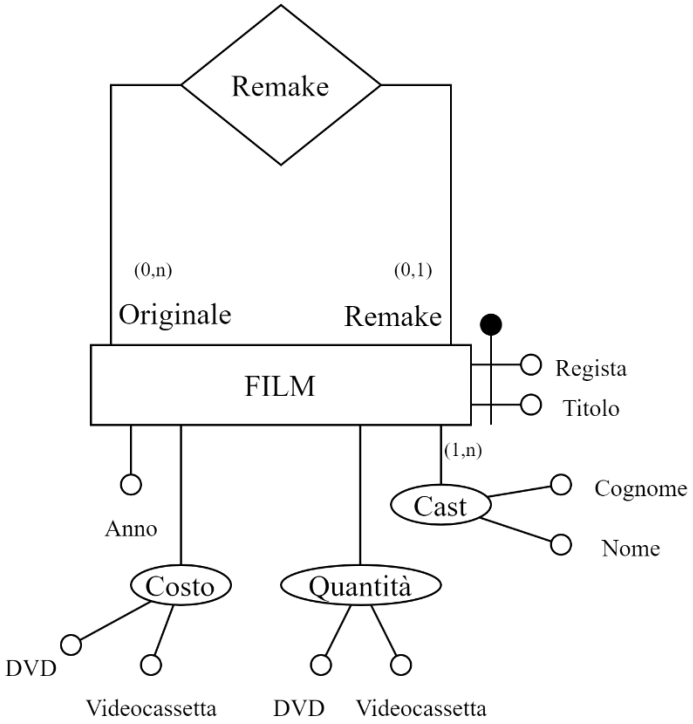


La porzione di schema relativa alle informazioni necessarie per un impiegato è ora completa:



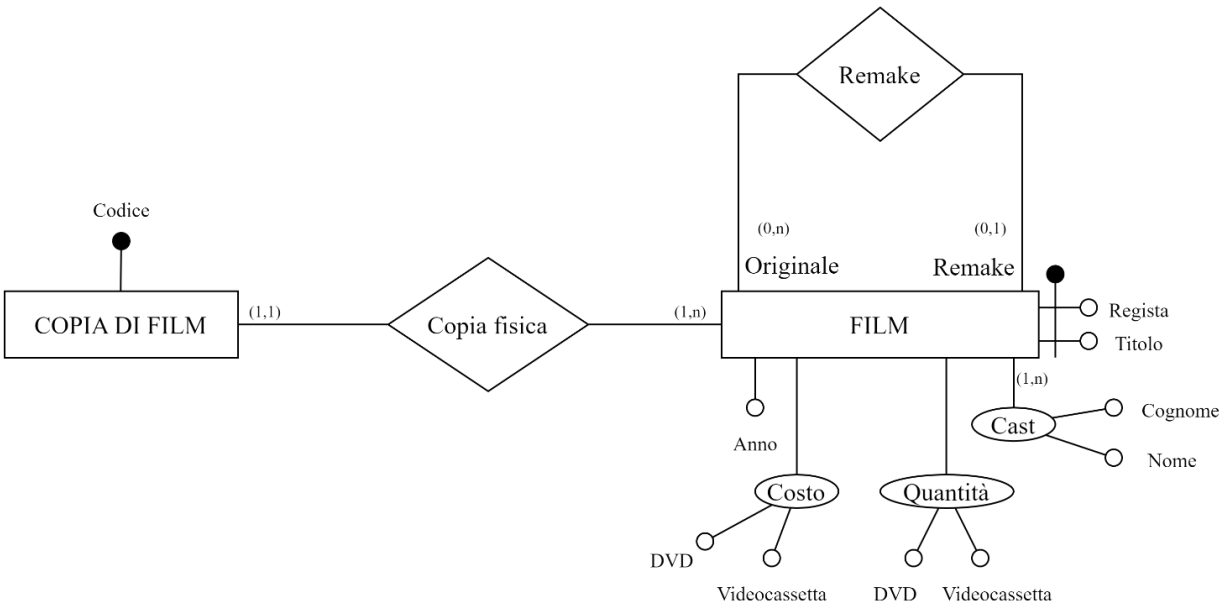
Si passa ora alla realizzazione della parte di schema che modella il catalogo dei film e mantiene le informazioni relative ai noleggi.

Come primo passo viene aggiunta un'entità film con i relativi attributi descritti dalla specifica. Inoltre, viene utilizzata una relazione ricorsiva sull'entità per rappresentare il concetto di *Remake*.

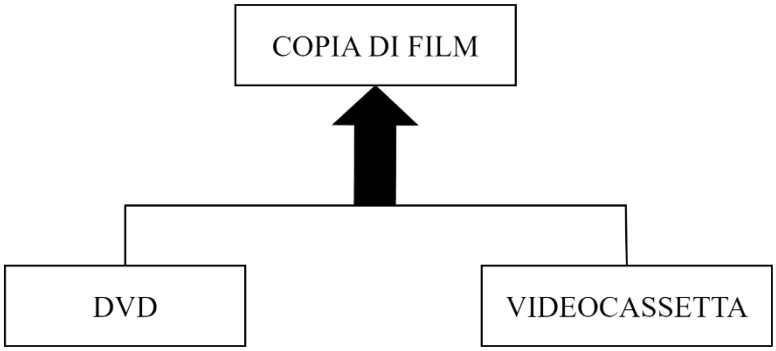


Si è scelto di rappresentare il cast del film come attributo multivalore poiché questo non è di grande importanza ai fini del sistema per cui non è stato rappresentato attraverso un'entità.

L'entità appena inserita va a rappresentare il concetto astratto di film. Per rappresentare una copia fisica specifica del film viene aggiunta una nuova entità in relazione con quella appena inserita utilizzando il pattern istanza-di. Per identificarla, come stabilito con il cliente durante un contatto, si utilizza un codice presente sulla copia fisica nel negozio che è univoco all'interno del negozio.

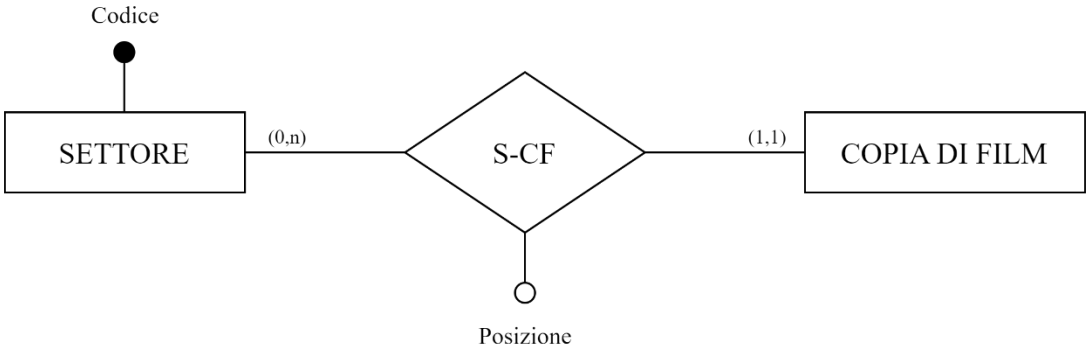


Il negozio però vuole gestire il prestito sia di *Videocassette* che di *DVD*, per cui si introduce una generalizzazione totale sull'entità *Copia di film* per discriminare il tipo di copia fisica.



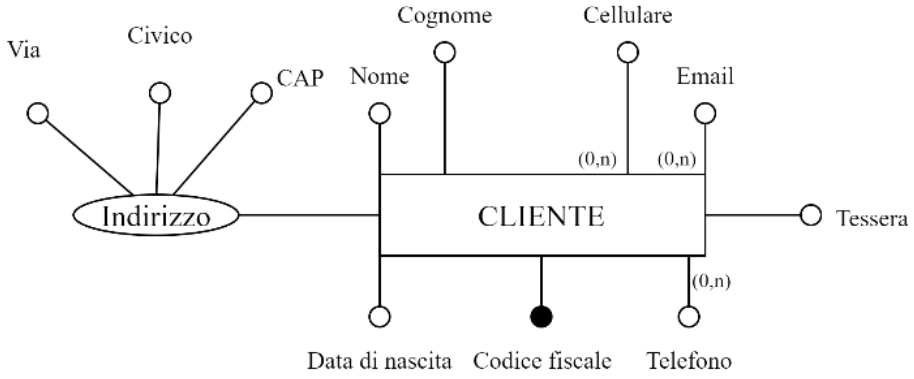
Gli attributi dell'entità *Copia di film* sono stati omessi per semplicità

Viene ora introdotta l'entità reparto per rappresentare la locazione nel negozio della copia fisica del film. Per specificare la posizione all'interno del reparto della copia fisica si è scelto di utilizzare un attributo di relazione piuttosto che un attributo collegato all'entità *Copia di film* per enfatizzare il fatto che la posizione è relativa all'interno del settore e non assoluta della copia all'interno del negozio.



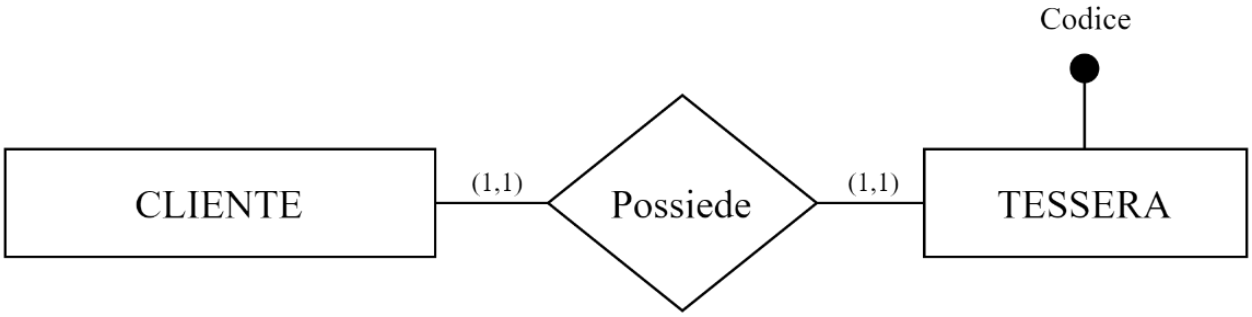
Gli attributi e le generalizzazioni dell'entità *Copia di film* sono stati omessi per semplicità

Per poter noleggiare una copia i clienti devono essere registrati nel sistema. Si introduce a tale scopo l'entità cliente con i relativi attributi derivati dalle specifiche.



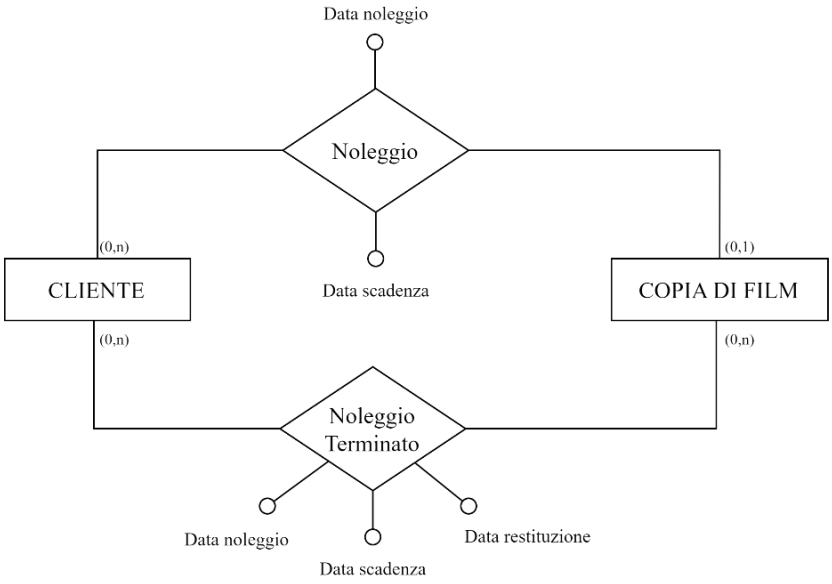
Poiché i recapiti del cliente sono in numero arbitrario, si è scelto di rappresentarli come attributi multivalore con cardinalità minima zero e utilizzare una regola aziendale per specificare l'obbligatorietà di almeno uno dei tre.

L'attributo *tessera* indica la tessera cliente che il negozio dà al cliente a seguito della registrazione. Si vuole però dare una maggiore rilevanza a tale informazione e per questo si applica una reificazione dell'attributo.



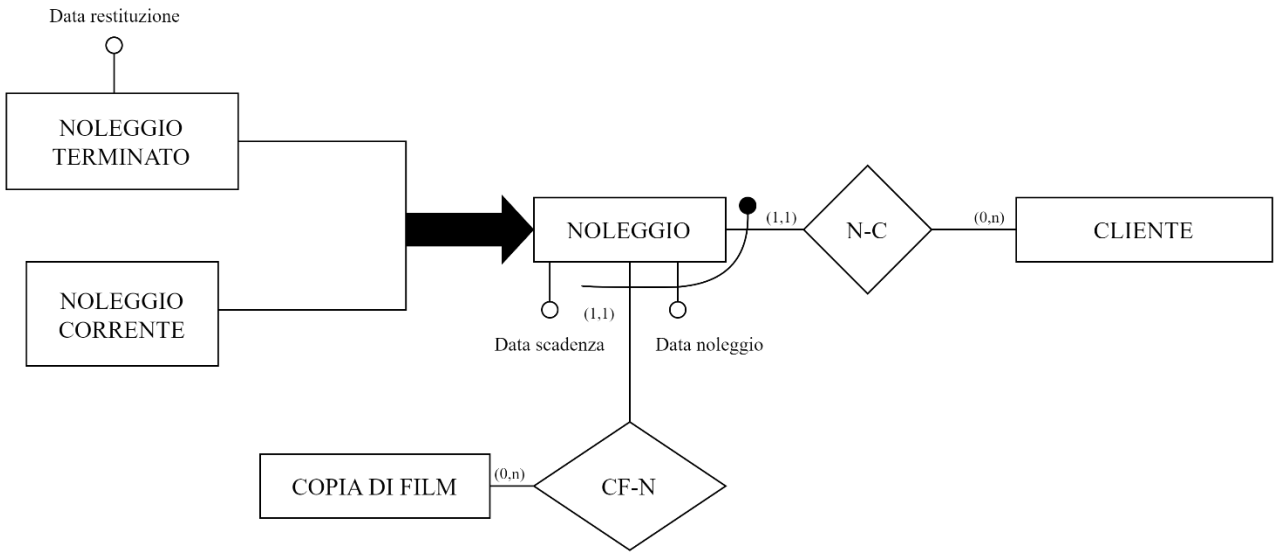
Gli attributi dell'entità *Cliente* sono stati omessi per semplicità

Una prima idea per rappresentare i vari noleggi delle copie è la seguente:



Gli attributi e le generalizzazioni dell'entità *Copia di film* e *Cliente* sono stati omessi per semplicità

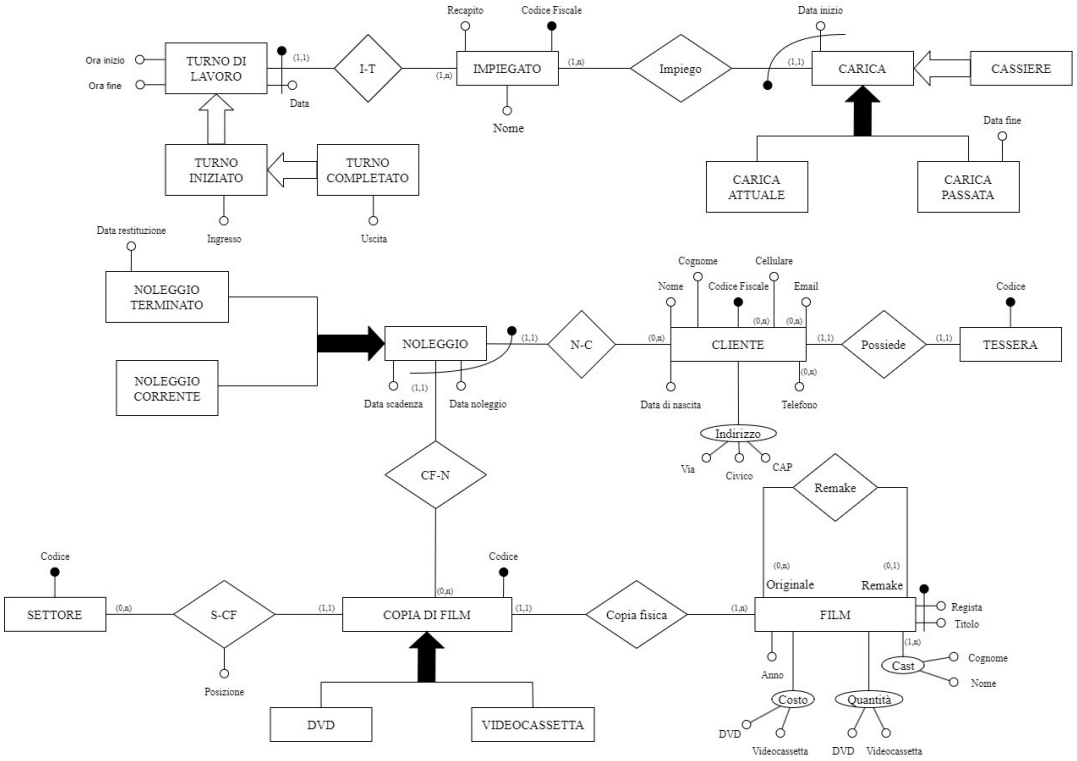
Questa soluzione presenta però un problema: un cliente non può noleggiare due volte la stessa copia di un film. Per risolvere il problema viene reificata la relazione e viene applicato il pattern di storicizzazione.



Gli attributi e le generalizzazioni dell'entità *Copia di film* e *Cliente* sono stati omessi per semplicità

In questo modo catturiamo la possibilità di un cliente di noleggiare la stessa copia fisica di un film più volte. Come nel caso precendente, si utlizza una regola aziendale per specificare che ogni copia di film può essere noleggiata da un solo cliente per volta. Si noti che, per scelta progettuale, la stessa copia fisica di un film non può essere noleggiata due volte dallo stesso cliente nello stesso giorno.

Integrazione finale



Regole aziendali

L'attributo *Ora fine* dell'entità *Turno di lavoro* deve essere maggiore dell'attributo *Ora inizio*.

L'attributo *Uscita* dell'entità *Turno Completato* deve essere maggiore dell'attributo *Ingresso*.

L'attributo *Data scadenza* dell'entità *Noleggio* deve essere maggiore dell'attributo *Data noleggio*.

L'attributo *Data restituzione* dell'entità *Noleggio terminato* deve essere maggiore o uguale dell'attributo *Data noleggio*.

Il cliente deve avere almeno un recapito tra *Cell*ulare, *E-mail* e *Telefono*.

Le quantità di *DVD* e *Videocassette* devono essere sempre maggiori o uguali di 0.

Un *Impiegato* deve avere una sola *Carica attuale* in ogni istante.

Una *Copia di film* può avere al più un *Noleggio corrente* in ogni istante.

Il proprietario non può inserire o eliminare un *Turno di lavoro* che fa riferimento a mesi precedenti rispetto alla data di esecuzione dell'operazione.

Dizionario dei dati

Per le entità figlie sono riportati solo gli attributi e gli identificatori aggiuntivi rispetto all'entità padre.

Entità	Descrizione	Attributi	Identificatori
Impiegato	Persona fisica che lavora nel negozio	Nome, Recapito	Codice Fiscale
Carica	Ruolo rivestito da un impiegato		Data Inizio, Impiego
Cassiere	Specializza *Carica.* Tipo di possibile carica che un impiegato può rivestire.		
Carica attuale	Specializza *Carica*. Ruolo corrente rivestito da un impiegato		
Carica passata	Specializza *Carica*. Ruoli passati rivestiti da un impiegato	Data fine	
Turno di lavoro	Giorno e ora in cui l'impiegato deve lavorare	Ora inizio, Ora fine	Data, I-T
Turno iniziato	Specializza *Turno di lavoro*. Indica i turni di lavoro che sono stati iniziati da un impiegato, con il relativo orario di ingresso, ma che non sono ancora stati completati	Ingresso	
Turno completato	Specializza *Turno iniziato*. Indica i turni di lavoro che sono stati completati da un impiegato con il relativo orario di uscita	Uscita	
Film	Film disponibile all'interno del negozio. Non rappresenta la copia fisica ma solo il concetto generale di film presente nel catalogo.	Anno, Costo, Quantità, Cast	Regista, Titolo
Copia di Film	Copia fisica di un film presente nel catalogo e che può essere effettivamente noleggiata		Codice
DVD	Specializza *Copia di Film*. Tipo di copia fisica che può essere noleggiata		
Videocassetta	Specializza *Copia di Film*. Tipo di copia fisica che può essere noleggiata		
Settore	Luogo del negozio in cui sono presenti le copie fisiche dei film		Codice
Noleggio	Noleggi effettuati dai clienti	Data scadenza	Data noleggio, CF-N, N-C
Noleggio terminato	Specializza *Noleggio*. Noleggi effettuati dai clienti in cui la copia fisica è stata restituita	Data restituzione	
Noleggio corrente	Specializza *Noleggio*. Noleggi effettuati dai clienti in cui la copia fisica non è stata ancora restituita		
Cliente	Persona fisica che noleggia le copie dei film	Nome, Cognome, Indirizzo, Data di nascita, Cellulare, E-mail, Telefono	Codice Fiscale
Tessera	Tessera cliente che viene consegnata al cliente al momento della registrazione		Codice

4. Progettazione logica

Volume dei dati

Si suppone di mantenere i dati dei noleggi e delle cariche per non più di 10 anni, mentre i turni di lavoro vengono mantenuti per massimo 5 anni.

Si stima che il negozio abbia a disposizione circa 50 impiegati, di cui 20 cassieri, e che ognuno di questi cambi ruolo mediamente 2 volte in 10 anni.

Ci si aspetta che il datore di lavoro inserisca i turni di lavoro con un preavviso di circa 2 mesi.

Per ogni film a catalogo, sono presenti in negozio circa 30 copie, divise equamente tra DVD e videocassette.

Si assume che circa il 15% dei film presenti nel catalogo siano remake di altri film presenti.

Per quanto riguarda il noleggio, ci si aspetta che ogni cliente registrato noleggi circa 2 film ogni mese.

Giornalmente vengono noleggiate circa 500 copie con un tempo di restituzione medio che si aggira attorno ai 5 giorni (la scadenza viene fissata in media a 7 giorni dalla data di noleggio).

Concetto nello schema	Tipo[1]	Volume atteso
Impiegato	E	50
Carica	E	150
Cassiere	E	20
Carica attuale	E	50
Carica passata	E	100
Turno di lavoro	E	91.250
Turno iniziato	E	50
Turno completato	E	88.200
Film	E	500
Copia di Film	E	15.000
DVD	E	7.500
Videocassetta	E	7.500
Settore	E	100
Noleggio	E	1.825.000
Noleggio terminato	E	1.822.500
Noleggio corrente	E	2.500
Cliente	E	7.500
Tessera	E	7.500
Impiego	R	150
I-T	R	91.250
Remake	R	75

[1] Indicare con E le entità, con R le relazioni

Copia fisica	R	15.000
S-CF	R	15.000
CF-N	R	1.825.000
N-C	R	1.825.000
Possiede	R	7.500

Tavola delle operazioni

Le operazioni di inserimento di *Film*, *Copia di film* e *Settore* non sono state inserite poiché ritenute banali, vengono infatti riportate le sole operazioni descritte nelle specifiche e che hanno un ruolo di importanza per il sistema.

Si noti che la frequenza attesa tiene conto anche delle ripetizioni delle operazioni sulle singole entità (es. *Inserisci turno di lavoro* è stimato come 1 al mese per ogni impiegato).

L'operazione di stampa dei report viene eseguita singolarmente e restituisce i dati di ogni impiegato presente.

Cod.	Descrizione	Frequenza attesa
P1	Inserisce impiegato	1 / anno
P2	Modifica carica	10 / anno
P3	Inserisci turno di lavoro	50 / mese
P4	Report mensile	2 / mese
P5	Report annuale	2 / anno
I1	Timbra ingresso	50 / giorno
I2	Timbra uscita	50 / giorno
C1	Registra cliente	10 / mese
C2	Inserisci recapito cliente	2 / giorno
C3	Registra noleggio	500 / giorno
C4	Registra restituzione	500 / giorno
C5	Lista noleggi scaduti	500 / mese

Costo delle operazioni

Per il calcolo del costo delle operazioni si stima che un accesso in lettura abbia costo 1 e un accesso in scrittura abbia costo 2.

Operazione P1:

L'operazione è composta da una prima fase in cui viene inserita l'anagrafica del nuovo impiegato e una seconda fase in cui viene inserita la carica attuale dell'impiegato. L'operazione, dunque, è composta da 3 scritture.

Costo singola operazione: 3 * 2 = 6 accessi

Costo totale: 6 accessi / anno

Operazione P2:

Per eseguire l'operazione deve essere rimossa l'entità *Carica attuale* per poi essere inserita come *Carica passata*; va poi inserita la nuova *Carica attuale* e la relativa associazione *Impiego* con l'*Impiegato*. Per poter essere riscritta, la *Carica* attuale deve essere prima letta. Vengono dunque eseguiti due accessi in scrittura e uno in lettura all'entità *Carica attuale*, uno per rimuovere la carica attuale e uno per inserire quella nuova, uno in scrittura sulla relazione *Impiego* e infine un accesso in scrittura su *Carica passata*.

Costo singola operazione: 1 + 2 * 2 + 2 + 2 = 9 accessi

Costo totale: 900 / anno

Operazione P3:

Viene prima inserito il turno di lavoro e successivamente viene collegato all'impiegato. L'operazione ha dunque bisogno di un accesso in scrittura a *Turno di lavoro* e uno a *I-T*. L'operazione va ripetuta per ogni giorno del mese.

Costo singola operazione: 2 * 30 + 2 * 30 = 120 accessi

Costo totale: 6.000 accessi / mese

Operazione P4:

L'operazione deve accedere circa 30 volte all'entità *Turno completato* per ottenere le informazioni circa le ore lavorate dal dipendente nel mese e deve ripetere quest'azione per ogni dipendente (che si sono stimati essere in media 50). Prima di accedere all'entità *Turno completato* è necessario accedere in lettura all'entità *Impiegato* e successivamente all'associazione *I-T* lo stesso numero di volte di *Turno completato*.

Costo singola operazione: 1 * 50 + 30 * 1 * 2 * 50 = 3.050 accessi

Costo totale: 6.100 accessi / mese

Operazione P5:

L'operazione deve accedere circa 365 volte all'entità *Turno completato* per ottenere le informazioni circa le ore lavorate dal dipendente nell'anno e deve ripetere quest'azione per ogni dipendente (che si sono stimati essere in media 50). Prima di accedere all'entità *Turno completato* è necessario accedere in lettura all'entità *Impiegato* e successivamente all'associazione *I-T* lo stesso numero di volte di *Turno completato*.

Costo singola operazione: 1 * 50 + 365 * 1 * 2 * 50 = 36.550 accessi

Costo totale: 73.100 accessi / anno

Operazione I1:

L'operazione deve accedere al *Turno di lavoro* giornaliero dell'*Impiegato* per spostarlo su *Turno iniziato*. L'operazione richiede dunque un accesso in scrittura e uno in lettura su *Turno di lavoro* per rimuovere il turno e un ulteriore accesso in scrittura su *Turno iniziato* per ricreare il turno.

Costo singola operazione: 1 + 2 + 2 = 5 accessi

Costo totale: 250 accessi / giorno

Operazione I2:

L'operazione deve accedere al *Turno iniziato* dell'*Impiegato* per spostarlo su *Turno completato*. L'operazione richiede dunque un accesso in scrittura e uno in lettura su *Turno iniziato* per rimuovere il turno e un ulteriore accesso in scrittura su *Turno completato* per ricreare il turno.

Costo singola operazione: 2 + 2 = 5 accessi

Costo totale: 250 accessi / giorno

Operazione C1:

L'operazione è composta da una prima fase in cui viene inserita l'anagrafica del cliente e una seconda fase in cui viene inserita la tessera cliente che gli è stata consegnata. Si hanno in tutto 3 accessi in scrittura, su *Cliente, Possiede* e *Tessera*. Si ha poi un ulteriore accesso in scrittura per inserire un recapito. Si noti che si sta supponendo che ogni cliente, durante la registrazione, inserisca in media un solo recapito.

Costo singola operazione: 2 + 2 + 2 + 2 = 8 accessi

Costo totale: 800 accessi / mese

Operazione C2:

Viene inserito un nuovo recapito ad un *Cliente* esistente. L'operazione è composta da una sola scrittura.

Costo singola operazione: 2 = 2 accessi

Costo totale: 4 accessi / giorno

Operazione C3:

Viene inserito un nuovo noleggio nell'entità *Noleggio corrente* con le rispettive relazioni. Prima di eseguire gli accessi in scrittura su *CF-N*, *Noleggio corrente* e *N-C* si deve eseguire un accesso in lettura su *Noleggio corrente* per capire se la *Copia di film* è attualmente noleggiata.

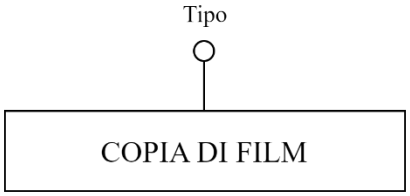
Costo singola operazione: 1 + 2 + 2 + 2 = 7 accessi
Costo totale: 3.500 accessi / giorno

Operazione C4:
Per eseguire l'operazione deve essere rimossa l'entità *Noleggio corrente* collegata alla *Copia di film* di cui si sta effettuando la restituzione per poi inserirla in *Noleggio terminato*. Vengono dunque eseguiti due accessi in scrittura e uno in lettura per reperire i dati su *Noleggio corrente*.
Costo singola operazione: 1 + 2 + 2 = 5
Costo totale: 2.500 accessi / giorno

Operazione C5:
Per eseguire l'operazione, si deve accedere in lettura ai *Noleggi correnti* per trovare tutti i noleggi scaduti. Dalle stime fatte i noleggi scaduti sono in media 100 a settimana per cui, per questi, si deve accedere alle *Copie di film* a cui sono collegati e ai *Clienti* che hanno effettuato tali noleggi. Per ognuno dei noleggi scaduti quindi si deve accedere in lettura a *CF-N* e successivamente a *Copia di film* per poi accedere in lettura a *N-C* ed infine a *Cliente*.
Costo singola operazione: 100 * 1 * 5 = 500 accessi
Costo totale: 250.000 accessi / mese

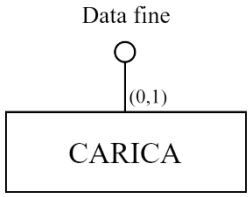
Ristrutturazione dello schema E-R

Lo schema E-R prodotto presenta una ridondanza sull'attributo composto *quantità*. Tale attributo, infatti, può essere calcolato a partire dalla relazione *Copia fisica*. Dalla tavola delle operazioni principali si osserva che tale attributo non è acceduto dalle operazioni significative per il sistema per cui tale ridondanza può essere eliminata.
Si è scelto, poiché ogni tessera è collegata ad un solo cliente e ogni cliente riceve la propria tessera al momento della registrazione (come da specifica), di accorpare le entità per risparmiare 2 accessi in scrittura sull'operazione C1. Inoltre, poiché l'entità presenta ora due identificatori, viene scelto come identificatore principale la tessera cliente.
Per quanto riguarda l'eliminazione delle generalizzazioni, si osserva che la generalizzazione su *Copia di film* può essere risolta semplicemente inserendo un attributo *tipo* sull'entità padre poiché non ci sono particolari operazioni che fanno riferimento alle entità figlie. Tale attributo è necessario per discriminare il tipo di copia fisica presente all'interno del negozio.



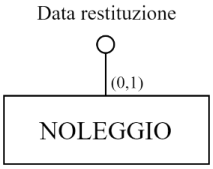
Gli ulteriori attributi dell'entità *Copia di film* sono stati omessi per semplicità

Lo stesso ragionamento può essere applicato all'entità *Carica* con la generalizzazione parziale in *Cassiere*. Viene dunque rimossa e aggiunto un attributo mansione all'entità.
Nello schema compaiono ora le generalizzazioni relative alla storicizzazione dell'entità *Carica* e dell'entità *Noleggio*.
La risoluzione sull'entità *Carica* può essere fatta osservando in particolare l'operazione P2 in cui una *Carica attuale* deve essere spostata a *Carica passata*. Si osserva che l'operazione di spostamento necessita di 1 operazione di lettura e 2 operazioni di scrittura, una per cancellare l'entità da *Carica attuale* e una per scriverla in *Carica passata*, lasciando immutate le relazioni che la coinvolgono. Andando a risolvere la generalizzazione aggiungendo l'attributo *data fine* di *Carica passata* su *Carica* e rendendolo opzionale (verrà impostato a *null* per la *Carica attuale* fungendo dunque da discriminante tra le due entità figlie di *Carica*), si va a risparmiare un accesso in scrittura per singola operazione P2 effettuata, poiché lo spostamento si riduce a valorizzare l'attributo *data fine* della *Carica attuale*. Si ottiene dunque:



Gli ulteriori attributi dell'entità *Carica* sono stati omessi per semplicità

Il costo dell'operazione P2 diviene dunque di 7 accessi per singola operazione. L'accesso in lettura non può essere eliminato poiché deve essere verificato che la data di fine sia effettivamente maggiore della data di inizio.
Si possono fare considerazione analoghe per quanto riguarda la generalizzazione su *Noleggio* andando a prendere in esame l'operazione C4. Viene aggiunto dunque l'attributo *data restituzione* come opzionale sull'entità *Noleggio* e otteniamo:

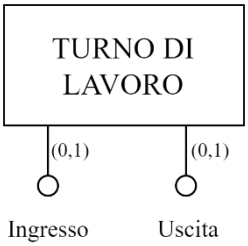


Gli ulteriori attributi dell'entità *Noleggio* sono stati omessi per semplicità

Così facendo il costo dell'operazione è ora di 3 accessi per singola operazione.

Lo stesso vale per le generalizzazioni su *Turno di lavoro*. In questo caso, le operazioni I1 e I2 comportano spostamenti tra l'entità *Turno di lavoro* e le entità *Turno iniziato* e *Turno completato*. Come discusso in precedenza, poiché anche in questo caso le relazioni restano invariate, rappresentare la generalizzazione attraverso gli attributi delle entità figlie sull'entità padre rendendoli opzionali ci permette di risparmiare 1 accesso in scrittura per singolo spostamento. Inoltre, sull'operazione I1 si può risparmiare anche l'accesso in lettura. Otteniamo quindi che il costo dell'operazione I1 diventa 2 accessi per singola operazione, mentre quello di I2 diventa 3 accessi. Per l'operazione I2 è necessario l'accesso in lettura poiché deve essere verificato che l'orario di uscita sia effettivamente maggiore di quello di entrata.

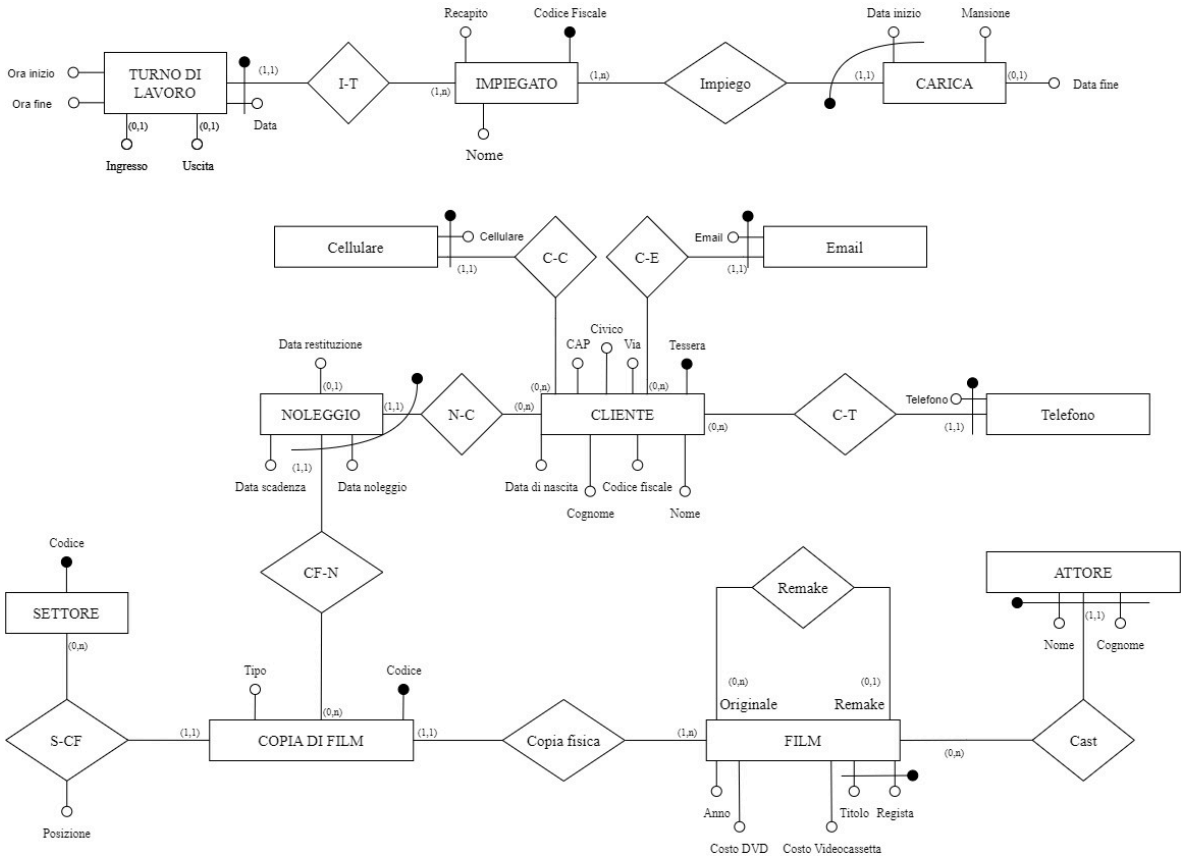
In questa ristrutturazione però dobbiamo tener presente che un'entità *Turno di lavoro* ha gli attributi *ingresso* e *uscita* con valori nulli, l'entità *Turno iniziato* avrà come valore nullo l'attributo *uscita* mentre per un'entità *Turno completato* gli attributi saranno tutti valorizzati.



Gli ulteriori attributi dell'entità *Turno di lavoro* sono stati omessi per semplicità

Le entità con attributi multivalore sono state risolte reificando tali attributi.

Lo schema ristrutturato è il seguente:



Trasformazione di attributi e identificatori

Non sono presenti nello schema attributi o identificatori esterni da modificare.

Traduzione di entità e associazioni

IMPIEGATO(<u>codice_fiscale</u>, nome, recapito)

CARICA(<u>impiegato, inizio</u>, fine*, mansione)

TURNO_LAVORO(<u>impiegato, data</u>, ora_inizio, ora_fine, ingresso*, uscita*)

SETTORE(<u>codice</u>)

CLIENTE(<u>tessera</u>, codice_fiscale, nome, cognome, data_nascita, indirizzo_via, indirizzo_cap, indirizzo_civico)

CELLULARE(<u>cliente, cellulare</u>)

EMAIL(<u>cliente , email</u>)

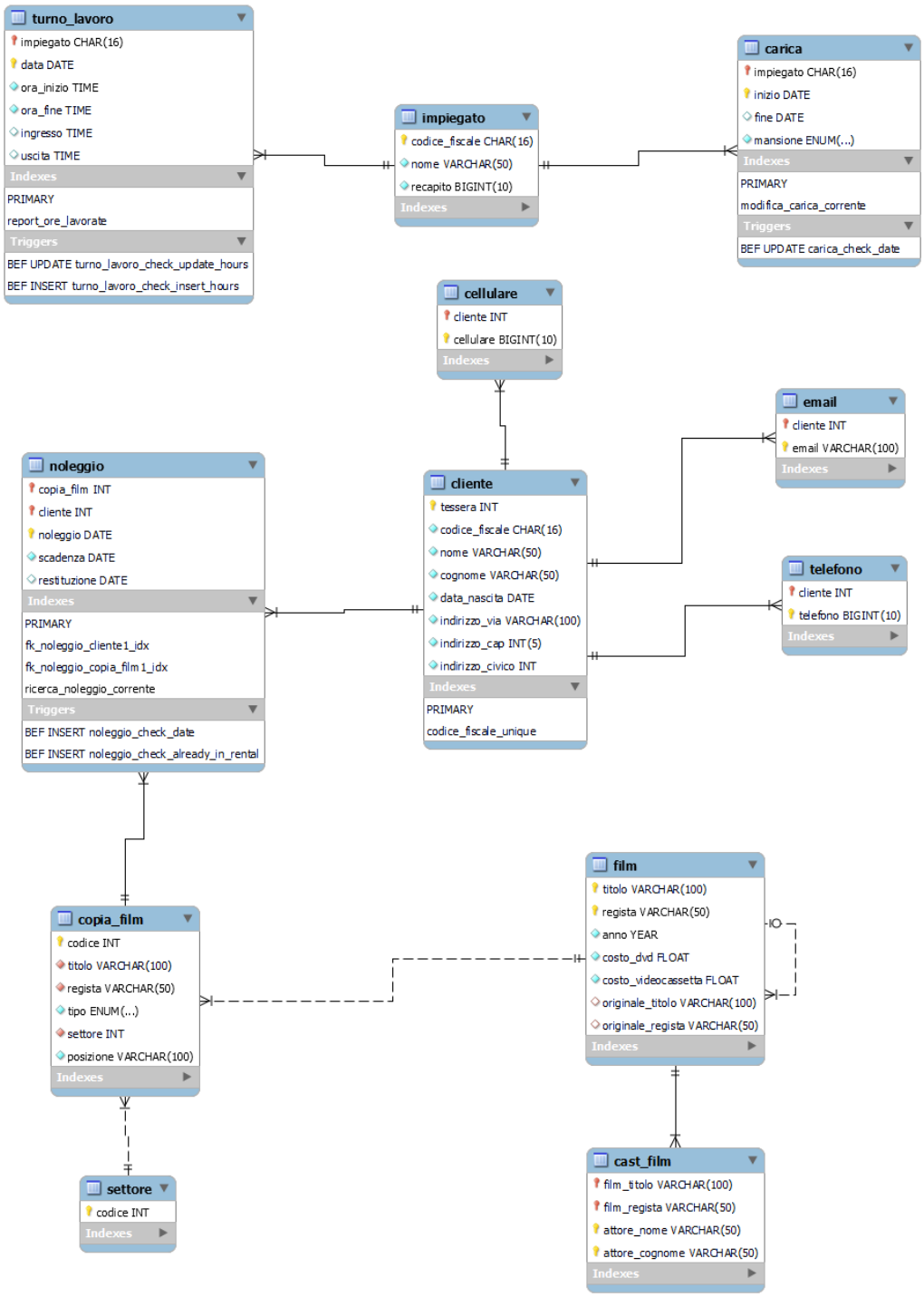
TELEFONO(<u>cliente , telefono</u>)

FILM(<u>titolo, regista</u>, anno, costo_dvd, costo_videocassetta, originale_titolo*, originale_regista*)

CAST_FILM(film_titolo, film_regista , attore_nome, attore_cognome)

COPIA_FILM(codice, titolo, regista, tipo, settore, posizione)

NOLEGGIO(copia_film, cliente, noleggio, scadenza, restituzione*)



Normalizzazione del modello relazionale

Il modello relazionale prodotto è in forma normale di Boyce e Codd. Per dimostrarlo, si può osservare che la condizione di normalità di Boyce e Codd è verificata per ogni relazione presente nello schema.

Si può partire ad eseguire questa analisi dalle relazioni più semplici: CELLULARE, EMAIL, TELEFONO, CAST_FILM e SETTORE. Si osserva subito che tali relazioni non presentano ulteriori attributi oltre quelli chiave, per cui sono necessariamente in forma normale.

Si osserva ora che, per come sono state progettate, le ulteriori relazioni presenti nello schema presentano una sola dipendenza funzionale in cui la chiave primaria di ognuna determina i restanti attributi.

Nella relazione CLIENTE ogni attributo che compare che non sia nella chiave primaria fa riferimento a qualcosa legato alla singola istanza di cliente individuato dalla *tessera*. Lo stesso discorso può essere fatto per la relazione FILM, COPIA_FILM e IMPIEGATO.

Nelle relazioni CARICA invece compare nella chiave anche la data di inizio di quella carica poiché oltre a dipendere dall'impiegato a cui essa è associata, i restanti attributi dipendono anche dalla data di inizio di validità della carica. Lo stesso discorso può essere esteso alle relazioni TURNO_LAVORO e NOLEGGIO.

Nel corso della progettazione si è scelto dunque di mantenere normalizzato l'intero schema per evitare qualsiasi anomalia. Durante lo studio sul costo delle operazioni non sono state rilevate trasformazioni tali che potessero portare lo schema a non essere in forma normale allo scopo di ottenere significativi aumenti prestazionali.

5. Progettazione fisica

Utenti e privilegi

Si è scelto di implementare attraverso un utente specifico la procedura di login. Fino a quando non verrà effettuato l'accesso al sistema non potranno essere eseguite altre procedure.
Per mantenere le credenziali di accesso è stata aggiunta una tabella login. Tale relazione mantiene un riferimento all'impiegato presente nel sistema a cui le credenziali fanno riferimento. Se tale riferimento è *null* si sta identificando il proprietario del negozio (come scelta progettuale si è scelto di dare la possibilità di avere più di un account ti tipo *proprietario*).
Una volta effettuato l'accesso verrà modificato il profilo con cui ci si connette al DBMS. Ogni profilo rappresenta una differente persona che opera all'interno del negozio, in particolare sono state individuate 3 figure principali:

1. Proprietario: gestisce il personale, i turni di lavoro, i settori, il catalogo dei film e delle relative copie.
2. Impiegato: può solo timbrare il cartellino.
3. Cassiere: può fare le stesse operazioni del ruolo impiegato e in più può gestire i noleggi e i clienti.

Il ruolo di cassiere o di impiegato viene selezionato in base alla carica attuale del dipendente.
I relativi utenti nella base di dati sono (con le rispettive procedure che possono eseguire):

- sn_login
  - login
- sn_proprietario
  - registraImpiegato
  - listaImpiegati
  - rimuoviImpiegato
  - modificaCarica
  - listaCariche
  - inserisciTurnoLavoro
  - listaTurni
  - rimuoviTurnoLavoro
  - reportMensile
  - reportAnnuale
  - inserisciFilm

  - listaFilm
  - rimuoviFilm
  - inserisciSettore
  - listaSettori
  - rimuoviSettore
  - inserisciCopiaFilm
  - listaCopieFilm
  - rimuoviCopiaFilm
- sn_cassiere
  - timbra
  - listaTurni
  - registraCliente
  - listaClienti
  - rimuoviCliente
  - inserisciRecapitiCliente
  - rimuoviRecapitiCliente
  - registraNoleggio
  - registraRestituzione
  - listaNoleggiCorrenti
  - listaNoleggiScaduti
  - listaNoleggiCopiaFilm
  - listaNoleggiCliente
  - listaNoleggiFilm
- sn_impiegato
  - timbra
  - listaTurni

Strutture di memorizzazione

Tabella login		
Colonna	**Tipo di dato**	**Attributi**[2]
username	Varchar(100)	PK, NN
password	Varchar(32)	NN

[2] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

impiegato	Char(16)	

Tabella impiegato		
Colonna	**Tipo di dato**	**Attributi**[3]
codice_fiscale	Char(16)	PK, NN
nome	Varchar(50)	NN
recapito	Bigint(10)	NN

Tabella carica		
Colonna	**Tipo di dato**	**Attributi**[4]
impiegato	Char(16)	PK, NN
inizio	Date	PK, NN
fine	Date	
mansione	Enum('Cassiere', 'Commesso')	NN

Tabella turno_lavoro		
Colonna	**Tipo di dato**	**Attributi**[5]
impiegato	Char(16)	PK, NN
data	Date	PK, NN
ora_inizio	Time	NN
ora_fine	Time	NN
ingresso	Time	
uscita	Time	

Tabella settore		
Colonna	**Tipo di dato**	**Attributi**[6]
codice	Int	PK, NN

Tabella cliente		
Colonna	**Tipo di dato**	**Attributi**[7]
tessera	Int	PK, NN, AI
codice_fiscale	Char(16)	NN, UQ

[3] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[4] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[5] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[6] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[7] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

nome	Varchar(50)	NN
cognome	Varchar(50)	NN
data_nascita	Date	NN
indirizzo_via	Varchar(100)	NN
indirizzo_cap	Int(5)	NN
indirizzo_civico	Int	NN

Tabella cellulare		
Colonna	**Tipo di dato**	**Attributi**[8]
cliente	Int	PK, NN
cellulare	Bigint(10)	PK, NN

Tabella email		
Colonna	**Tipo di dato**	**Attributi**[9]
cliente	Int	PK, NN
email	Varchar(100)	PK, NN

Tabella telefono		
Colonna	**Tipo di dato**	**Attributi**[10]
cliente	Int	PK, NN
telefono	Bigint(10)	PK, NN

Tabella film		
Colonna	**Tipo di dato**	**Attributi**[11]
titolo	Varchar(100)	PK, NN
regista	Varchar(50)	PK, NN
anno	Year	NN
costo_dvd	Float	NN
costo_videocassetta	Float	NN
originale_titolo	Varchar(100)	
originale_regista	Varchar(50)	

Tabella cast_film		
Colonna	**Tipo di dato**	**Attributi**[12]

[8] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[9] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[10] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[11] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

film_titolo	Varchar(100)	PK, NN
film_regista	Varchar(50)	PK, NN
attore_nome	Varchar(50)	PK, NN
attore_cognome	Varchar(50)	PK, NN

Tabella copia_film		
Colonna	**Tipo di dato**	**Attributi**[13]
codice	Int	PK, NN
titolo	Varchar(100)	NN
regista	Varchar(50)	NN
tipo	Enum('DVD', 'Videocassetta')	NN
settore	Int	NN
posizione	Varchar(100)	NN

Tabella noleggio		
Colonna	**Tipo di dato**	**Attributi**[14]
copia_film	Int	PK, NN
cliente	Int	PK, NN
noleggio	Date	PK, NN
scadenza	Date	NN
restituzione	Date	

Indici

L'operazione P2 richiede di aggiornare la tupla relativa alla carica attuale dell'impiegato. Per velocizzare la ricerca di tale tupla si sceglie di inserire un indice sulla relazione *Carica*. Poiché tale ricerca viene effettuata sui campi *impiegato* e *fine* l'indice conterrà proprio tali campi.

Tabella carica	
Indice modifica_carica_corrente	**Tipo**[15]**:**
impiegato, fine	IDX

Sulla relazione *turno_lavoro* le operazioni P4 e P5 necessitano di accessi in lettura per reperire i dati di *ingresso* e *uscita* degli impiegati. Per velocizzare questa richiesta viene inserito un indice su tali

---

[12] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[13] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[14] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

[15] IDX = index, UQ = unique, FT = full text, PR = primary.

campi. Per ottimizzare le operazioni di timbratura I1 e I2, si decide di inserire all'indice anche l'attributo *data* e *impiegato*. Inoltre, in questo modo le operazioni P4 e P5 possono estrarre tutti i dati di loro interesse direttamente dall'indice.

Tabella turno_lavoro	
Indice report_ore_lavorate	**Tipo[16]:**
impiegato, data, ingresso, uscita	IDX

Durante la progettazione si è scelto di utilizzare come chiave primaria l'attributo *tessera* per la relazione cliente, ma ogni persona può essere identificata anche tramite *codice_fiscale*. Si vuole quindi rendere univoco tale campo e per farlo viene utilizzato un indice.

Tabella cliente	
Indice codice_fiscale_unique	**Tipo[17]:**
codice_fiscale	UQ

Per aumentare le prestazioni dell'operazione C5 viene introdotto un indice sugli attributi *restituzione* e *scadenza* per reperire più velocemente le tuple che fanno riferimento a noleggi in corso e noleggi scaduti.

Tabella noleggio	
Indice ricerca_noleggio_corrente	**Tipo[18]:**
restituzione, scadenza	IDX

Trigger

I trigger utilizzati sono stati realizzati per andare a controllare i vincoli imposti dalle regole aziendali. In particolare, sono stati realizzati trigger di tipo *before* insert e *before update* per tutte quelle regole che richiedevano che un attributo fosse maggiore rispetto ad un altro (in particolare, i campi di tipo data e orario).

Tabella turno_lavoro: BEFORE_INSERT

Il seguente trigger implementa il controllo che ogni *ora_fine* della relazione TURNO_LAVORO sia maggiore di *ora_inizio*:

[16] IDX = index, UQ = unique, FT = full text, PR = primary.

[17] IDX = index, UQ = unique, FT = full text, PR = primary.

[18] IDX = index, UQ = unique, FT = full text, PR = primary.

```
CREATE TRIGGER `sistema_noleggio`.`turno_lavoro_check_insert_hours` BEFORE INSERT ON `turno_lavoro` FOR EACH ROW
BEGIN
	IF New.ora_fine <= New.ora_inizio THEN
		SIGNAL SQLSTATE '45000' SET message_text = "L'orario di uscita deve essere maggiore di quello di entrata";
    END IF;
END
```

Tabella turno_lavoro: BEFORE_UPDATE

Il seguente trigger implementa il controllo che ogni *uscita* della relazione TURNO_LAVORO sia maggiore di *ingresso*:

```
CREATE TRIGGER `sistema_noleggio`.`turno_lavoro_check_update_hours` BEFORE UPDATE ON `turno_lavoro` FOR EACH ROW
BEGIN
	IF New.ingresso is not null AND New.uscita <= New.ingresso THEN
		SIGNAL SQLSTATE '45000' SET message_text = "L'orario di uscita deve essere maggiore di quello di entrata";
	END IF;
END
```

Tabella carica: BEFORE_UPDATE

Il seguente trigger implementa il controllo che ogni *fine* della relazione CARICA sia maggiore di *inizio* e della data corrente in cui modificata la carica:

```
CREATE TRIGGER `sistema_noleggio`.`carica_check_date` BEFORE UPDATE ON `carica` FOR EACH ROW
BEGIN

	IF New.fine < CURDATE() THEN
		SIGNAL SQLSTATE '45000' SET message_text = "La data di fine deve essere maggiore o uguale della data odierna";
    END IF;

	IF New.fine <= Old.inizio THEN
		SIGNAL SQLSTATE '45000' SET message_text = "La data di fine deve essere maggiore della data di inizio";
```

```
    END IF;
END
```

Tabella noleggio: BEFORE_INSERT

Il seguente trigger implementa il controllo che ogni *scadenza* della relazione NOLEGGIO sia maggiore della data di *noleggio*:

```
CREATE TRIGGER `sistema_noleggio`.`noleggio_check_date` BEFORE INSERT ON `noleggio` FOR EACH ROW
BEGIN
	IF New.scadenza <= New.noleggio THEN
		SIGNAL SQLSTATE '45000' SET message_text = "La data di scadenza deve essere maggiore della data di noleggio";
	END IF;
END
```

Tabella noleggio: BEFORE_INSERT

Si è scelto di implementare anche il controllo sulla disponibilità di una copia per il noleggio tramite trigger. In particolare:

```
CREATE TRIGGER `sistema_noleggio`.`noleggio_check_already_in_rental` BEFORE INSERT ON `noleggio` FOR EACH ROW
BEGIN
	declare var_check TINYINT;
	SELECT count(*) FROM noleggio_corrente WHERE copia_film = New.copia_film INTO var_check;
	if var_check > 0 then
			signal sqlstate '45000' set message_text = 'La copia è attualmente noleggiata';
	end if;
END
```

Eventi

Gli eventi istanziati vengono utilizzati per rimuovere i noleggi e le cariche vecchi più di 10 anni e i turni di lavoro vecchi più di 5 anni.

Entrambi gli eventi vengono schedulati, durante la configurazione, il 1° gennaio dell'anno successivo, in maniera tale da eseguire quando il carico di lavoro del sistema è quasi pari a zero e

non ci sono operazioni eseguite su di esso da parte degli utilizzatori. Una volta terminati vengono rischedulati automaticamente dal sistema per l'anno successivo.
Non ci sono procedure che utilizzano eventi effimeri.
Prima della loro istanziazione viene attivato lo scheduler del DBMS tramite il comando:

```
SET GLOBAL event_scheduler = on;
```

L'evento di pulizia per noleggi e cariche è il seguente:

```
CREATE EVENT IF NOT EXISTS `sistema_noleggio`.`cleanup_10`
ON SCHEDULE
    EVERY 1 YEAR
    STARTS CONCAT(extract(year from CURDATE() + INTERVAL 1 year), '-01-01 01:00:00')
ON COMPLETION PRESERVE
DO BEGIN
    DELETE FROM `noleggio` WHERE `restituzione` < (CURDATE() - INTERVAL 10 YEAR);
    DELETE FROM `carica` WHERE `fine` < (CURDATE() - INTERVAL 10 YEAR);
END
```

L'evento di pulizia per i turni di lavoro è il seguente:

```
CREATE EVENT IF NOT EXISTS `sistema_noleggio`.`cleanup_5`
ON SCHEDULE
    EVERY 1 YEAR
        STARTS CONCAT(extract(year from CURDATE() + INTERVAL 1 year), '-01-01 01:00:00')
ON COMPLETION PRESERVE
DO BEGIN
    DELETE FROM `turno_lavoro` WHERE `data` < (CURDATE() - INTERVAL 5 YEAR);
END
```

Viste

Le viste sono state utilizzate per rappresentare le entità figlie nelle generalizzazioni dello schema concettuale che sono state accorpate all'entità padre. In tal modo si vuole ridurre la necessità da parte delle operazioni che hanno bisogno di utilizzare tali entità di conoscere come sono rappresentate a livello logico. Le uniche procedure che dovrebbero conoscere in che modo sono rappresentati tali dati sono quelle che eseguono le scritture sulle tabelle sottostanti le viste.

Le procedure che utilizzano solo accessi in lettura alle entità figlie dovrebbero essere progettate per accedere solo alle viste.

Vista carica_attuale:

Contiene la carica attualmente valida per ogni impiegato:

```
CREATE VIEW `carica_attuale` AS
SELECT impiegato, mansione FROM carica WHERE (fine IS NULL AND CURDATE() >= inizio) OR
(CURDATE() BETWEEN inizio AND fine);
```

Vista turno_terminato:

Contiene i turni conclusi da un impiegato:

```
CREATE VIEW `turno_terminato` AS
SELECT impiegato, data, ingresso, uscita FROM turno_lavoro WHERE uscita IS NOT NULL;
```

Vista turno_iniziato:

Contiene i turni iniziati da un impiegato:

```
CREATE VIEW `turno_iniziato` AS
SELECT impiegato, data FROM turno_lavoro WHERE ingresso IS NOT NULL AND uscita IS NULL;
```

Vista noleggio_corrente:

Contiene i noleggi correnti delle copie dei film:

```
CREATE VIEW `noleggio_corrente` AS
SELECT * FROM noleggio WHERE restituzione IS NULL;
```

Stored Procedures e transazioni

Alcune transazioni fanno uso della seguente funzione, che per semplicità di lettura viene riportata all'inizio. Questa funzione serve per estrarre i valori da una lista separata da un carattere (il *var_delimiter*) e restituire quello in posizione *var_occurrence*:

Funzione tokenize_string_list:

```
CREATE FUNCTION `tokenize_string_list` (var_string TEXT, var_delimiter CHAR(1),
var_occurrence INT)
```

```
RETURNS TEXT
DETERMINISTIC
BEGIN
    return substring_index(substring_index(var_string, var_delimiter, var_occurrence), var_delimiter, -1);
END
```

Procedura registraImpiegato (operazione P1):

La procedura realizza l'inserimento di un impiegato nel sistema.

Viene realizzata mediante una transazione complessa poiché necessita di inserire in maniera atomica e isolata, oltre i dati anagrafici dell'impiegato, anche le relative informazioni di login e di carica. La procedura si compone infatti di 3 scritture e va in commit solo se tutte e 3 vengono eseguite senza alcun errore.

Per il livello di isolamento si è scelto *read uncommited* poiché non ci sono letture ma solo scritture, per cui non possono presentarsi anomalie.

```
CREATE PROCEDURE `registraImpiegato` (in var_username VARCHAR(100), in var_password VARCHAR(100), in var_role ENUM('Cassiere', 'Commesso'), in var_start_date DATE, in var_cf CHAR(16), in var_name VARCHAR(50), in var_phone BIGINT(10))
begin

    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;

    set autocommit=0;
    set transaction isolation level read uncommitted;
    start transaction;
    INSERT INTO impiegato(codice_fiscale, nome, recapito) VALUES (var_cf, var_name, var_phone);
    INSERT INTO carica(impiegato, inizio,  mansione) VALUES (var_cf, var_start_date, var_role);
    INSERT INTO login (username, password, impiegato) VALUES (var_username, MD5(var_password), var_cf);
    commit;
```

```
    set autocommit=1;
end
```

Procedura modificaCarica (operazione P2):

La procedura permette di cambiare la carica di un impiegato. Per scelta progettuale, si è deciso che la carica attuale non viene cambiata nel giorno stesso in cui viene eseguita la procedura ma il proprietario può decidere in che giorno terminare la carica dell'impiegato e dal giorno seguente far partire la nuova carica. In tale scenario, la carica attuale di un impiegato può dunque non essere necessariamente quella con l'attributo *fine* impostato a *null*.

Anche in questo caso si fa uso di una transazione più complessa, poiché vengono effettuati dei controlli iniziali e delle scritture che necessitano di essere eseguite in maniera atomica e isolata.

La procedura, prima di modificare l'ultima mansione associata al dipendente, controlla che questa sia differente dalla nuova mansione del dipendente e in caso siano uguali manda in abort la transazione.

Se il controllo viene superato, la transazione prima modifica l'ultima carica dell'impiegato inserendo la data di fine di quest'ultima e successivamente inserisce la nuova carica nel sistema.

Per il livello di isolamento si è scelto *serializable* poiché, prima di eseguire le scritture, la transazione esegue una lettura per controllare che la mansione sia differente, per cui possono presentarsi anomalie dovute a transazioni concorrenti che inseriscono anch'esse una nuova carica per quell'impiegato. Si vogliono dunque evitare anomalie di tipo *inserimento fantasma*. Con questo livello di isolamento si evitano inoltre anomalie di tipo *aggiornamento fantasma* dovute a transazioni concorrenti che cercano di aggiornare l'ultima carica inserita di un impiegato. La scelta di un livello di isolamento così alto è data anche dal fatto che essa viene eseguita con una frequenza molto bassa, per cui si preferisce avere una maggiore correttezza d'esecuzione.

```
CREATE PROCEDURE `modificaCarica` (in var_employee CHAR(16), in var_end_date DATE, in
var_role ENUM('Cassiere', 'Commesso'))
begin

    declare var_start_date DATE;
    declare var_current_role ENUM('Cassiere', 'Commesso');

    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
```

```
    end;

    set autocommit=0;
    set transaction isolation level serializable;

    set var_start_date = var_end_date + INTERVAL 1 day;

    start transaction;
    SELECT mansione FROM carica WHERE impiegato = var_employee AND fine IS NULL INTO
var_current_role;
    if var_current_role = var_role
    then
        signal sqlstate '45000' set message_text = 'La nuova mansione deve essere
differente da quella precedente';
    end if;

    UPDATE carica SET fine = var_end_date WHERE impiegato = var_employee AND fine IS
NULL;
    INSERT INTO carica(impiegato, inizio, mansione) VALUES(var_employee, var_start_date,
var_role);
    commit;
    set autocommit=1;

end
```

Procedura inserisciTurnoLavoro (operazione P3):

La procedura viene eseguita per inserire i turni di lavoro di un intero mese per un impiegato.

Prima di eseguire, la transazione controlla che i turni di lavoro non siano già presenti e nel caso ci siano viene mandata in abort.

L'utilizzo di una transazione più complessa è necessario per inserire i turni dell'intero mese in maniera atomica e indivisibile. In particolare, si passa come argomento della procedura una lista di giorni e una lista di orari separati da ';'. Utilizzando la funzione *tokenize_string_list*, vengono estratti tali giorni e orari e inseriti nella relazione. Se un solo inserimento va in errore, tutto l'inserimento va in abort e deve essere rifatto.

Per il livello di isolamento si è scelto *serializable* poiché prima di eseguire le scritture, la transazione esegue una lettura per controllare che il turno di lavoro non sia stato già inserito, per cui possono presentarsi anomalie dovute a transazione concorrenti che provano a scrivere il turno dopo che la

procedura ha letto i dati necessari per il controllo. Si vogliono dunque evitare anomalie di tipo *inserimento fantasma*.

```
CREATE PROCEDURE `inserisciTurnoLavoro` (in var_employee CHAR(16), in var_month TINYINT(2), in var_year YEAR, in var_list_days VARCHAR(84), in var_list_hours VARCHAR(372))
begin

    -- La variabile var_list_days contiene un elenco del tipo 1;2;3;4;...; -> giorni a cui si riferiscono gli orari
    -- La variabile var_list_hours contiene un elenco del tipo 00:00-00:00;00:00-00:00;....; -> orari di inizio e fine turno

    declare var_counter INT;  -- Indice del loop
    declare var_day TINYINT(2); -- Contiene il numero del giorno da inserire
    declare var_hours CHAR(11); -- Contiene gli orari del tipo 00:00-00:00
    declare var_start_hour TIME; -- Contiene gli orari del tipo 00:00:00
    declare var_end_hour TIME; -- Contiene gli orari del tipo 00:00:00

    declare var_check TINYINT; -- Mantiene il controllo per capire se il turno di quel mese è stato già inserito o meno

    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;

    set autocommit=0;
    set transaction isolation level serializable;

    start transaction;

    /**
      * I controlli vengono eseguiti all'interno della procedura e non attraverso un trigger poiché in questo modo viene eseguito una sola volta
      * anziché per ogni riga inserita nella tabella.
      */
```

```
    if CAST(CONCAT(var_year, '-', LPAD(var_month, 2, '0'), '-01') as date) < CAST(CONCAT(extract(year from CURDATE()), '-', LPAD(extract(month from CURDATE()), 2, '0'), '-01') as date) then
        signal sqlstate '45000' set message_text = 'Il mese non può essere minore di quello odierno';
    end if;

    /**
      * Non si può sfruttare l'univocità della chiave primaria per eseguire il controllo poiché se viene inserito un turno in un mese compilato
      * ma in un giorno non presente tale giorno verrà inserito senza generare errori
      */
    SELECT count(*) FROM turno_lavoro WHERE extract(month from data) = var_month AND extract(year from data) = var_year AND impiegato = var_employee INTO var_check;
    if var_check > 0 then
        signal sqlstate '45000' set message_text = 'Turno di lavoro già presente';
    end if;

    set var_counter = 1;
    insert_loop: loop

        set var_hours = tokenize_string_list(var_list_hours, ';', var_counter);
        if var_hours = ''
        then
            leave insert_loop;
        end if;

        set var_day = tokenize_string_list(var_list_days, ';', var_counter);

        set var_start_hour = tokenize_string_list(var_hours, '-', 1);
        set var_end_hour = tokenize_string_list(var_hours, '-', 2);

        INSERT INTO turno_lavoro(impiegato, data, ora_inizio, ora_fine) VALUES (var_employee, CONCAT(var_year, '-', var_month, '-', var_day), var_start_hour, var_end_hour);

        set var_counter = var_counter+1;
    end loop;

    commit;
```

```
    set autocommit=1;
end
```

Procedura reportMensile (operazione P4):

La procedura viene realizzata mediante una semplice ricerca sulla vista *turno_terminato* filtrandola per mese e anno richiesti dal proprietario.

```
CREATE PROCEDURE `reportMensile` (in var_month TINYINT(2), in var_year YEAR)
begin
    SELECT nome, data, TIMEDIFF(uscita, ingresso) as ore_lavorate
    FROM turno_terminato
        JOIN impiegato ON codice_fiscale = impiegato
    WHERE extract(month from data) = var_month AND extract(year from data) = var_year
    ORDER BY impiegato, data;
end
```

Procedura reportAnnuale (operazione P5):

La procedura viene realizzata mediante una semplice ricerca sulla vista *turno_terminato* filtrandola per anno richiesto dal proprietario.

```
CREATE PROCEDURE `reportAnnuale` (in var_year YEAR)
begin
    SELECT nome, extract(month from data) as mese,
SEC_TO_TIME(SUM(TIME_TO_SEC(TIMEDIFF(uscita, ingresso)))) as ore_lavorate
    FROM turno_terminato
        JOIN impiegato ON codice_fiscale = impiegato
    WHERE extract(year from data) = var_year
    GROUP BY impiegato, nome, mese
    ORDER BY impiegato, mese;
end
```

Procedura timbra (operazioni I1 e I2):

La procedura viene realizzata per eseguire la timbratura di ingresso e di uscita degli impiegati.
La proceduta controlla se è presente l'orario di ingresso del dipendente, in tal caso esegue la timbratura in uscita, altrimenti esegue la timbratura in entrata. Come scelta progettuale, si è scelto di dare la possibilità al dipendente di timbrare l'uscita più volte, in caso, per errore, venga timbrata troppo presto rispetto a quello che è poi il reale orario di uscita dal negozio.

Poiché la timbratura può avvenire in genere prima o dopo l'effettivo turno di lavoro, si è scelto di non controllare gli orari di ingresso e uscita rispetto agli orari stabili dal proprietario.
Per il livello di isolamento si è scelto *repeatable read* poiché, per garantire la correttezza dell'operazione, vanno evitate eventuali anomalie di *aggiornamento fantasma*.

```
CREATE PROCEDURE `timbra` (in var_employee CHAR(16))
begin

    declare var_started_job TIME;
    declare var_affected_row INT;

    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;

    set autocommit=0;
    set transaction isolation level repeatable read;


    start transaction;

    SELECT ingresso FROM turno_lavoro WHERE impiegato = var_employee AND data = CURDATE() INTO var_started_job;

    if var_started_job is null
    then

    -- TIMBRATURA IN INGRESSO
    UPDATE turno_lavoro SET ingresso = CURTIME() WHERE impiegato = var_employee AND data = CURDATE();
    SELECT ROW_COUNT() INTO var_affected_row;
    if var_affected_row <= 0 then
        signal sqlstate '45000' set message_text = 'Non c\'è nessun turno di lavoro per oggi';
    end if;
```

```
    else

    -- TIMBRATURA IN USCITA
    UPDATE turno_lavoro SET uscita = CURTIME() WHERE impiegato = var_employee AND data = CURDATE();
    SELECT ROW_COUNT() INTO var_affected_row;
    if var_affected_row <= 0 then
        signal sqlstate '45000' set message_text = 'Non c\'è nessun turno di lavoro per oggi';
    end if;

    end if;
    commit;
    set autocommit=1;

end
```

Procedura _inserisciRecapitiClienti:

La procedura viene invocata dalle altre due procedure per eseguire l'inserimento dei recapiti del cliente. Il livello di isolamento in cui tale procedura opera viene ereditato dal chiamante.
Viene restituito nell'ultimo parametro il numero di recapiti totali inseriti.

```
CREATE PROCEDURE `_inserisciRecapitiCliente` (in var_client INT, in var_mobile_phones TEXT, in var_emails TEXT, in var_phones TEXT, out var_affected_rows INT)
begin

    -- La variabile var_mobile_phones contiene un elenco del tipo 0000000000;0000000000;....; -> numeri di cellulare del cliente
    -- La variabile var_emails contiene un elenco del tipo xxx@xx;xxx@xxx;....; -> indirizzi email del cliente
    -- La variabile var_phones contiene un elenco del tipo 0000000000;0000000000;....; -> numeri fissi del cliente

    declare var_counter INT;
    declare var_aux TEXT;

    set var_affected_rows = 0;

    -- CELLULARE
```

```
set var_counter = 1;
insert_mobile_phones_loop: loop
    set var_aux = tokenize_string_list(var_mobile_phones, ';', var_counter);
    if var_aux = ''
    then
        leave insert_mobile_phones_loop;
    end if;
    INSERT INTO cellulare(cellulare, cliente) VALUES (var_aux, var_client);
    set var_affected_rows = var_affected_rows+1;
    set var_counter = var_counter+1;
end loop;

-- EMAIL

set var_counter = 1;
insert_emails_loop: loop
    set var_aux = tokenize_string_list(var_emails, ';', var_counter);
    if var_aux = ''
    then
        leave insert_emails_loop;
    end if;
    INSERT INTO email(email, cliente) VALUES (var_aux, var_client);
    set var_affected_rows = var_affected_rows+1;
    set var_counter = var_counter+1;
end loop;

-- TELEFONO

set var_counter = 1;
insert_phones_loop: loop
    set var_aux = tokenize_string_list(var_phones, ';', var_counter);
    if var_aux = ''
    then
        leave insert_phones_loop;
    end if;
    INSERT INTO telefono(telefono, cliente) VALUES (var_aux, var_client);
    set var_affected_rows = var_affected_rows+1;
    set var_counter = var_counter+1;
end loop;
```

```
end
```

Procedura registraCliente (operazione C1):

La procedura permette di registrare un nuovo cliente del negozio. In input, oltre i dati anagrafici del cliente, possono essere passati (come liste separate da ';') i recapiti del cliente. Poiché il cliente deve avere almeno un recapito, si è scelto di utilizzare una transazione per controllare se tale recapito viene inserito, altrimenti l'intero inserimento del cliente viene mandato in abort.

Per estrarre i dati dalle liste viene usata anche in questo caso la funzione precedentemente descritta *tokenize_string_list*. I recapiti vengono inseriti utilizzando un'ulteriore procedura che viene richiamata anche nella procedura relativa all'operazione C2 e descritta precedentemente.

Al termine della procedura il sistema genera un numero univoco associato alla tessera cliente che viene restituita al chiamante della procedura tramite il parametro *var_card*.

Poiché la transazione presenta solamente inserimenti, si è scelto come livello di isolamento *read uncommmitted*.

```
CREATE PROCEDURE `registraCliente` (in var_cf CHAR(16), in var_name VARCHAR(50), in var_surname VARCHAR(50), in var_birth_date DATE, in var_address VARCHAR(100), in var_cap INT(5), in var_number INT,
                                    in var_mobile_phones TEXT, in var_emails TEXT, in var_phones TEXT, out var_card INT)
begin

    declare var_affected_rows INT; -- Contiene i recapiti inseriti per il cliente

    -- La variabile var_mobile_phones contiene un elenco del tipo 0000000000;0000000000;....; -> numeri di cellulare del cliente
    -- La variabile var_emails contiene un elenco del tipo xxx@xx;xxx@xxx;....; -> indirizzi email del cliente
    -- La variabile var_phones contiene un elenco del tipo 0000000000;0000000000;....; -> numeri fissi del cliente

    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
```

```
    end;

    set autocommit=0;
    set transaction isolation level read uncommitted;

    start transaction;
    INSERT INTO cliente(tessera, codice_fiscale, nome, cognome, data_nascita, indirizzo_via, indirizzo_cap, indirizzo_civico) VALUES (NULL, var_cf, var_name, var_surname, var_birth_date, var_address, var_cap, var_number);
    set var_card = last_insert_id();
    call _inserisciRecapitiCliente(var_card, var_mobile_phones, var_emails, var_phones, var_affected_rows);
    if var_affected_rows <= 0 then
        signal sqlstate '45000' set message_text = 'Il cliente deve avere almeno un recapito';
    end if;
    commit;
    set autocommit=1;
end
```

Procedura inserisciRecapitiCliente (operazione C2):

La procedura realizza l'inserimento di ulteriori recapiti ad un cliente registrato.

Il livello di isolamento utilizzato è read uncommitted poiché sono presenti solo scritture senza letture.

```
CREATE PROCEDURE `inserisciRecapitiCliente` (in var_client INT, in var_mobile_phones TEXT, in var_emails TEXT, in var_phones TEXT)
begin

    -- La variabile var_mobile_phones contiene un elenco del tipo 0000000000;0000000000;....; -> numeri di cellulare del cliente
    -- La variabile var_emails contiene un elenco del tipo xxx@xx;xxx@xxx;....; -> indirizzi email del cliente
    -- La variabile var_phones contiene un elenco del tipo 0000000000;0000000000;....; -> numeri fissi del cliente

    declare var_affected_rows INT;

    declare exit handler for sqlexception
```

```
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;

    set autocommit=0;
    set transaction isolation level read uncommitted;

    start transaction;
    call     _inserisciRecapitiCliente(var_client,     var_mobile_phones,     var_emails,
var_phones, var_affected_rows);
    commit;
    set autocommit=1;
end
```

Procedura registraNoleggio (operazione C3):

La procedura inserisce semplicemente il nuovo noleggio. In caso in cui la copia sia attualmente noleggiata il trigger manderà in abort l'operazione sollevando un'eccezione.

```
CREATE PROCEDURE `registraNoleggio` (in var_copy INT, in var_client INT, in
var_expiration_date DATE)
begin
    INSERT INTO noleggio(copia_film, cliente, scadenza) VALUES (var_copy, var_client,
var_expiration_date);
end
```

Procedura registraRestituzione (operazione C4):

La procedura registra semplicemente la restituzione. In caso la restituzione non si associata effettivamente a nessun noleggio la procedura invierà un messaggio di errore.

```
CREATE PROCEDURE `registraRestituzione` (in var_copy INT, in var_client INT)
begin
    declare var_affected_rows TINYINT;
    UPDATE noleggio_corrente SET restituzione = CURDATE() WHERE copia_film = var_copy
AND cliente = var_client;
    SELECT ROW_COUNT() INTO var_affected_rows;
    if var_affected_rows <= 0 then
```

```
        signal sqlstate '45000' set message_text = 'Nessun noleggio collegato';
    end if;
end
```

Procedura listaNoleggiScaduti (operazione C5):

La procedura utilizza la vista dei noleggi correnti per estrarre i noleggi scaduti e li restituisce al chiamante.

```
CREATE PROCEDURE `listaNoleggiScaduti` ()
begin
    SELECT tessera, codice_fiscale, nome, cognome,
           codice, titolo, regista, tipo,
           noleggio, scadenza, restituzione
    FROM noleggio_corrente
        JOIN copia_film ON copia_film = codice
        JOIN cliente ON cliente = tessera
    WHERE scadenza < CURDATE()
    ORDER BY noleggio DESC, scadenza DESC, restituzione;
end
```

Vengono riportare ora le ulteriori procedure realizzate per poter gestire i dati presenti nel sistema.

Procedura inserisciCopiaFilm:

```
CREATE PROCEDURE `inserisciCopiaFilm` (in var_code INT, in var_title VARCHAR(100), in var_director VARCHAR(50), in var_type ENUM('DVD', 'Videocassetta'), in var_sector INT, in var_position VARCHAR(100))
begin
    INSERT INTO copia_film(codice, titolo, regista, tipo, settore, posizione) VALUES(var_code, var_title, var_director, var_type, var_sector, var_position);
end
```

Procedura inserisciFilm:

La procedura permette di inserire un nuovo film nel catalogo.

Tra i parametri di input la procedura accetta una lista di nomi e una lista di cognomi degli attori principali del cast. Poiché si vuole realizzare un inserimento atomico, gli inserimenti vengono fatti all'interno di una transazione.

Per estrarre i nomi e i cognomi dalla lista viene utilizzata la solita funzione *tokenize_string_list*.

Poiché in essa compaiono solo operazioni di scrittura, si è scelto di utilizzare come livello di isolamento *read uncommitted*.

```
CREATE PROCEDURE `inserisciFilm` (in var_title VARCHAR(100), in var_director VARCHAR(50), in var_year YEAR, in var_price_dvd FLOAT, in var_price_videotape FLOAT, in var_original_title VARCHAR(100), in var_original_director VARCHAR(50),
                                  in var_names TEXT, in var_surnames TEXT)
begin

    -- La variabile var_names contiene un elenco del tipo xxx;xxx;....; -> nomi degli attori
    -- La variabile var_surnames contiene un elenco del tipo xxx;xxx;....; -> cognomi degli attori

    declare var_actor_name VARCHAR(50);
    declare var_actor_surname VARCHAR(50);
    declare var_counter INT;

    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;

    set autocommit=0;
    set transaction isolation level read uncommitted;

    start transaction;
    INSERT INTO film(titolo, regista, anno, costo_dvd, costo_videocassetta, originale_titolo, originale_regista) VALUES (var_title, var_director, var_year, var_price_dvd, var_price_videotape, var_original_title, var_original_director);

    set var_counter = 1;
    insert_loop: loop

        set var_actor_name = tokenize_string_list(var_names, ';', var_counter);
        set var_actor_surname = tokenize_string_list(var_surnames, ';', var_counter);
        if var_actor_name = '' or var_actor_surname = '' then
```

```
            leave insert_loop;
        end if;

        INSERT INTO cast_film(attore_nome, attore_cognome, film_titolo, film_regista)
VALUES(var_actor_name, var_actor_surname, var_title, var_director);

        set var_counter = var_counter + 1;

    end loop;

    commit;
    set autocommit=1;
end
```

Procedura inserisciSettore:

```
CREATE PROCEDURE `inserisciSettore` (in var_code INT)
begin
    INSERT INTO settore(codice) VALUES(var_code);
end
```

Procedura listaCariche:

```
CREATE PROCEDURE `listaCariche` (in var_employee CHAR(16))
begin
    SELECT inizio, COALESCE(fine, 'corrente') as fine, mansione
    FROM carica
    WHERE impiegato = var_employee
    ORDER BY inizio DESC, fine DESC;
end
```

Procedura listaClienti:

La procedura ritorna al chiamante la lista dei clienti registrati al sistema con i relativi recapiti.

In particolare, il primo result set contiene la lista dei clienti presenti all'interno del sistema e i successivi result set contengono, per ogni cliente trovato, i suoi numeri di cellulare, le sue email e i suoi numeri di telefono.

Poiché le letture devono essere eseguite in maniera isolata rispetto alle altre transazioni, vengono gestite da una transazione più complessa che fa uso di un cursore per eseguire la ricerca dei recapiti di un singolo cliente per volta.

Il livello di isolamento scelto è *serializable* poiché devono essere evitati gli inserimenti fantasma tra la lettura di tutti i clienti all'interno dallo statement di select e la lettura dei clienti da parte del cursore.

```
CREATE PROCEDURE `listaClienti` ()
begin

    declare done bool default false;
    declare var_client INT;

    declare cur cursor for SELECT tessera FROM cliente ORDER BY tessera;
    declare continue handler for not found set done = true;

    set transaction isolation level serializable;

    start transaction read only;
        open cur;
        SELECT tessera, codice_fiscale, nome, cognome, data_nascita, indirizzo_via,
indirizzo_cap, indirizzo_civico FROM cliente ORDER BY tessera;

        read_loop: loop
            fetch cur into var_client;
            SELECT cellulare FROM cellulare WHERE cliente = var_client;
            SELECT email FROM email WHERE cliente = var_client;
            SELECT telefono FROM telefono WHERE cliente = var_client;
            if done then
                leave read_loop;
            end if;
        end loop;

    commit;
end
```

Procedura listaCopieFilm:

```
CREATE PROCEDURE `listaCopieFilm` ()
begin
    SELECT codice, titolo, regista, tipo, settore, posizione
    FROM copia_film;
end
```

Procedura listaFilm:

La procedura ritorna la lista dei film presenti nel catalogo del sistema. Come per i clienti, viene restituito, nel primo result set, la lista di ogni film e nei successivi result set, per ogni film del primo result set, gli attori principali del film. Per le stesse motivazioni elencate precendemente, per la procedura *listaClienti* si è scelto anche in questo caso il livello di isolamento *serializable*.

```
CREATE PROCEDURE `listaFilm` ()
begin

    declare done bool default false;
    declare var_title VARCHAR(100);
    declare var_director VARCHAR(50);

    declare cur cursor for SELECT titolo, regista FROM film ORDER BY titolo, regista;
    declare continue handler for not found set done = true;

    set transaction isolation level serializable;

    start transaction read only;
        open cur;
        SELECT titolo, regista, anno, costo_dvd, costo_videocassetta, originale_titolo,
originale_regista FROM film ORDER BY titolo, regista;
        read_loop: loop

            fetch cur into var_title, var_director;

            SELECT attore_nome as nome, attore_cognome as cognome
            FROM cast_film
            WHERE film_titolo = var_title AND film_regista = var_director;

            if done then
                leave read_loop;
            end if;
        end loop;
    commit;
end
```

Procedura listaImpiegati:

```
CREATE PROCEDURE `listaImpiegati` ()
begin
    SELECT codice_fiscale, nome, recapito, username, mansione
    FROM impiegato
        JOIN login ON login.impiegato = codice_fiscale
        JOIN carica_attuale ON codice_fiscale = carica_attuale.impiegato
    ORDER BY codice_fiscale;
end
```

Procedura listaNoleggiCliente:

```
CREATE PROCEDURE `listaNoleggiCliente` (in var_client INT)
begin
    SELECT tessera, codice_fiscale, nome, cognome,
           codice, titolo, regista, tipo,
           noleggio, scadenza, restituzione
    FROM noleggio
        JOIN copia_film ON copia_film = codice
        JOIN cliente ON cliente = tessera
    WHERE cliente = var_client
    ORDER BY noleggio DESC, scadenza DESC, restituzione;
end
```

Procedura listaNoleggiCopiaFilm:

```
CREATE PROCEDURE `listaNoleggiCopiaFilm` (in var_code INT)
begin
    SELECT tessera, codice_fiscale, nome, cognome,
           codice, titolo, regista, tipo,
           noleggio, scadenza, restituzione
    FROM noleggio
        JOIN copia_film ON copia_film = codice
        JOIN cliente ON cliente = tessera
    WHERE copia_film = var_code
    ORDER BY noleggio DESC, scadenza DESC, restituzione;
end
```

Procedura listaNoleggiCorrenti:

```
CREATE PROCEDURE `listaNoleggiCorrenti` ()
begin
    SELECT tessera, codice_fiscale, nome, cognome,
```

```
            codice, titolo, regista, tipo,
            noleggio, scadenza, restituzione
    FROM noleggio_corrente
        JOIN copia_film ON copia_film = codice
        JOIN cliente ON cliente = tessera
    ORDER BY noleggio DESC, scadenza DESC, restituzione;
end
```

Procedura listaNoleggiFilm:

```
CREATE PROCEDURE `listaNoleggiFilm` (in var_title VARCHAR(100), in var_director VARCHAR(50))
begin
    SELECT tessera, codice_fiscale, nome, cognome,
           codice, titolo, regista, tipo,
           noleggio, scadenza, restituzione
    FROM noleggio
        JOIN copia_film ON copia_film = codice
        JOIN cliente ON cliente = tessera
    WHERE titolo = var_title AND regista = var_director
    ORDER BY noleggio DESC, scadenza DESC, restituzione;
end
```

Procedura listaSettori:

```
CREATE PROCEDURE `listaSettori` ()
begin
    SELECT codice
    FROM settore;
end
```

Procedura listaTurni:

```
CREATE PROCEDURE `listaTurni` (in var_employee CHAR(16), in var_month TINYINT(2), in var_year YEAR)
begin
    SELECT extract(day from data) as giorno, ora_inizio, ora_fine
    FROM turno_lavoro
    WHERE extract(month from data) = var_month AND extract(year from data) = var_year
    AND impiegato = var_employee
    ORDER BY data;
end
```

Procedura login:

La procedura permette il login al sistema.

Le credenziali dell'utente vengono confrontate con gli utenti presenti all'interno della tabella *login*. Se non c'è alcun riscontro viene sollevata un'eccezione e notificato all'utente che le credenziali sono errate.

Se invece viene trovato il relativo account per accedere al sistema, si vanno a impostare le varabili in uscita per ritornare al chiamante il ruolo dell'utente ed eventualmente a quale impiegato fa riferimento l'account.

```
CREATE PROCEDURE `login` (in var_username VARCHAR(100), in var_password VARCHAR(100), out var_role ENUM('Proprietario', 'Cassiere', 'Commesso'), out var_employee CHAR(16))
begin
    declare var_check_username VARCHAR(100);

    SELECT username, login.impiegato, mansione
    FROM login
        LEFT JOIN carica_attuale ON login.impiegato = carica_attuale.impiegato
    WHERE username = var_username AND password = MD5(var_password) INTO var_check_username, var_employee, var_role;

       if var_check_username is null then
        signal sqlstate '45000' set message_text = 'Credenziali errate';
    end if;

    if var_employee is null then
              set var_role = 'Proprietario';
    end if;
end
```

Si noti che le procedure di rimozione dei dati sono state realizzate tenendo conto che le chiavi esterne sulle entità sono state istanziate utilizzando le impostazioni di *cascade* o di *set null* su *on update* e *on delete*.

Procedura rimuoviCliente:

```
CREATE PROCEDURE `rimuoviCliente` (in var_card INT)
begin
```

```
    DELETE FROM cliente WHERE tessera = var_card;
end
```

Procedura rimuoviCopiaFilm:

```
CREATE PROCEDURE `rimuoviCopiaFilm` (in var_code INT)
begin
    DELETE FROM copia_film WHERE codice = var_code;
end
```

Procedura rimuoviFilm:

```
CREATE PROCEDURE `rimuoviFilm` (in var_title VARCHAR(100), in var_director VARCHAR(100))
begin
    DELETE FROM film WHERE titolo = var_title AND regista = var_director;
end
```

Procedura rimuoviImpiegato:

```
CREATE PROCEDURE `rimuoviImpiegato` (in var_cf CHAR(16))
begin
    DELETE FROM impiegato WHERE codice_fiscale = var_cf;
end
```

Procedura rimuoviRecapitiCliente:

La procedura riprende la stessa struttura della procedura di inserimento ma il livello di isolamento utilizzato è repeatable read poiché si vogliono evitare anomalie tali per cui durante il controllo del numero totale di recapiti di un cliente uno di questi venga cancellato.

A differenza di questa però, al termine della rimozione viene controllato se il dipendente possiede ancora un recapito valido, in caso contrario viene sollevata un'eccezione e la transazione viene mandata in abort.

```
CREATE PROCEDURE `rimuoviRecapitiCliente` (in var_client INT, in var_mobile_phones TEXT,
in var_emails TEXT, in var_phones TEXT)
begin

    -- La variabile var_mobile_phones contiene un elenco del tipo
0000000000;0000000000;....; -> numeri di cellulare del cliente
    -- La variabile var_emails contiene un elenco del tipo xxx@xx;xxx@xxx;....; ->
indirizzi email del cliente
```

```
    -- La variabile var_phones contiene un elenco del tipo 0000000000;0000000000;....; -
> numeri fissi del cliente

    declare var_counter INT;
    declare var_aux TEXT;

    declare var_count_mobile_phones INT;
    declare var_count_emails INT;
    declare var_count_phones INT;

    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;

    set autocommit=0;
    set transaction isolation level repeatable read;

    start transaction;

    -- CELLULARE

    set var_counter = 1;
    remove_mobile_phones_loop: loop
        set var_aux = tokenize_string_list(var_mobile_phones, ';', var_counter);
        if var_aux = ''
        then
            leave remove_mobile_phones_loop;
        end if;
        DELETE FROM cellulare WHERE cellulare = var_aux AND cliente = var_client;
        set var_counter = var_counter+1;
    end loop;

    -- EMAIL

    set var_counter = 1;
    remove_emails_loop: loop
        set var_aux = tokenize_string_list(var_emails, ';', var_counter);
```

```
        if var_aux = ''
        then
            leave remove_emails_loop;
        end if;
        DELETE FROM email WHERE email = var_aux AND cliente = var_client;
        set var_counter = var_counter+1;
    end loop;

    -- TELEFONO

    set var_counter = 1;
    remove_phones_loop: loop
        set var_aux = tokenize_string_list(var_phones, ';', var_counter);
        if var_aux = ''
        then
            leave remove_phones_loop;
        end if;
        DELETE FROM telefono WHERE telefono = var_aux AND cliente = var_client;
        set var_counter = var_counter+1;
    end loop;

    SELECT count(*) FROM cellulare WHERE cliente = var_client INTO
var_count_mobile_phones;
    SELECT count(*) FROM email WHERE cliente = var_client INTO var_count_emails;
    SELECT count(*) FROM telefono WHERE cliente = var_client INTO var_count_phones;

    if var_count_mobile_phones <= 0 and var_count_emails <= 0 and var_count_phones <= 0
then
        signal sqlstate '45000' set message_text = 'Il cliente deve avere almeno un
recapito';
    end if;

    commit;
    set autocommit=1;
end
```

Procedura rimuoviTurnoLavoro:

La procedura permette di rimuovere un turno di lavoro quando questo non è stato ancora iniziato o terminato. Prima di essere eliminato, viene eseguito il controllo su tale vincolo e in caso non sia stato rispettato la transazione viene mandata in abort.
La scelta del livello di isolamento come *repeatable read* è data dal fatto che si vogliono evitare anomalie di tipo *aggiornamento fantasma* dovute a transazioni concorrenti che cercano di aggiornare i turni di lavoro inserendo gli orari di *ingresso* o *uscita*.

```
CREATE PROCEDURE `rimuoviTurnoLavoro` (in var_employee CHAR(16), in var_month TINYINT(2), in var_year YEAR)
begin

    declare var_check TINYINT; -- Mantiene il controllo per capire se il turno di quel mese è stato già iniziato o meno

    declare exit handler for sqlexception
        begin
            rollback;
            set autocommit=1;
            resignal;
        end;

    set autocommit=0;
    set transaction isolation level repeatable read;

    start transaction;

    if CAST(CONCAT(var_year, '-', LPAD(var_month, 2, '0'), '-01') as date) < CAST(CONCAT(extract(year from CURDATE()), '-', LPAD(extract(month from CURDATE()), 2, '0'), '-01') as date) then
        signal sqlstate '45000' set message_text = 'Il mese non può essere minore di quello odierno';
    end if;

    SELECT count(*) FROM turno_iniziato WHERE extract(month from data) = var_month AND extract(year from data) = var_year AND impiegato = var_employee INTO var_check;
    if var_check > 0 then
        signal sqlstate '45000' set message_text = 'Turni di lavoro già iniziati';
    end if;
```

```
    SELECT count(*) FROM turno_terminato WHERE extract(month from data) = var_month AND
extract(year from data) = var_year AND impiegato = var_employee INTO var_check;
    if var_check > 0 then
        signal sqlstate '45000' set message_text = 'Turni di lavoro già terminati';
    end if;

    DELETE FROM turno_lavoro WHERE extract(month from data) = var_month AND extract(year
from data) = var_year AND impiegato = var_employee;

    commit;
    set autocommit=1;
end
```