Sententia

Progetto basi di dati

A.A. 2021/2022

Leonardo Temperanza

Amanjot Singh

Alessandro Dodi

(Gruppo n.28)

**Indice dei contenuti**

**Requisiti di progetto e descrizione dell’ambito applicativo**

Si intende realizzare un database a supporto di un social network. Per definire il contesto su cui si va a realizzare la base di dati, tale social network è chiamato “Sententia”, che significa opinione dal latino, il cui scopo è quello di dare la possibilità a chiunque di esprimere la propria opinione su qualunque argomento essi desiderino dandone una recensione. L’idea principale che lo rende unico a tutti gli effetti è quello di non imporre limiti di alcun tipo e organizzare le recensioni in modo tale che siano trovabili e consultabili efficientemente.

Il fulcro del sistema si colloca indubbiamente nell’utente, in quanto essenzialmente tutto ruota attorno alla figura dello stesso, a ciò che decide di pubblicare, ai messaggi che invia e riceve, e le sue transazioni. I loro dati sono: Email e Username che sono entrambi univoci, nome, cognome, password, e foto profilo. Una prima funzionalità riguarda la richiesta di amicizia tra utenti. L’amicizia non è necessariamente reciproca, ovvero un utente A può avere B come amico ma non necessariamente è vero per B rispetto ad A. Un’amicizia può essere annullata, ma non è necessario tenere traccia dell’intera catena di amicizie/annullamenti e neanche la data in cui è stata effettuata l’annullazione.

Le recensioni sono rappresentate nel seguente modo: esse riguardano un oggetto, che a sua volta appartiene ad una categoria, e contengono informazioni quali un codice identificativo, una foto, un titolo e un valore. Esistono due tipologie di interazioni tra utenti: quelle pubbliche, che consistono nel commentare una determinata recensione, e quelle private, gestite da un sistema di messaggistica che consente l’invio e ricezione di messaggi testuali, immagini e persino altre recensioni. I commenti riguardanti una determinata recensione sono caratterizzati da un codice identificativo in base alla recensione (per facilità di visualizzazione e ricerca dei vari commenti) e dal testo dello stesso. Si richiede inoltre la possibilità di rispondere a commenti. Un’altra forma di espressione della propria opinione è sotto forma della medaglia, che un utente può conferire (solamente una volta) ad una determinata recensione. Per quanto riguarda invece i messaggi fra utenti, in generale si necessita di un codice identificativo in base alla coppia di utenti coinvolti, inoltre occorre conoscere se siano stati letti o meno, e nel caso in cui essi siano testuali si richiede solamente il testo del messaggio stesso, in caso contrario contengono una descrizione opzionale, affiancata da un’immagine oppure un collegamento ad una recensione.

Le interazioni tra utenti necessitano di un controllo per preservare la sicurezza della piattaforma, occorre dunque introdurre una figura il cui compito è quello di scorrere la lista dei contenuti della stessa ed eventualmente rimuoverli se si reputa che violino le regolazioni imposte. Tale figura prende il nome di moderatore, e i dati che vengono memorizzati sono gli stessi dell’utente “utilizzatore” a meno della foto profilo e dell’indirizzo IP. Un moderatore può fare varie cose: può effettuare un “ban” su uno o più utenti che possono essere annullati successivamente se necessario, e può “nascondere” una recensione temporaneamente. Non rappresenta un problema il fatto che un moderatore e un utente abbiano lo stesso username o la stessa email, perché può capitare in certi casi rari che un moderatore sia anche un utente che fa utilizzo dei servizi erogati.

Le recensioni vengono considerate con più cautela, dato che può essere una fonte di monetizzazione e può richiedere svariate ore di lavoro, mentre per i commenti ci si aspetta che non siano particolarmente importanti, motivo per il quale la “cancellazione” di una recensione è in tutti i casi soltanto temporanea, dando la possibilità al creatore di contestarla con una motivazione valida. Invece i commenti possono essere rimossi direttamente dalla base di dati, e non è necessario neanche memorizzarsi la data o il moderatore che l’ha fatto. In base al numero di violazioni che ha effettuato un determinato utente, un moderatore può decidere di “bandire” un utente, e tale operazione deve essere memorizzata assieme al moderatore e all’utente coinvolti. È necessario che le informazioni che riguardano un utente bandito restino nel database, compreso il suo indirizzo IP, al fine di individuazione dello stesso nel caso tentasse di creare un nuovo account.

Essendo il progetto una base di dati di supporto ad un social network, è importante monitorare il tempo esatto di ciascuna pubblicazione. Questo include i messaggi, le medaglie, i commenti e le recensioni, ma anche il ban di una recensione, le varie transazioni, e le amicizie tra utenti. Questo processo di controllo dei contenuti viene affidato in parte anche agli utenti utilizzatori, in quanto si permette a questi ultimi di segnalare una o più recensioni. La segnalazione può avere una tipologia tra quelle predefinite, oppure l’utente può scrivere liberamente la sua motivazione se questa non è presente tra quelle selezionabili. Una segnalazione da parte di un utente riferita ad una particolare recensione può essere effettuata soltanto una volta.

Un altro aspetto molto importante è il discorso sulla monetizzazione. Vi sono due tipologie di utenti: gli utenti normali e gli utenti premium. Gli utenti normali sono quelli descritti fino ad ora, mentre quelli premium hanno tutte le caratteristiche degli utenti normali con l’aggiunta del fatto che possono ricevere donazioni monetarie da utenti (sia da utenti normali sia premium).

Una peculiarità degli utenti premium è che hanno la possibilità di definire piani di abbonamento, ciascuno con un determinato periodo e quantità di pagamento (per esempio: 2€ al mese), ai quali una molteplicità di utenti possono iscriversi. Dell’iscrizione occorre memorizzare la data di effettuazione e l’eventuale data di abbandono. Inoltre, gli utenti premium possono anche impostare incentivi per l’iscrizione a determinati piani per mezzo dell’esclusività temporanea di una recensione. Per ciascun piano e ciascuna recensione pubblicata dallo stesso utente si può avere un’esclusività definita come una data a partire dalla quale la visione esclusiva è permessa (solo per gli iscritti a quel piano). Una stessa recensione può fare parte di più esclusività diverse (soltanto per piani diversi, non all’interno dello stesso piano) ciascuna con una data di visualizzazione anticipata differente. Allo stesso tempo un piano può promettere l’esclusività di più recensioni con date diverse. Vi è inoltre una proprietà della recensione che ne indica la data in cui è visibile a tutti gli utenti.

Per questo progetto occorre tenere traccia di tutte le transazioni effettuate dagli utenti, in modo tale che essi stessi possano, sotto determinate condizioni, annullarla, e occorre anche tenere traccia del fatto che una transazione sia già stata annullata o meno. Le transazioni possono essere di due tipologie: le transazioni manuali e quelle automatiche. Quelle manuali possono essere effettuate liberamente da un utente qualsiasi verso un utente premium, con una quantità di denaro a libera scelta, mentre quelle automatiche sono quelle effettuate per effetto dell’iscrizione ad un piano. Esse sono identificate da un TRN, un codice utilizzato universalmente per identificare una transazione, e devono contenere il numero della carta di credito dell’utente donatore e l’identificativo dell’utente premium destinatario. Un utente può avere più carte di credito (anche nessuna) e una carta di credito può essere utilizzata da più utenti. È richiesto di imporre il vincolo che in una certa data, un determinato utente può soltanto effettuare una donazione rivolta ad un determinato utente premium, e questo va garantito sia per le transazioni manuali sia per quelle automatiche, con l’unica differenza che per quanto riguarda quelle automatiche si permette che un utente ne effettui più di una nei confronti di uno stesso utente premium, ma soltanto in piani diversi. In altre parole, si impedisce ad un utente di effettuare più di una transazione automatica riferita ad un unico piano.

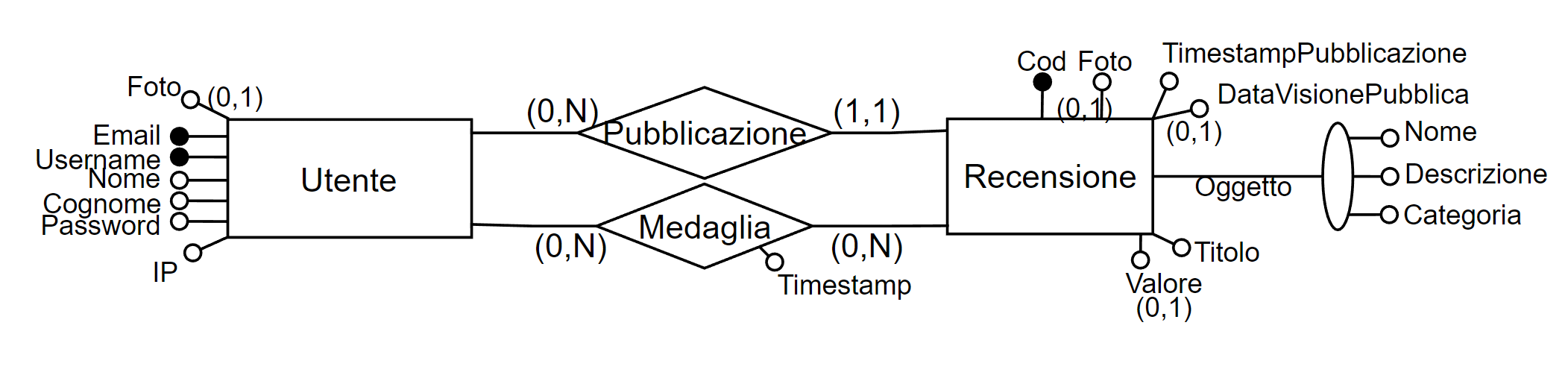
Senza addentrarsi troppo nel dettaglio, in quanto si andrebbe fuori dallo scopo del progetto in questione, i pagamenti vengono svolti nella modalità seguente. L’azienda sviluppatrice del social network gestisce entrate e uscite (transazioni dall’utente donatore all’azienda e quelle dall’azienda all’utente destinatario), in modo tale che essa possa incassare una certa percentuale. Per le entrate si utilizza la carta di credito, mentre per quanto riguarda le uscite vengono gestite con un bonifico tramite IBAN. L’IBAN è un’informazione necessaria solo per gli utenti premium, mentre non la si memorizza per gli utenti normali, ed entrambi possono fornire zero o più carte di credito (e ne devono avere almeno una se hanno intenzione di effettuare transazioni).

Essendo il progetto in questione un social network, esso necessita di continui cambiamenti e aggiunte, non solo per scopi di manutenzione ma anche per il fine di ritenere l’attenzione del pubblico, quindi è richiesta la possibilità di poter agilmente aggiungere funzionalità.

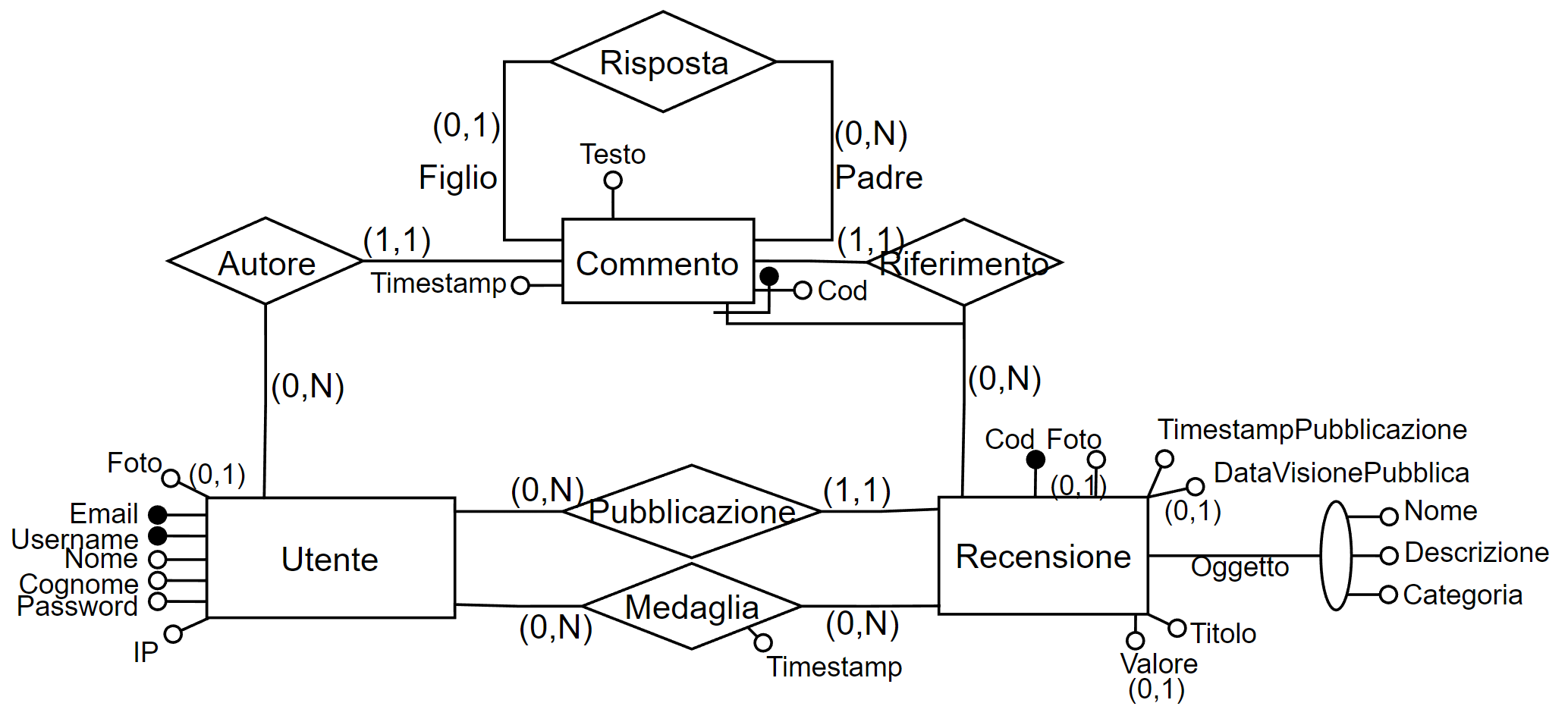
Il social network deve poter essere robusto ad eventuali sistemi di sovraccarico delle risorse, per esempio l’invio automatizzato di un numero elevato di commenti o recensioni (anche chiamato spam). Questo ha anche lo scopo di migliorare l’esperienza utente. Si richiede che un utente possa effettuare al massimo 1,000 pubblicazioni (recensioni o commenti) in un’ora.

**Progetto concettuale – Schema E/R**

I requisiti del progetto rilevano come fulcro il legame tra gli utenti e le recensioni, che viene rappresentata nel seguente schema E/R (estremamente semplificato e privo della maggior parte delle funzionalità richieste):



È possibile ampliare questo schema con l’obiettivo di implementare ciascuna richiesta. Innanzitutto, i commenti possono rispondere ad altri commenti, viene dunque introdotta un’auto-associazione uno a molti. Non viene gestito, in questa prima fase di progettazione concettuale, il problema che un commento può essere in risposta ad un altro commento pur non riferendosi alla stessa recensione. Questo verrà in effetti gestito nella fase appena successiva, ovvero di progettazione logica.

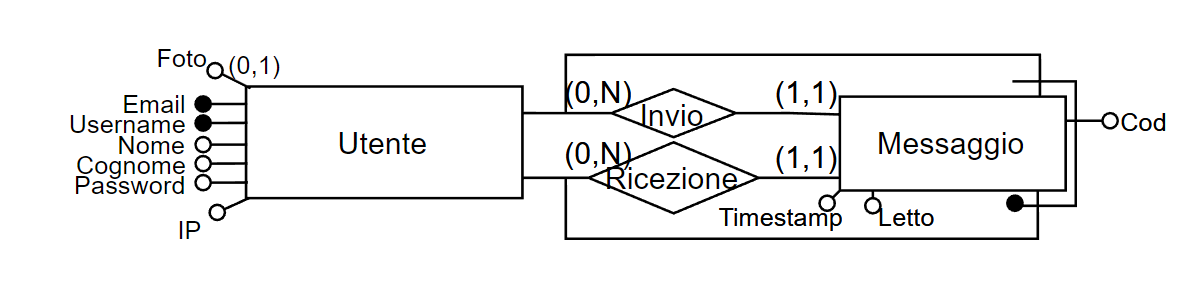
Si applica una strategia di progettazione bottom-up, dove si sviluppano più sottoprogetti separati per poi metterli insieme per ottenere lo schema finale. Il vantaggio sta nel fatto che è più facile da gestire una visione ristretta rispetto ad una globale, mentre il lato negativo è la difficoltà di integrazione, ovvero il fatto che mettere insieme più parti eterogenee può rivelarsi essere difficoltoso. Si ritiene la scelta adatta per questo progetto in quanto è naturalmente scomponibile in più parti che sono relativamente indipendenti tra di loro.

Per ragionare sul problema risulta quindi efficace scomporlo in più componenti:

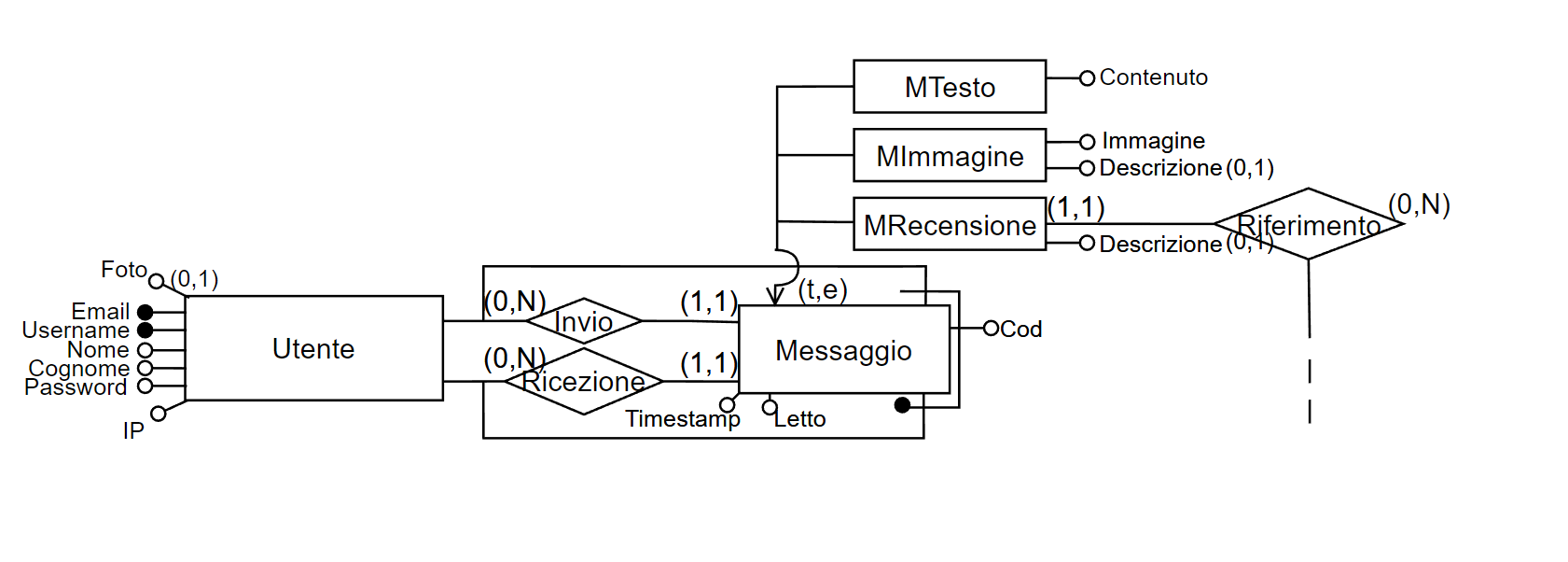
1. La messaggistica e le amicizie tra utenti
2. I moderatori e il controllo dei contenuti
3. Gli utenti premium e le transazioni

Si può notare che queste tre componenti sono strettamente correlate agli utenti e alle recensioni. Per la natura estremamente dinamica della realtà che si va a rappresentare, si pone come obiettivo la semplicità del diagramma ER, lasciando la gestione dei vincoli più complessi ai trigger, discussi in una parte successiva di questo progetto.

La messaggistica tra utenti è gestita attraverso un’entità Messaggio legata all’entità Utente in due relazioni uno a molti (dove il lato molti è il Messaggio).

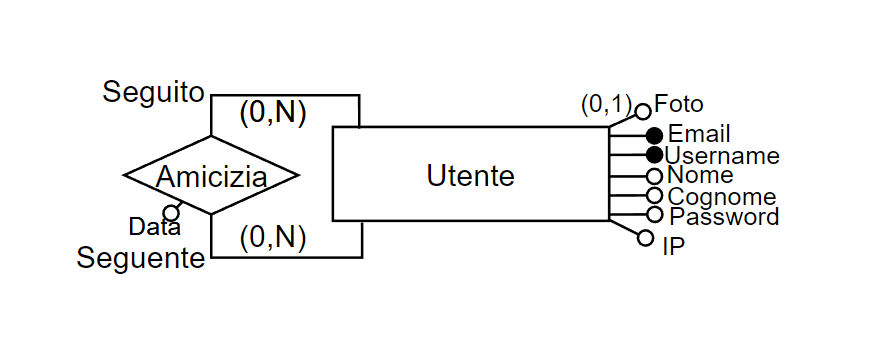


Il messaggio contiene un codice che identifica lo stesso a partire dalla coppia di utenti “mittente-destinatario”, e le tipologie differenti vengono modellate con l’uso di una gerarchia, come si può vedere dalla seguente figura.

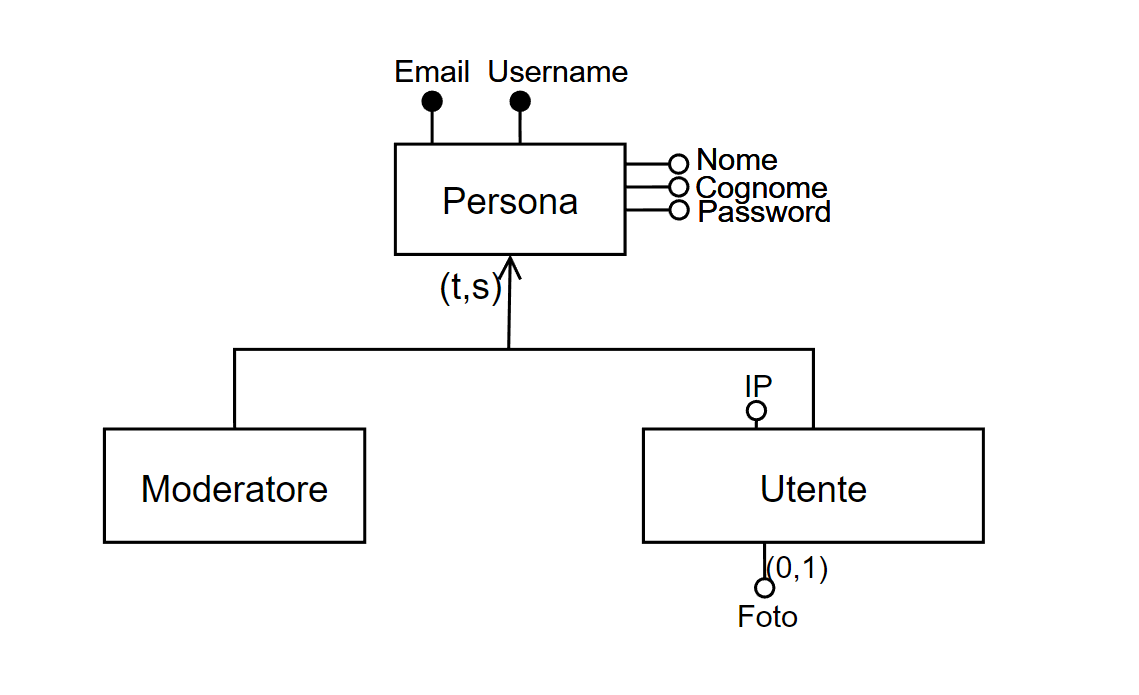


Si tratta di una gerarchia totale ed esclusiva perché un messaggio può essere solamente uno e uno solo di quei 3 tipi. Ci si rende conto del fatto che la descrizione opzionale è comune alle due entità MImmagine e MRecensione, si potrebbe dunque riscrivere come un’altra gerarchia che ha come entità padre “MNonTesto” o similmente denominata, tuttavia per questioni di semplicità di rappresentazione si sceglie di evitare di modificare il diagramma in questo modo. In ogni caso le decisioni di traduzione in uno schema logico verranno intraprese nella fase successiva.

Le amicizie vengono modellate attraverso un’auto-associazione molti a molti, dato che un utente può richiedere l’amicizia a più utenti diversi.



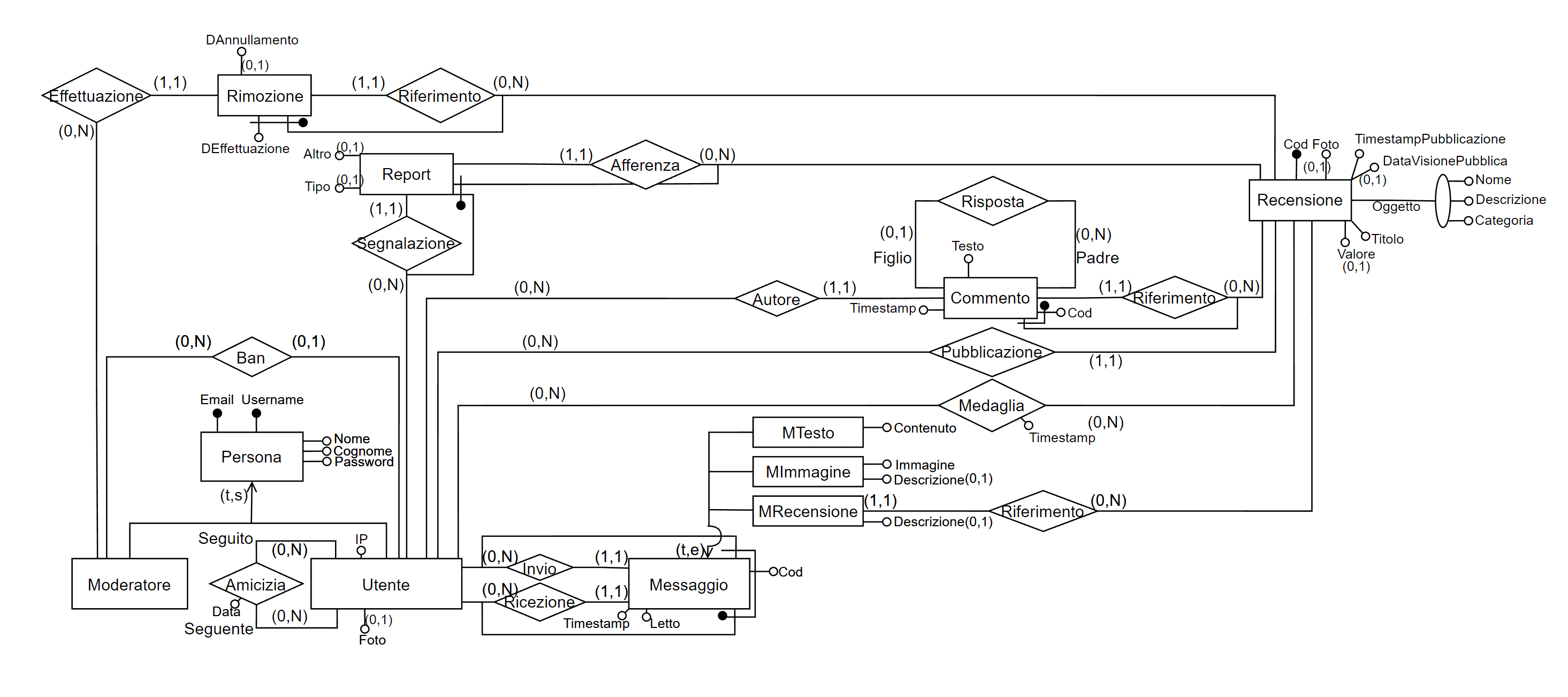
I moderatori e gli utenti normali condividono una grande porzione di dati, è possibile introdurre in questo caso una gerarchia totale e sovrapposta, dove l’entità padre viene chiamata “Persona”. è totale perché non ci si interessa delle persone che non sono né utenti né moderatori, ed è sovrapposta perché, come indicato dal testo, ci possono essere moderatori che sono al contempo utenti.



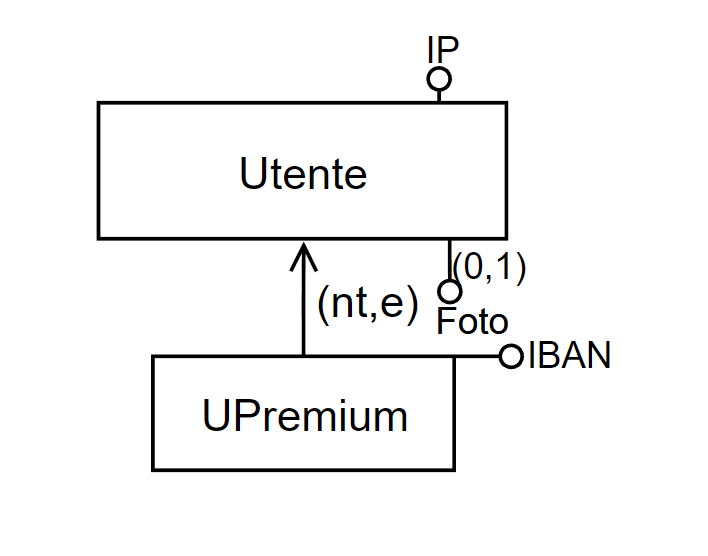
Un moderatore può bandire un determinato utente, e questo si traduce come una semplice associazione uno a molti.

Il moderatore può “disattivare”, ovvero rendere invisibile temporaneamente, le recensioni. Per progettare questo, si può ricorrere all’uso di un’entità “rimozione” che ha come chiave composta la data di effettuazione e il riferimento alla recensione, e contiene un’eventuale data di annullamento. In questo modo una rimozione può essere effettuata più volte in date diverse, ma questo deve essere permesso soltanto se vi sia una data di annullamento non nulla (che indica che sia effettivamente stata annullata), un vincolo che verrà gestito nelle fasi successive. Un utente può effettuare un report, il quale è identificato dall’utente e dalla recensione che sono coinvolti. In questo modo una recensione può essere disattivata e riattivata più volte.

Di seguito è riportato il diagramma E/R aggiornato.



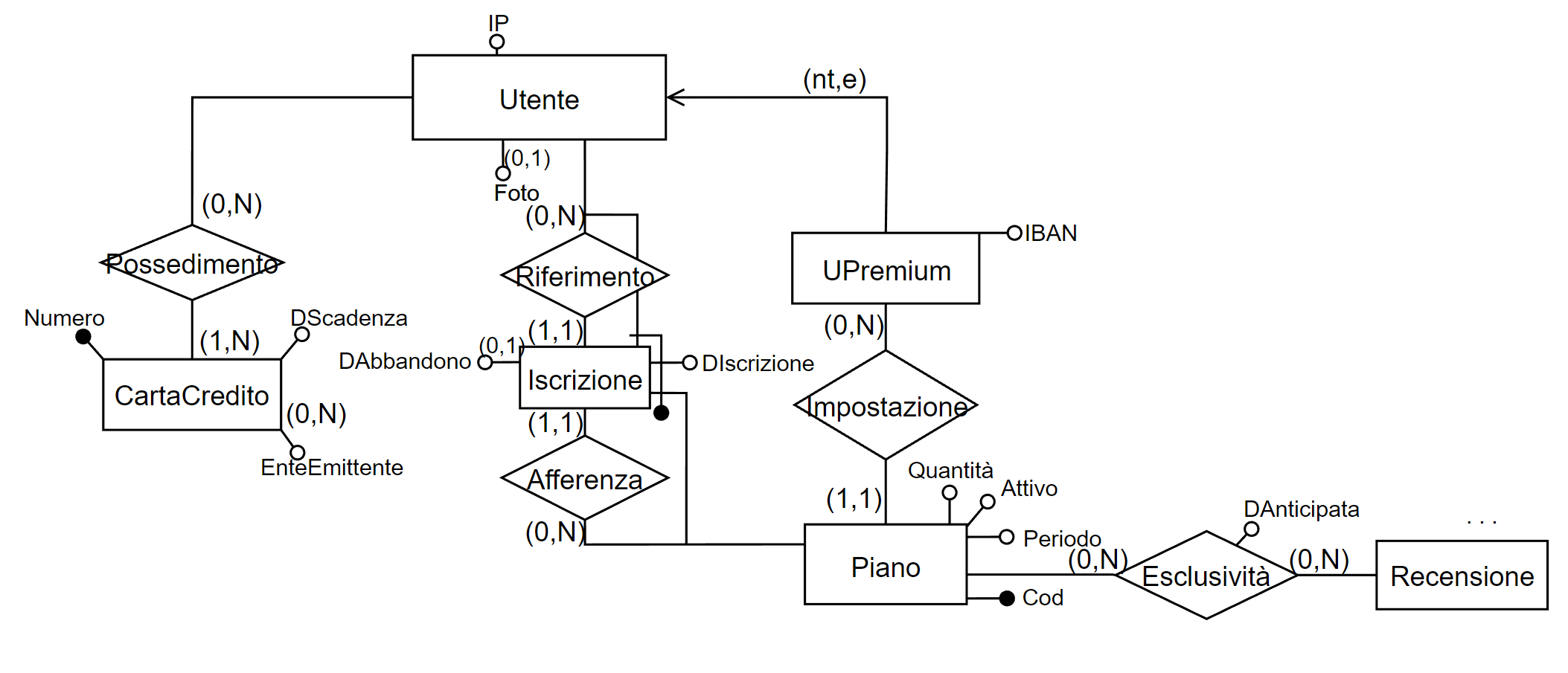
Per quanto riguarda il sistema di monetizzazione di questo progetto, innanzitutto si può notare che gli utenti premium non sono altro che utente con alcune caratteristiche aggiuntive, in altre parole una specializzazione di utente. Questo si rappresenta all’interno dello schema ER sotto forma di una gerarchia non totale ed esclusiva come segue.



La gerarchia è parziale perché un utente può non essere premium (gli utenti normali), e una discussione sull’esclusività in una gerarchia con una sola entità figlia è per certi aspetti irrilevante.

Il piano è in un’associazione uno a molti con l’entità UPremium, dove la partecipazione di UPremium è facoltativa, e l’esclusività di una recensione viene gestita con una semplice associazione molti a molti tra Piano e Recensione contenente un attributo DAnticipata, ovvero la data oltre la quale gli utenti iscritti a quel piano sono abilitati alla visualizzazione della recensione. In questa fase non viene gestito il fatto che un piano può abilitare l’esclusività della recensione pubblicata da un altro utente, persino non premium.

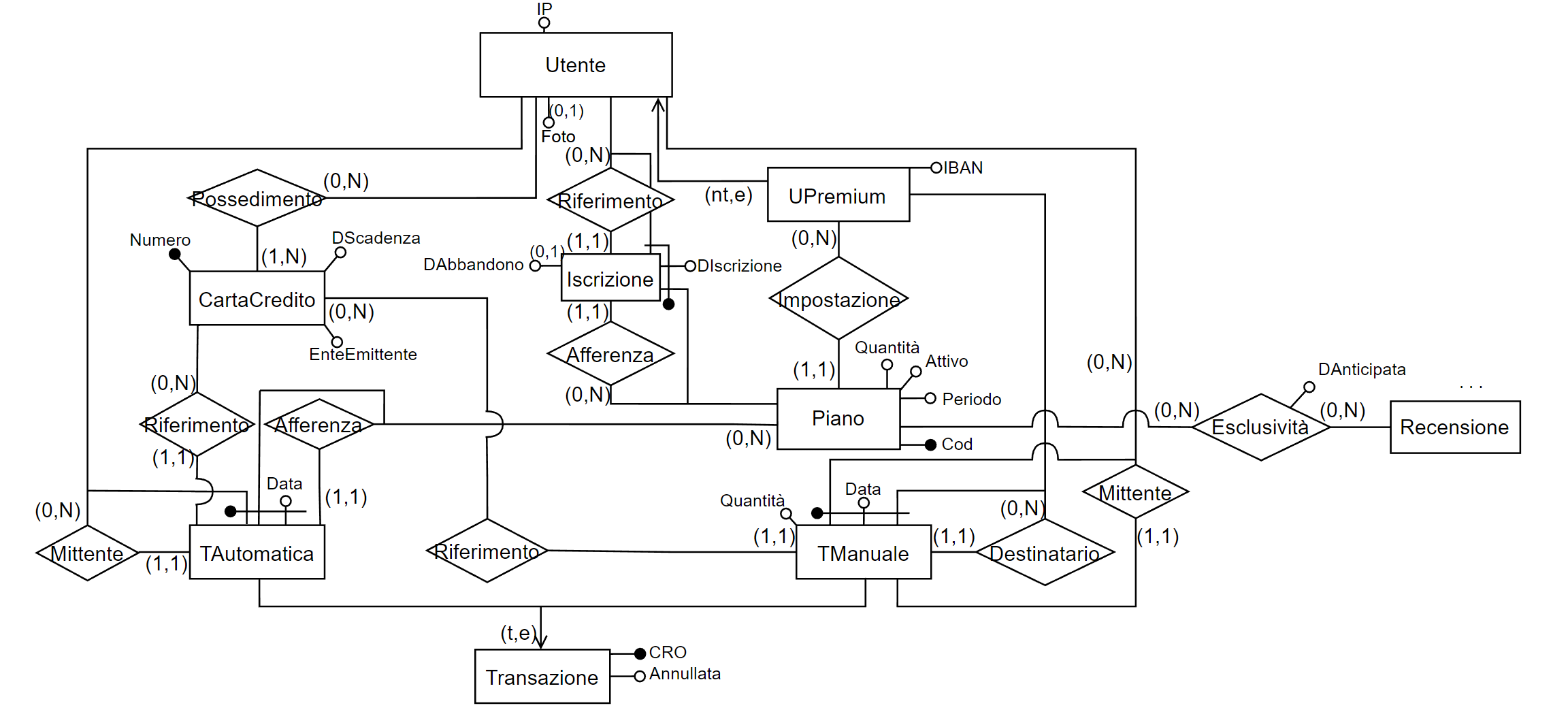
Le iscrizioni vengono gestite con altrettanta semplicità, con un’entità Iscrizione che è identificata dall’utente che la effettua, il piano, e la data. Essa contiene un’eventuale data di abbandono. Similmente al ragionamento fatto sull’entità Rimozione, occorre assicurarsi successivamente che non sia possibile iscriversi due volte ad uno stesso piano senza averlo prima abbandonato. Dato che bisogna memorizzare alcune caratteristiche sulla carta di credito associata ad un particolare utente, si crea un’entità apposita “CartaCredito” che si trova in un’associazione molti a molti con l’entità Utente. Essa è identificata dal numero e contiene informazioni quali la data di scadenza e l’ente emittente. Lo schema E/R finora considerato è il seguente:



Le due tipologie di transazione sono per certi aspetti differenti, in quanto la transazione manuale deve contenere la somma di denaro donata, la data, l’utente premium destinatario della donazione, la carta di credito utilizzata e l’utente donatore, mentre la transazione automatica si riferisce ad un piano che contiene già l’utente premium e la somma di denaro. Si realizza una gerarchia totale ed esclusiva per raggruppare le caratteristiche comuni delle due tipologie. Può essere ragionevole pensare che un piano possa essere modificato nel tempo da un utente premium e persino cancellato. Si può notare, tuttavia, una problematica legata a questa casistica; se di una transazione automatica si tiene solamente traccia del piano che l’ha attivata, allora nel momento in cui quest’ultimo viene eliminato i dati riguardanti la somma pagata e l’utente premium verranno totalmente persi, rendendo conseguentemente lo storico di transazioni inutilizzabile. A questo punto si possono intraprendere due strade, di cui si espongono i rispettivi lati positivi e negativi. Come prima cosa, è possibile salvare all’interno della transazione automatica anche i dati della transazione in aggiunta del piano che l’ha attivata. Questo comporta una certa ridondanza dei dati che implica un piccolo spreco di memoria e i costi associati al mantenimento della consistenza attraverso i trigger, tuttavia si può liberamente modificare la relazione dei piani senza avere alcuna ripercussione significativa sullo storico. D’altra parte, si può impedire la modifica permanente di un piano, sostituendola con l’aggiunta di un nuovo piano con i cambiamenti apportati. Adottando questa strategia si memorizzano quindi tutti i piani che sono mai esistiti. Dato che in questo caso si vuole memorizzare uno storico, e i dati riguardanti i piani passati possono rivelarsi utili, si opta per la seconda scelta.

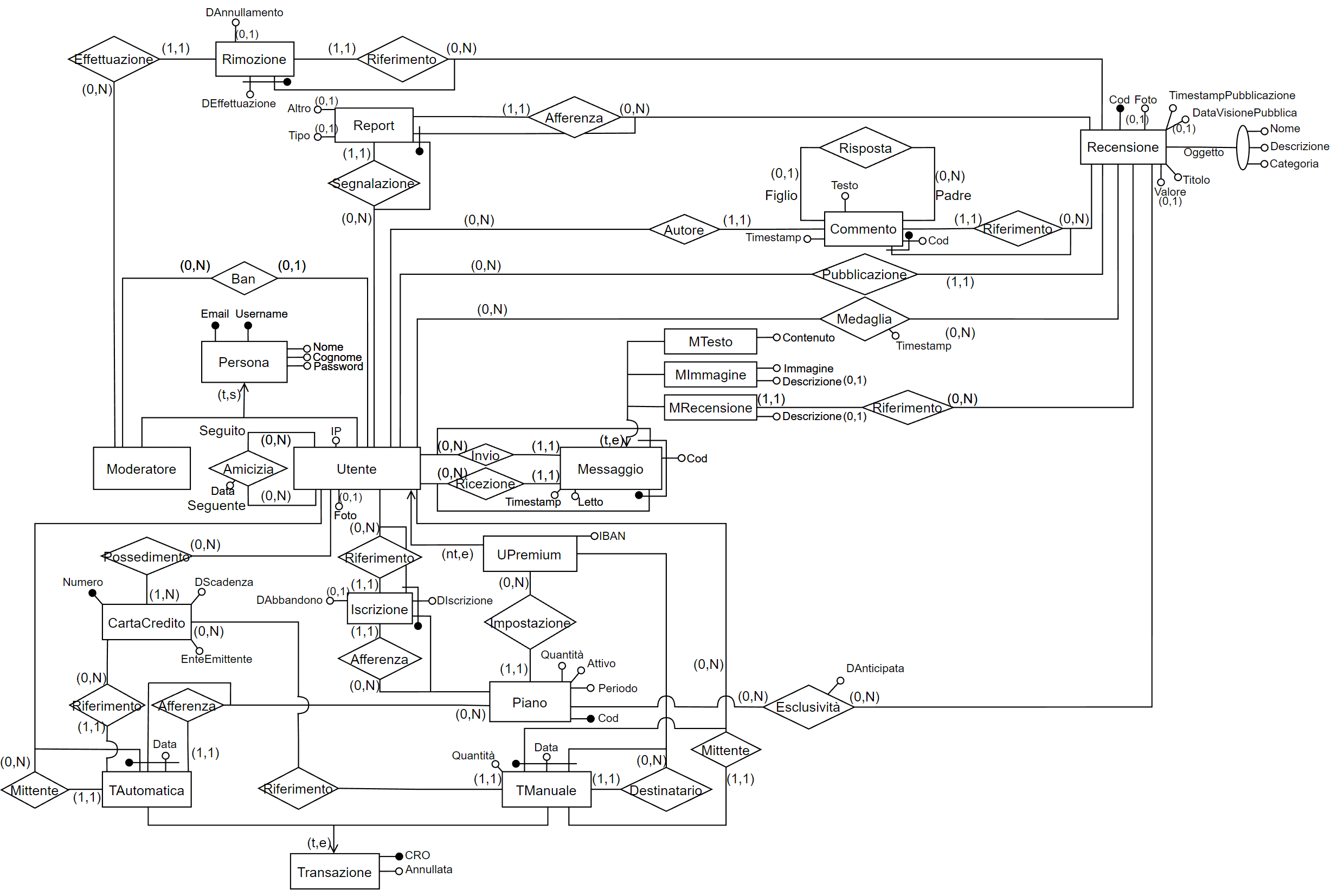
Da qua la progettazione si dirama ancora una volta, in quanto vi sono più modi per gestire la dicotomia tra piani attivi e inattivi. Una possibilità è quella di tenere due entità separate, e al posto di eliminare o modificare i dati già esistenti si crea una copia e la si inserisce nell’entità aggiuntiva “PianoInattivo”. Questo presenta il vantaggio considerevole della semplicità implementativa in certe condizioni, in quanto basta operare sull’entità “PianoAttivo” (o semplicemente “Piano” senza dover controllare ogni volta un attributo booleano e in generale impattare il funzionamento ordinario del database, il che potrebbe essere soggetto a errori. Tuttavia, a differenza della scelta di una semplice aggiunta di un attributo booleano, le relazioni presenti tra un piano (che sia attivo oppure no) e le altre entità potrebbero essere difficili da gestire. Come si vedrà nelle fasi successive, si opta per la presenza di un attributo booleano con l’obiettivo di ovviare il problema della complicazione delle interrogazioni facendo uso di una vista.

In seguito a queste riflessioni, il diagramma E/R risultante è il seguente:



Dato che nella transazione viene memorizzata sia la carta di credito sia l’utente donatore, come conseguenza negativa con questo modello E/R è possibile immagazzinare una transazione da parte di un utente che non è in possesso della carta a cui si riferisce la transazione stessa, si tratta di un altro vincolo che verrà gestito in seguito tramite i trigger.

Si presenta dunque lo schema finale:



**Progetto logico – Schema relazionale**

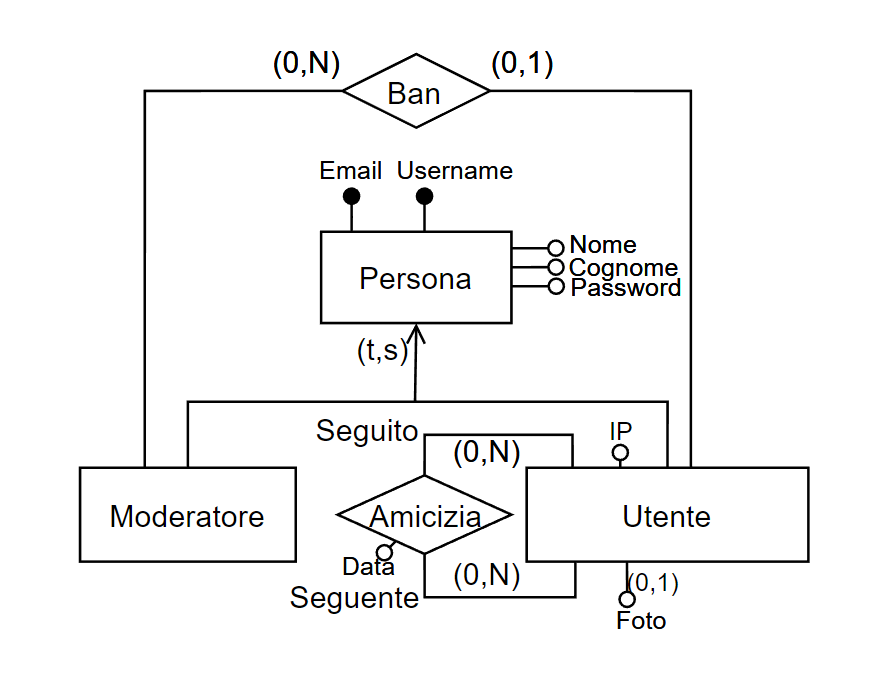
La progettazione logica permette di definire alcuni dettagli implementativi per poter operare nella pratica sui concetti stabiliti durante la fase di progetto concettuale, e viene decomposta nelle seguenti fasi:

* Eliminazione delle gerarchie ISA
* Selezione delle chiavi primarie, eliminazione delle identificazioni esterne
* Trasformazione degli attributi composti e/o multipli
* Traduzione di entità e associazioni in schemi di relazioni
* Verifica di normalizzazione

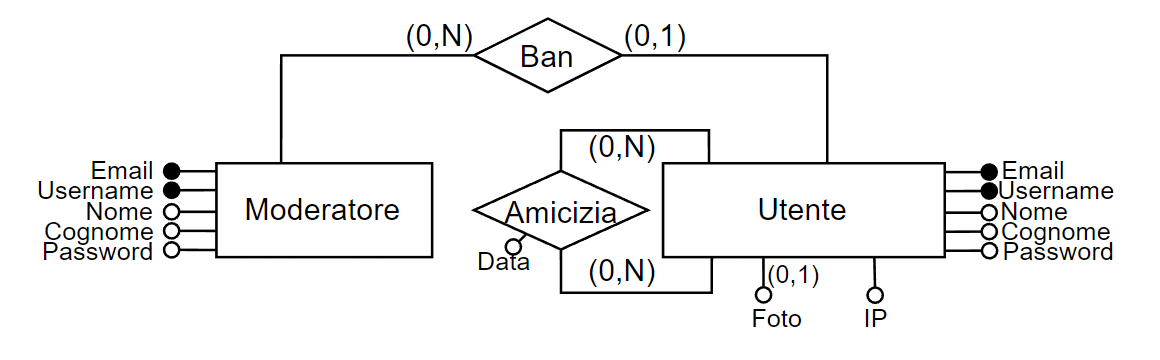
Si procede nell’ordine descritto, e si concluderà con una sezione sui dati derivati seguita dallo schema logico completo.

**Eliminazione delle gerarchia ISA**

Nello schema E/R sono presenti 4 gerarchie ISA, si propone di focalizzare l’attenzione sulla seguente gerarchia per prima:

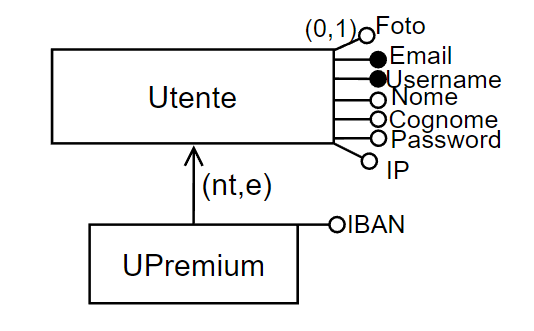


Nell’immagine sono evidenziate alcune associazioni la cui presenza influisce sulla decisione. Per effettuare una decisione appropriata sull’eliminazione, occorre tenere in considerazione che vi è un’ulteriore gerarchia sottostante, con Utente come entità padre. Proprio per questo motivo, e per il fatto che l’entità padre non si interfaccia con alcuna entità nella base di dati, si ritiene che la scelta migliore in questo caso sia un collasso verso il basso, come segue:

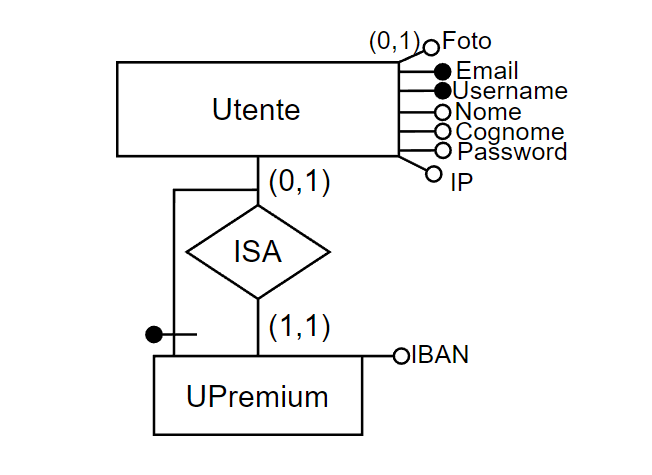


Questo richiederà, a livello implementativo, la creazione di alcuni trigger che si occuperanno di garantire che i valori di “Email” e “Username” siano univoci all’interno di entrambe le relazioni, dato che la gerarchia è esclusiva.

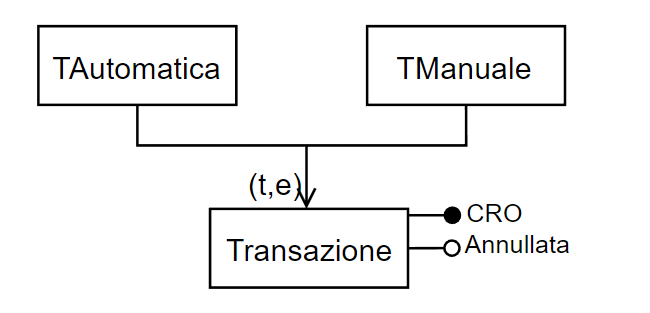
Si prende ora in considerazione la gerarchia immediatamente sottostante (nell’immagine è già stata applicata l’eliminazione della gerarchia già considerata):



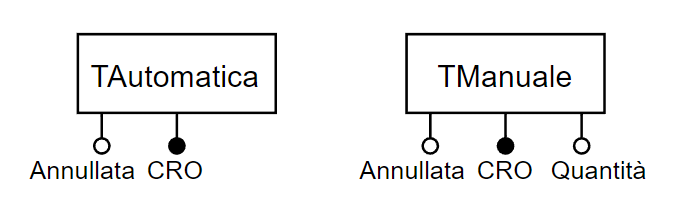
Vi sono numerose interazioni/associazioni che legano sia l’entità padre che l’entità figlia a tutte le altre, inoltre si può notare che vi sono numerose associazioni che legano le due entità Utente e UPremium, per esempio tutti gli utenti possono donare ma solamente gli utenti premium sono in grado di ricevere denaro. Si ritiene quindi che un mantenimento delle entità sia benefico in questo caso, dato che un collasso verso il basso non è possibile a causa della parzialità della gerarchia, e un collasso verso l’alto comporterebbe numerose complicazioni per la gestione delle varie associazioni. La sua traduzione è quindi la seguente:



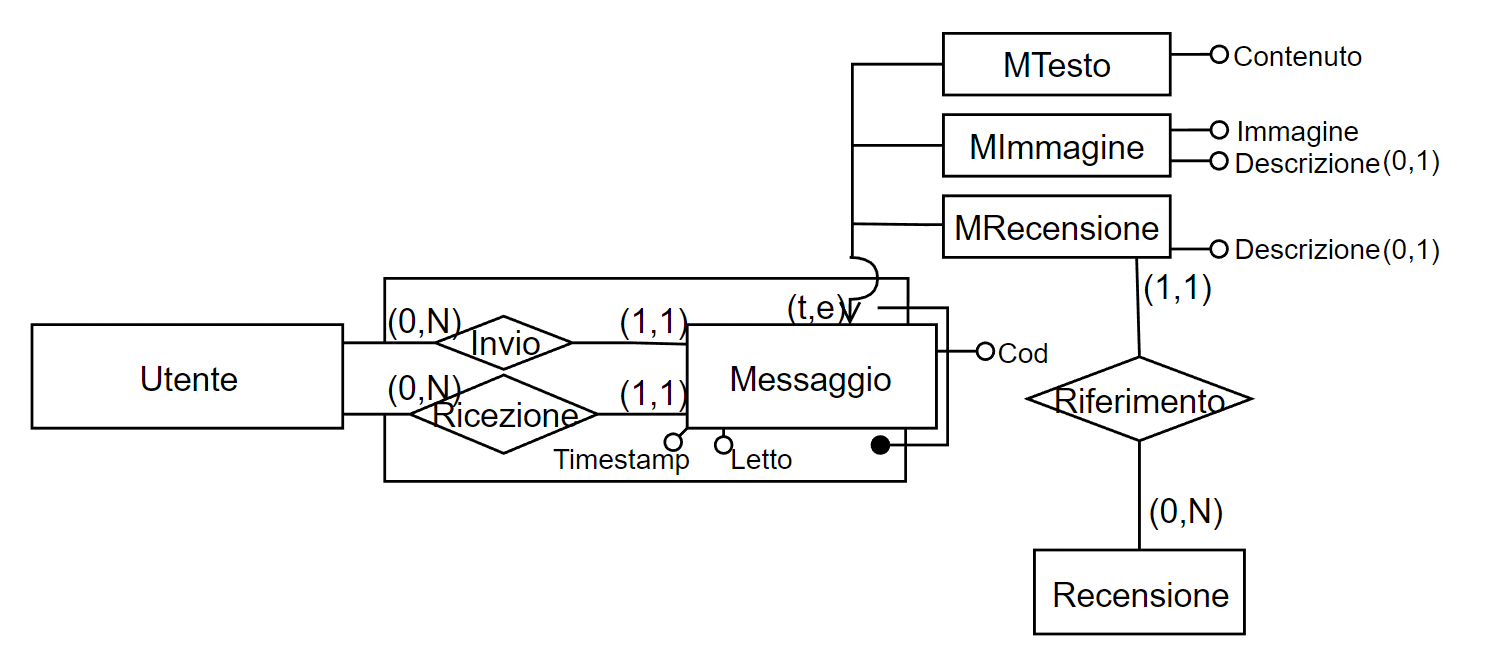
Per quanto riguarda la gerarchia relativa alle transazioni:



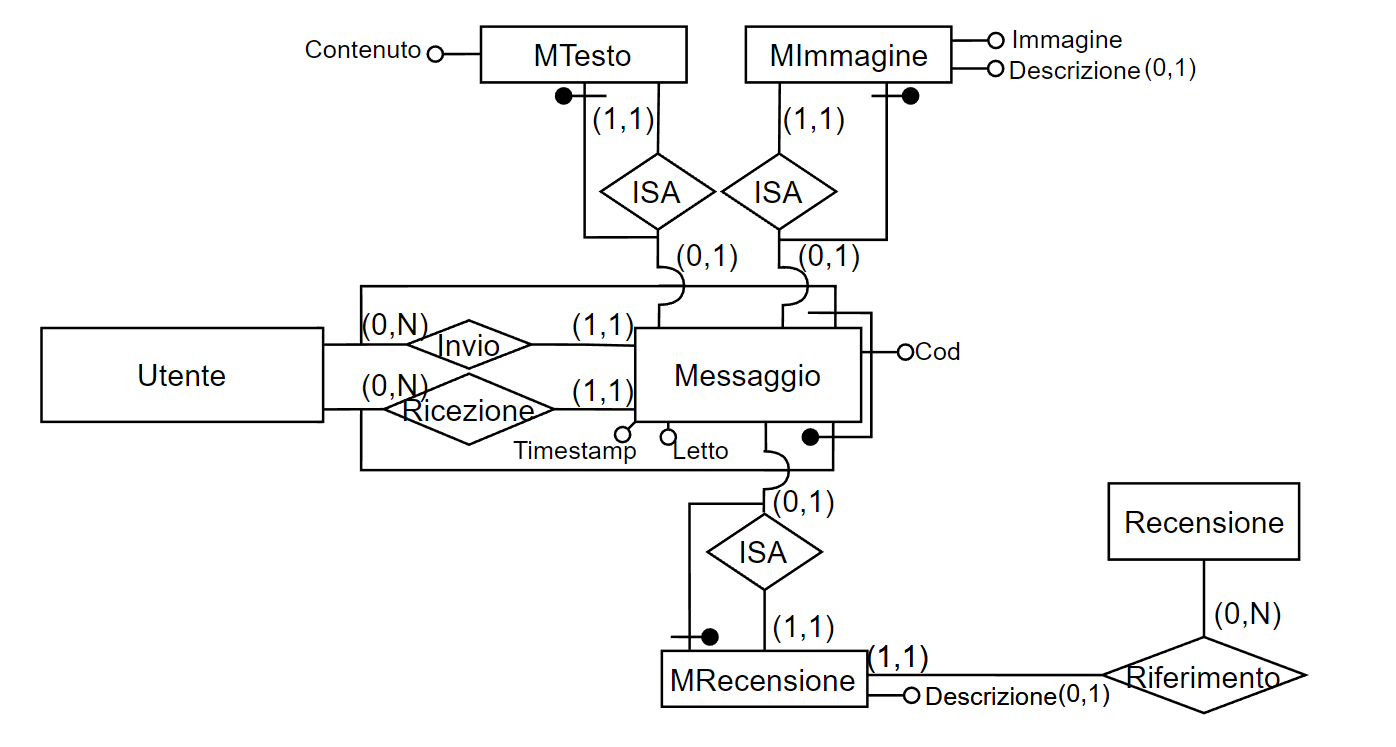
Essa può essere tradotta in più modi, tuttavia per il fatto che l’entità padre contiene solamente due attributi e nessuna associazione che la lega ad altre entità, un semplice collasso verso il basso è sufficiente.



Infine occorre gestire l’eliminazione della gerarchia di Messaggio. Si tratta di una gerarchia totale ed esclusiva, rappresentata nell’immagine seguente:



Innanzitutto occorre specificare che ci si aspetta che la maggior parte dei messaggi siano testuali, perciò non converrebbe effettuare un collasso verso l’alto in quanto si avrebbe un numero di NULL eccessivamente elevato. A questo punto le due alternative sono pressochè interscambiabili, un collasso verso il basso implica il dover realizzare un trigger che garantisce che venga rispettata l’unicità del codice data una coppia mittente-destinatario di utenti, mentre con il mantenimento delle entità occorrerebbe crearne uno per assicurarsi che ad un Messaggio è collegato uno e soltanto uno tra MTesto, MImmagine, e MRecensione (per la totalità della gerarchia). In questo specifico caso, in una chat può essere efficace poter ottenere uno storico dei messaggi in maniera efficiente e veloce senza entrare nel dettaglio e visualizzarne il contenuto. Questa separazione tra gli attributi generali del Messaggio, ovvero “Timestamp” e “Letto”, e i contenuti, può essere desiderata oppure no, a seconda dell’applicazione, e ha il lato negativo che in questo caso si occupa più spazio a causa della ripetizione della chiave composta. È stato scelto un mantenimento delle entità, da cui si ottiene:



**Selezione delle chiavi primarie, eliminazione delle identificazioni esterne**

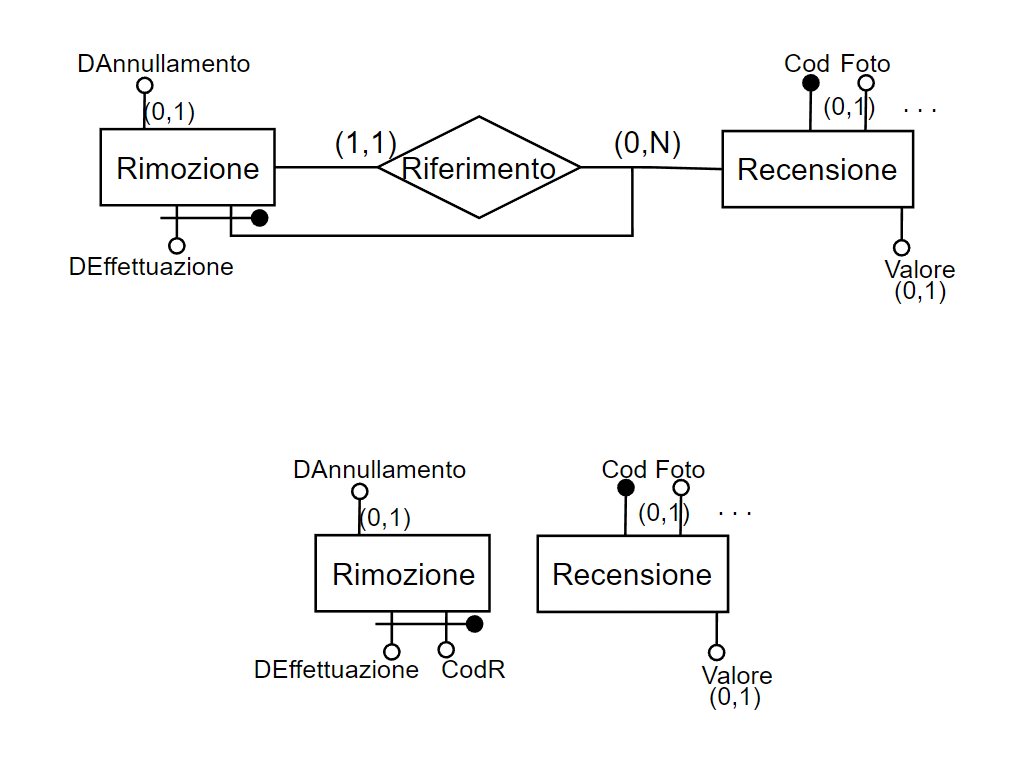
Vi è una scelta di selezione delle chiavi primarie solamente in quattro casi: Utente, Moderatore, TManuale, TAutomatica. Negli ultimi due casi una delle due chiavi è composta ed esterna, quindi occorrerà prima eliminare quella esterna per poi scegliere tra le due. Tra lo username e l’email, è stata scelta come chiave primaria lo username per il semplice fatto che è generalmente più corto e che si suppone che verrà utilizzato più spesso per cercare un determinato utente.

UPremium ha una chiave esterna che viene tradotta in questo modo:

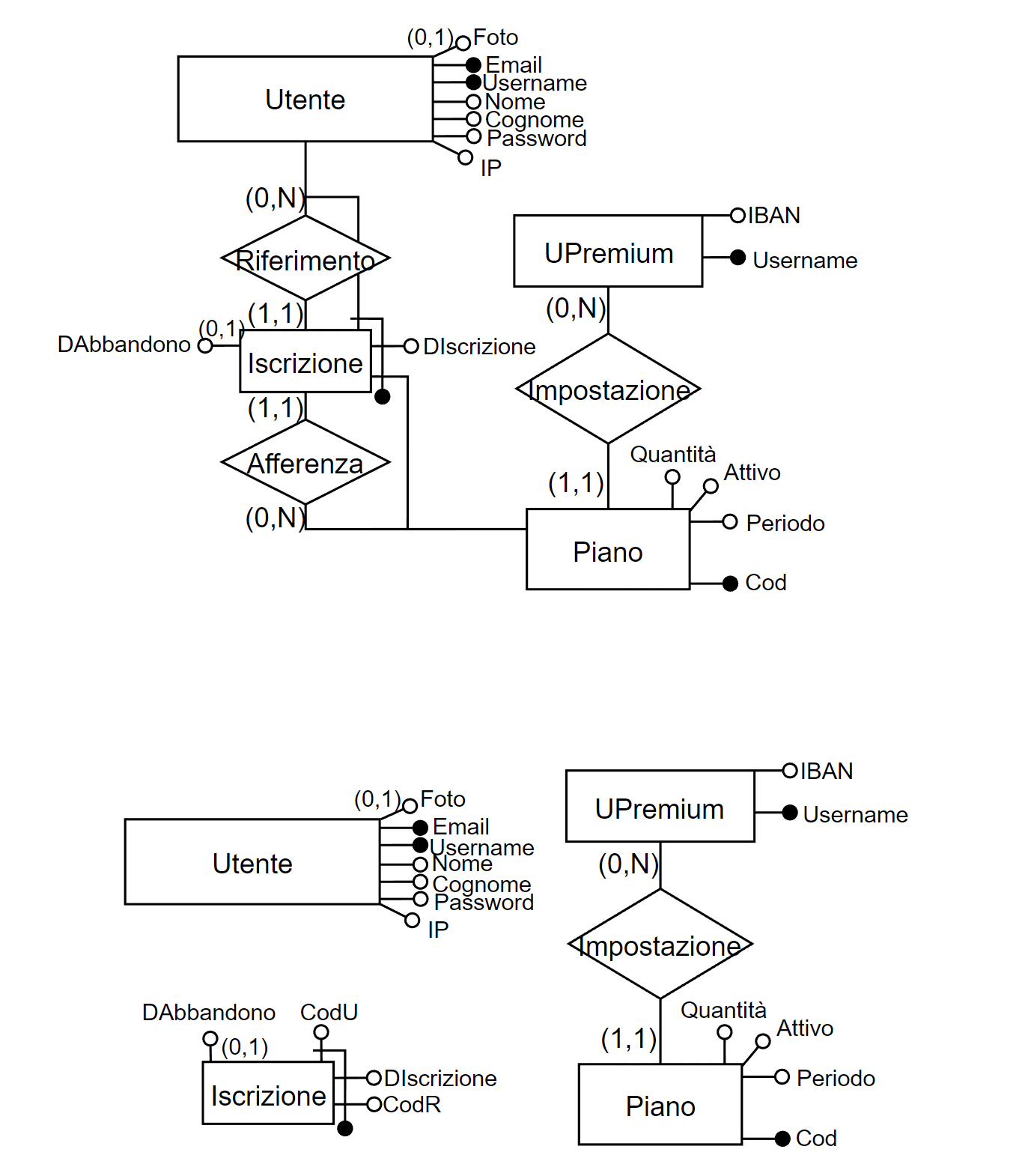
Immagine che contiene testo

Descrizione generata automaticamente

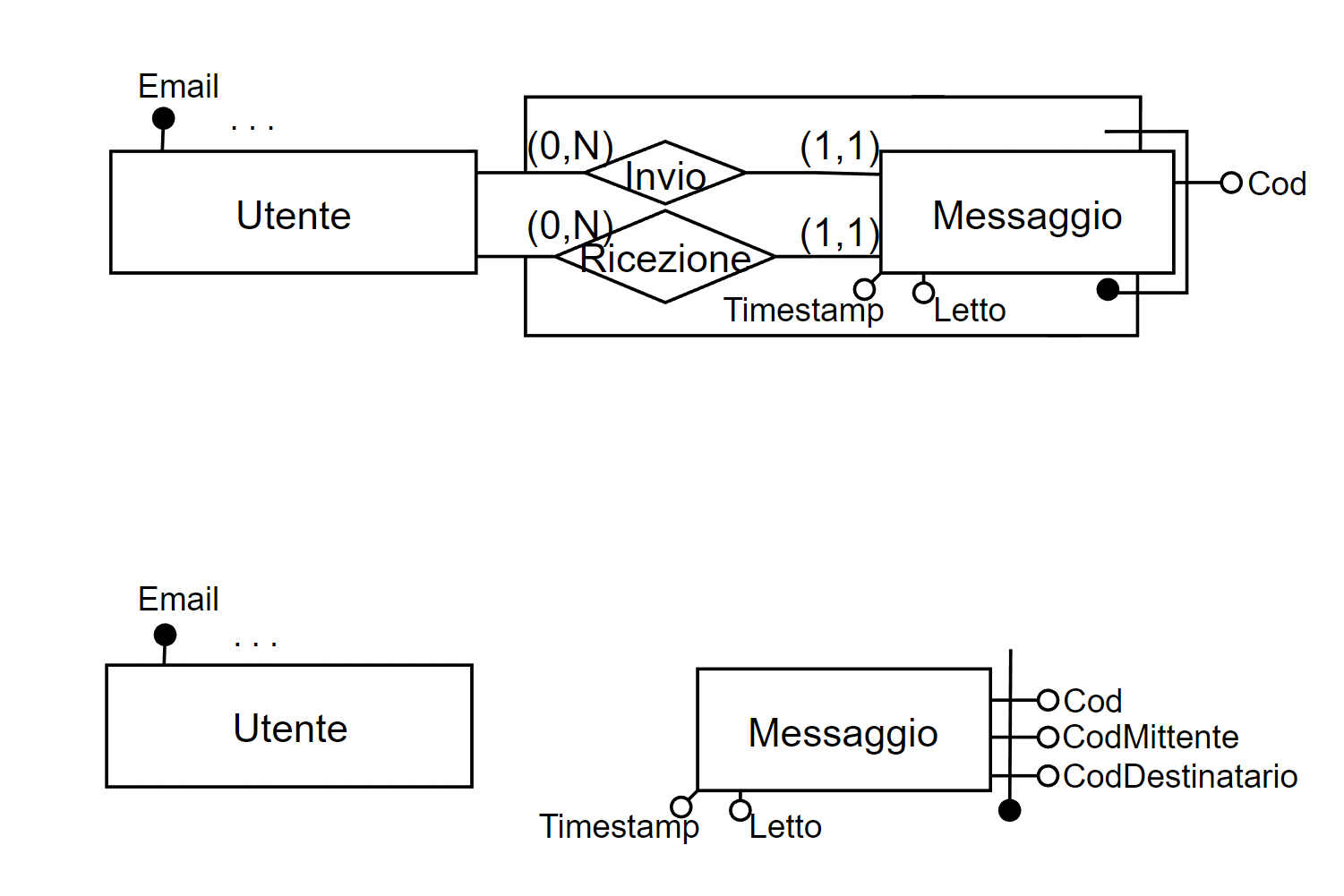
L’entità rimozione possiede una chiave composta di cui una parte deriva dall’associazione Riferimento che la lega a recensione.



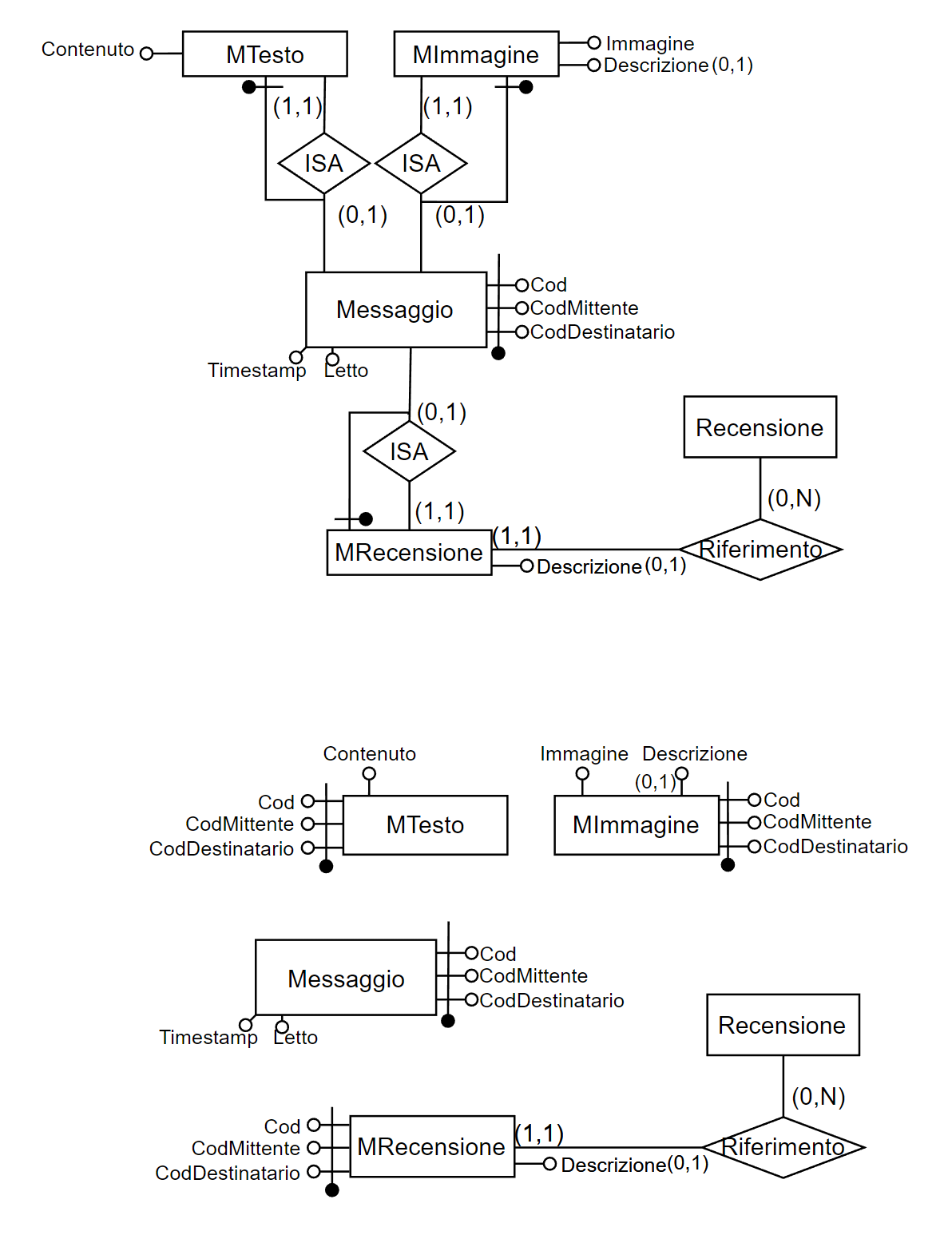
Allo stesso modo, l’entità Iscrizione ha una chiave che è formata dall’associazione Riferimento, Afferenza, e dalla data nella quale l’utente ha effettuato una determinata iscrizione ad un piano.



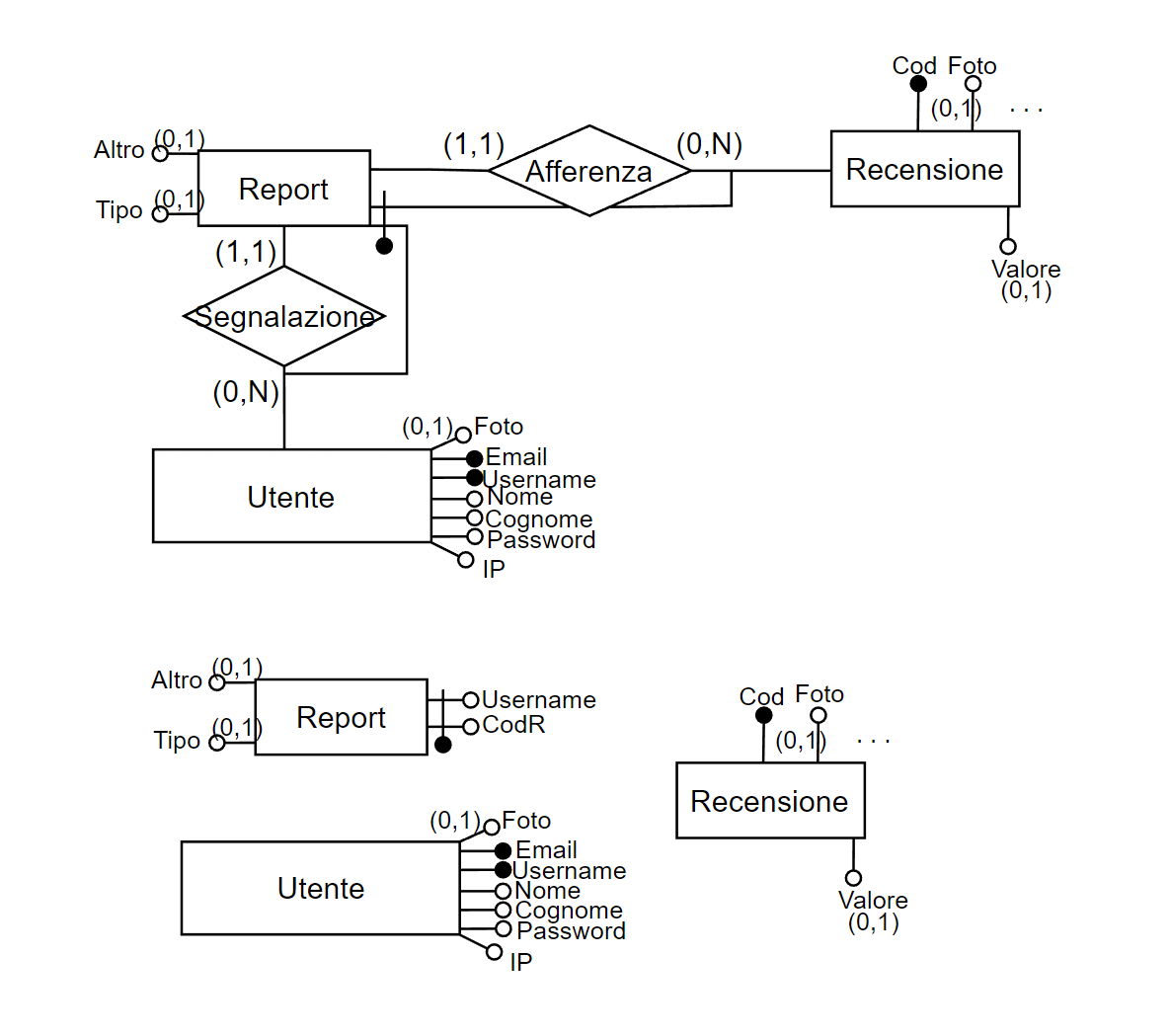
L’entità Messaggio è identificata da un codice che è solamente univoco tra i messaggi che hanno una determinata coppia di utenti come mittente-destinatario, che si traduce nel seguente modo:



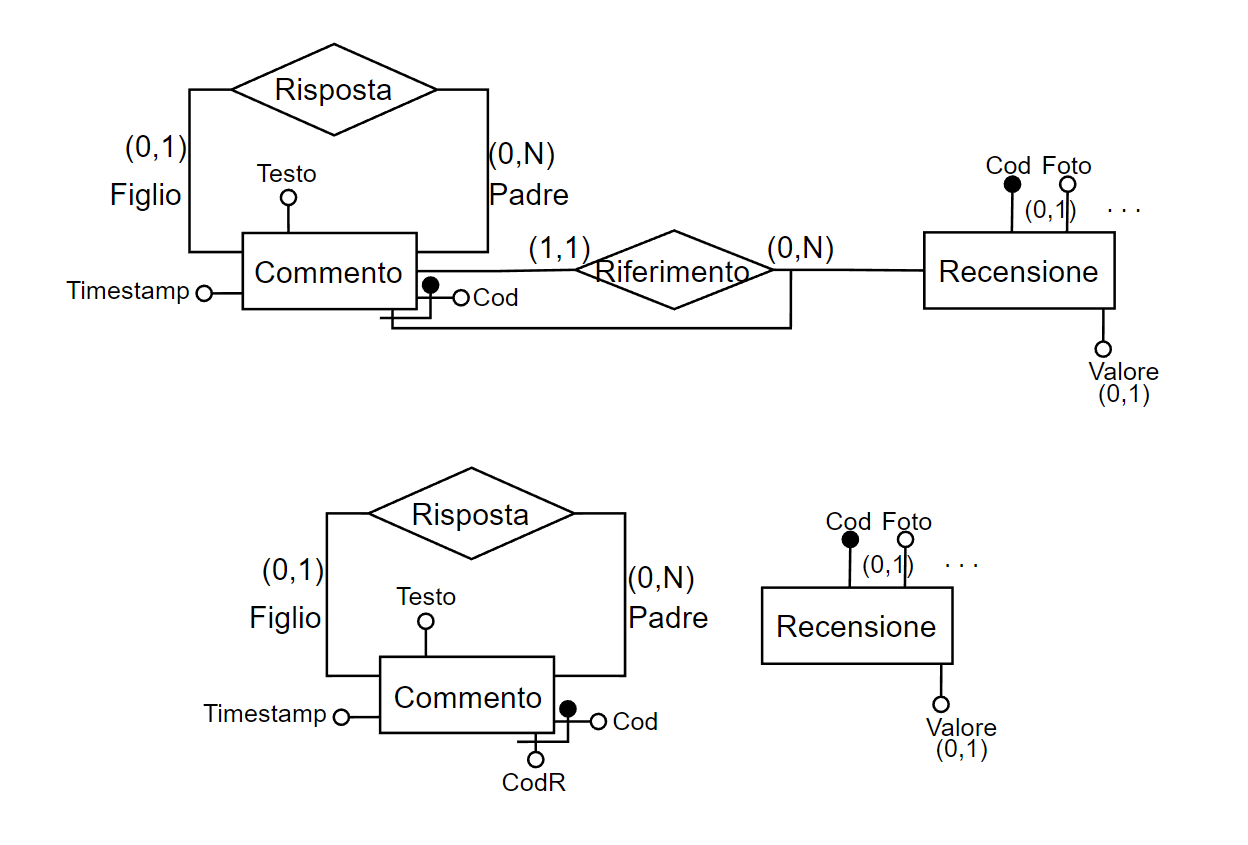
Le relazioni MTesto, MImmagine e MRecensioni vengono tutte trasformate in questo modo:



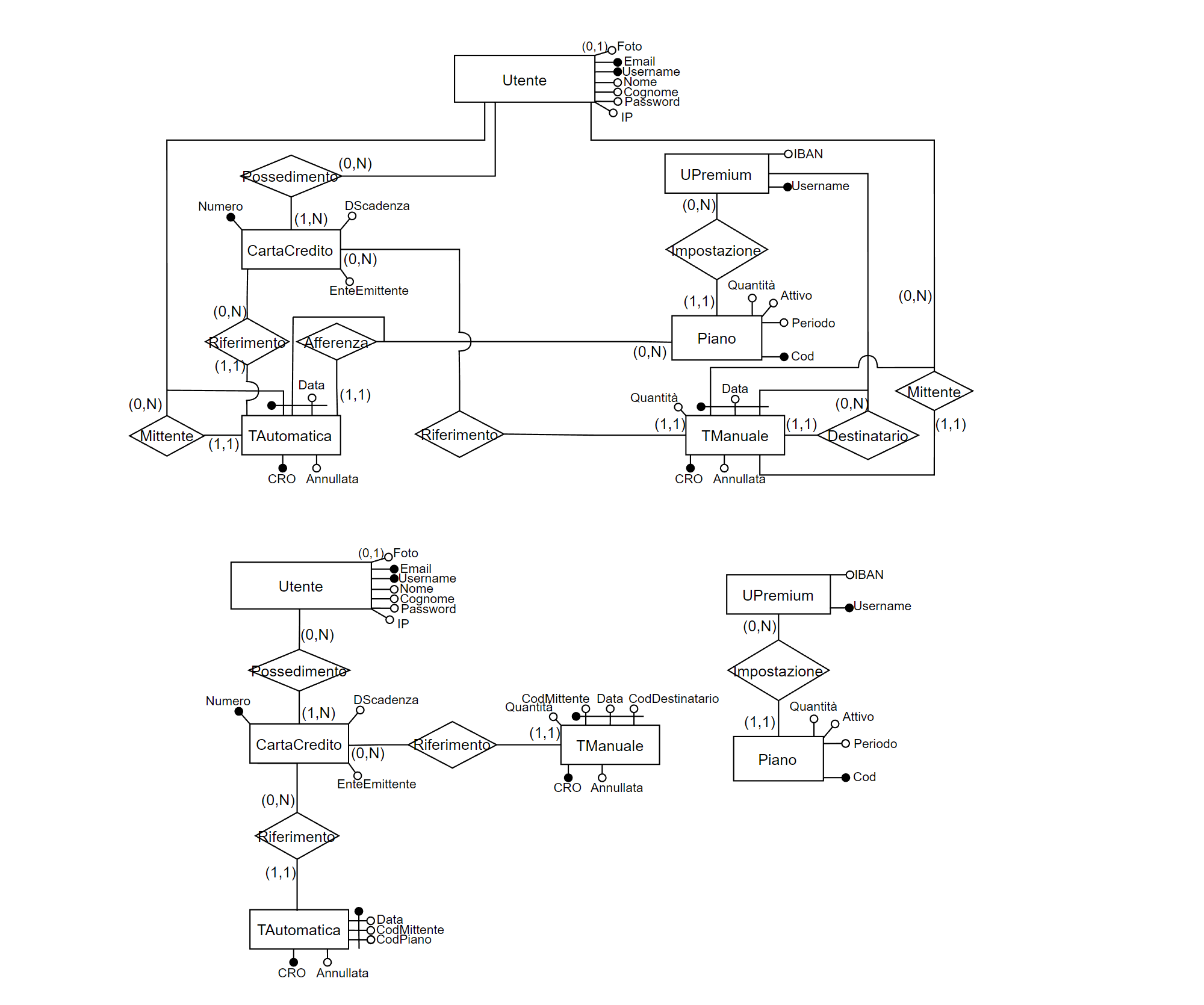
Un Report è identificato dall’utente che lo ha effettuato e la recensione a cui si riferisce, per cui si ha:



Un Commento è identificato da un codice univoco all’interno della recensione a cui si riferisce, e questo nella schema logico ha la seguente traduzione:



TAutomatica e TManuale sono simili, la prima è identificata dalla data di effettuazione, dal piano a cui si riferisce e dal mittente, mentre la seconda dalla data, dal mittente e dal destinatario.



Per TAutomatica e TManuale si compie la stessa decisione sulla scelta della chiave primaria, ovvero il CRO, lasciando le due chiavi composte come chiavi alternative. Questo per il semplice fatto che in generale si preferisce una chiave semplice piuttosto che una complessa.

**Trasformazione degli attributi multipli e/o composti**

L’unico attributo composto presente all’interno dello schema E/R è l’oggetto della recensione. Dato che uno stesso oggetto, dotato di un titolo e di una descrizione che può essere in certi casi molto lunga, può ripetersi più volte, per non occupare eccessivamente spazio è stato deciso di creare una relazione che contiene i dati dell’oggetto e un codice intero identificativo che verrà utilizzato dalla recensione qualora volesse associarsi ad esso.

Si può osservare la trasformazione nel seguente schema E/R:

[Immagine trasformazione]

Sono anche presenti alcuni attributi ripetuti, come la categoria dell’oggetto, il tipo di un report e l’ente emittente di una carta di credito; in tutti questi 3 casi è stato scelto di non trasformarli in entità separate in quanto è la scelta più semplice dal punto di vista progettuale, non occorre effettuare accessi in più su relazioni differenti per arrivare a queste informazioni, e lo spreco di spazio è alquanto trascurabile.

**Traduzioni di entità e associazioni in schema di relazioni**

Si prendono in esame prima di tutto le relazioni Moderatore, Utente, e UtentePremium.

UtentePremium non è legato a nessun’altra entità tramite associazione, perciò si può applicare una traduzione standard, per quanto riguarda invece Moderatore e Utente, sono legati da un’associazione uno a molti. Si suppone in generale che il numero di utenti “banditi” da un moderatore sia vastamente ridotto in proporzione al numero di utenti totale, motivo per il quale è stato deciso di tradurre l’associazione come una relazione a parte. L’amicizia tra utenti è un’auto-associazione N:N con partecipazione facoltativa da entrambe le parti, che viene tradotta come una relazione in possesso di due chiavi esterne che si riferiscono a due utenti separati. Si ricorda che un’amicizia non è necessariamente ricambiata, dunque non sono necessari trigger o quant’altro per imporre l’unicità della coppia (ovvero rendere impossibile gli inserimenti A-B e B-A). Lo schema logico risultante è il seguente:

Moderatore(Email, Username, Nome, Cognome, Password)

PK: Username

AK: Email

Utente(Email, Username, Nome, Cognome, Password, Foto, IP)

PK: Username

AK: Email

Ban(CodUtente, CodModeratore)

PK: CodUtente

FK: CodUtente REFERENCES Utente

FK: CodModeratore REFERENCES Moderatore

UtentePremium(CodUtente, IBAN)

PK: CodUtente

FK: CodUtente REFERENCES Utente

Amicizia(Seguito, Seguente, Data)

PK: (Seguito, Seguente)

FK: Seguito REFERENCES Utente

FK: Seguente REFERENCES Utente

Si passa ora a ciò che riguarda le recensioni. L’associazione 1:N tra Oggetto e Recensione viene tradotta accorpandola all’interno di Recensione, trattandosi di una partecipazione obbligatoria rispetto al lato molti. La separazione in una relazione a parte potrebbe avere determinati vantaggi in termini di compattezza e contiguità dei collegamenti, e nei casi in cui sia estremamente frequente la scansione delle coppie CodR-CodO questo può risultare vantaggioso in termini di efficienza di questa operazione. Tuttavia non si tratta del caso in questione, dunque si opta per una traduzione più comune, come riportato di seguito. Proseguendo con la traduzione, la Medaglia non è altro che un’associazione molti a molti con partecipazione facoltativa da entrambe le parti, che risulta nella creazione di una relazione distinta che contiene gli identificatori dell’utente che l’ha “creata” e della recensione a cui si riferisce.

Oggetto(CodO, Nome, Descrizione, CodC, Categoria)

PK: CodO

Recensione(CodR, Foto, DataVisionePubblica, Titolo, Valore,

Descrizione, DataPubblicazione, CodO, CodUtente)

PK: CodR

FK: CodO REFERENCES Oggetto

FK: CodUtente REFERENCES Utente

Medaglia(CodUtente, CodR, Timestamp)

PK: (CodUtente, CodR)

FK: CodUtente REFERENCES Utente

FK: CodR REFERENCES Recensione

L’entità Rimozione è legata alle entità Moderatore e Recensione tramite le associazioni Effettuazione e Riferimento rispettivamente, che sono 1:N e che possono essere accorpate nella relazione Rimozione, come mostra lo schema logico qui di seguito. Gli utenti possono effettuare report e commenti. L’associazione tra Report e Utente è 1:N, e la relativa traduzione consiste nel suo accorpamento nella relazione Report, e lo stesso può essere detto per l’associazione tra Report e Recensione. L’associazione tra Commento e Utente è 1:N, si può quindi accorpare all’entità Commento, e lo stesso ragionamento si può applicare per quanto riguarda l’associazione tra Commento e Recensione. Per quanto riguarda l’auto-associazione, la traduzione standard di accorpamento in Commento sarebbe caratterizzata dalla presenza di due attributi CodCRisposta e CodRRisposta, che costituiscono la chiave primaria dell’entità stessa. Con questa traduzione perfettamente valida, occorrerebbe creare un trigger per fare in modo che un commento non possa rispondere ad una recensione diversa da quella a cui si riferisce. In questo caso specifico, per via di com’è fatta la chiave di Commento, si può gestire questo vincolo a partire da questa fase. Basta eliminare l’attributo CodRRisposta e impostare la coppia (CodCRisposta, CodR) come Foreign Key che si riferisce ad un commento, dove CodR è il codice della recensione a cui si riferisce. In questo modo un commento può rispondere solamente ad un altro commento che si riferisce alla stessa recensione. Di seguito è riportato lo schema relativo a Report e Commento per esteso:

Rimozione(DataEffettuazione, CodR, DAnnullamento, CodModeratore)

PK: (DataEffettuazione, CodR)

FK: CodR REFERENCES Recensione

FK: CodModeratore REFERENCES Moderatore

Report(CodUtente, CodR, Tipo, Altro)

PK: (CodUtente, CodR)

FK: CodUtente REFERENCES Utente

FK: CodR REFERENCES Recensione

Commento(CodC, CodR, CodU, Timestamp, Testo, CodCRisposta)

PK: (CodC, CodR)

FK: CodR REFERENCES Recensione

FK: CodU REFERENCES Utente

FK: (CodCRisposta, CodR) REFERENCES Commento

Tra Messaggio è Utente sono presenti due associazioni 1:N, entrambe con partecipazione facoltativa da parte dell’utente e obbligatoria da parte del messaggio. Queste vengono tradotte accorpandole alla relazione Messaggio. Le varie tipologie di messaggio MTesto, MImmagine e MRecensione, per effetto dell’eliminazione delle gerarchie ISA e delle identificazioni esterne non hanno alcune associazioni da analizzare e vengono quindi tradotte in maniera standard.

Messaggio(CodM, CodMittente, CodDestinatario, Letto, Timestamp)

PK: (CodM, CodMittente, CodDestinatario)

FK: CodMittente REFERENCES Utente

FK: CodDestinatario REFERENCES Utente

MTesto(CodM, CodMittente, CodDestinatario, Contenuto)

PK: (CodM, CodMittente, CodDestinatario)

FK: (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio

MImmagine(CodM, CodMittente, CodDestinatario, Immagine, Descrizione)

PK: (CodM, CodMittente, CodDestinatario)

FK: (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio

MRecensione(CodM, CodMittente, CodDestinatario, CodR)

PK: (CodM, CodMittente, CodDestinatario)

FK: (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio

FK: CodR REFERENCES Recensione

Tra le entità CartaCredito e Utente vi è un’associazione N:M. Essa viene tradotta come una relazione separata contenente i codici di un utente e delle sue rispettive carte di credito possedute. Tale relazione viene chiamata CartaUtente, per la mancanza di un nome migliore.

CartaCredito(Numero, DScadenza, EnteEmittente)

PK: Numero

CartaUtente(CodCarta, CodU)

PK: (CodCarta, CodU)

FK: CodU REFERENCES Utente

FK: CodCarta REFERENCES Carta

Un Piano è impostato da un solo UtentePremium, dunque l’associazione che lega le due entità viene accorpata nella relazione Piano, nello stesso modo in cui Iscrizione si trova in associazioni 1:N con Piano e Recensione. Tra Piano e Recensione vi è un’associazione N:M, da cui risulta naturale estrapolarla in una relazione a parte, denominata appunto Esclusività.

Piano(CodP, Quantita, Periodo, Attivo, CodUtentePremium)

PK: CodP

FK: CodUtentePremium REFERENCES UtentePremium

Esclusivita(DAnticipata, CodP, CodR)

PK: (CodP, CodR)

FK: CodP REFERENCES Piano

FK: CodR REFERENCES Recensione

Iscrizione(CodP, CodUtente, DIscrizione, DAbbandono)

PK: (CodP, CodUtente, DIscrizione)

FK: CodUtente REFERENCES Utente

FK: CodP REFERENCES Piano

I due tipi di transazioni sono simili e per questo motivo verranno descritti assieme. Le transazioni sono coinvolte in un’associazione 1:N con Utente, con CartaCredito, e nel caso di TransazioneAutomatica Piano, altrimenti UPremium. Tutte queste associazioni vengono accorpate nel rispettivo lato molti (TransazioneAutomatica e TransazioneManuale), come si può vedere nel seguente schema logico:

TransazioneAutomatica(CRO, Annullata, Data, CodPiano, CodUtente,

NumeroCartaDiCredito)

PK: CRO

FK: CodPiano REFERENCES Piano

FK: CodUtente REFERENCES Utente

FK: NumeroCartaDiCredito REFERENCES CartaCredito

AK: (Data, CodPiano, CodUtente)

TransazioneManuale(CRO, Annullata, Data, CodMittente, CodDestinatario,

NumeroCartaDiCredito)

PK: CRO

FK: CodMittente REFERENCES Utente

FK: CodDestinatario REFERENCES Utente

FK: NumeroCartaDiCredito REFERENCES CartaCredito

AK: (Data, CodMittente, CodDestinatario)

**Verifica della normalizzazione**

Una volta effettuata l’analisi di normalizzazione è stato appurato che le relazioni si trovano in forma normale.

**Studio di dati derivati**

Si ritiene opportuno effettuare le analisi dei dati derivati per stabilire quali sono benefici da inserire e mantenere nella base di dati e quali non lo sono. Per una misurazione più accurata, gli accessi in lettura si contano solo se strettamente necessari (non prima di ogni scrittura). Inoltre i dati sono puramente esemplificativi, potrebbero non essere completamente realistici.

La prima discussione è svolta sulla visibilità delle recensioni, per essere più precisi, è possibile stabilire se una recensione è attualmente nello stato “invisibile” andando a cercare nella relazione Rimozione, e se è presente una rimozione senza data di annullamento allora la recensione è invisibile. Questa procedura può essere semplificata nel caso in cui si utilizzasse un attributo “Visibile” nella recensione.

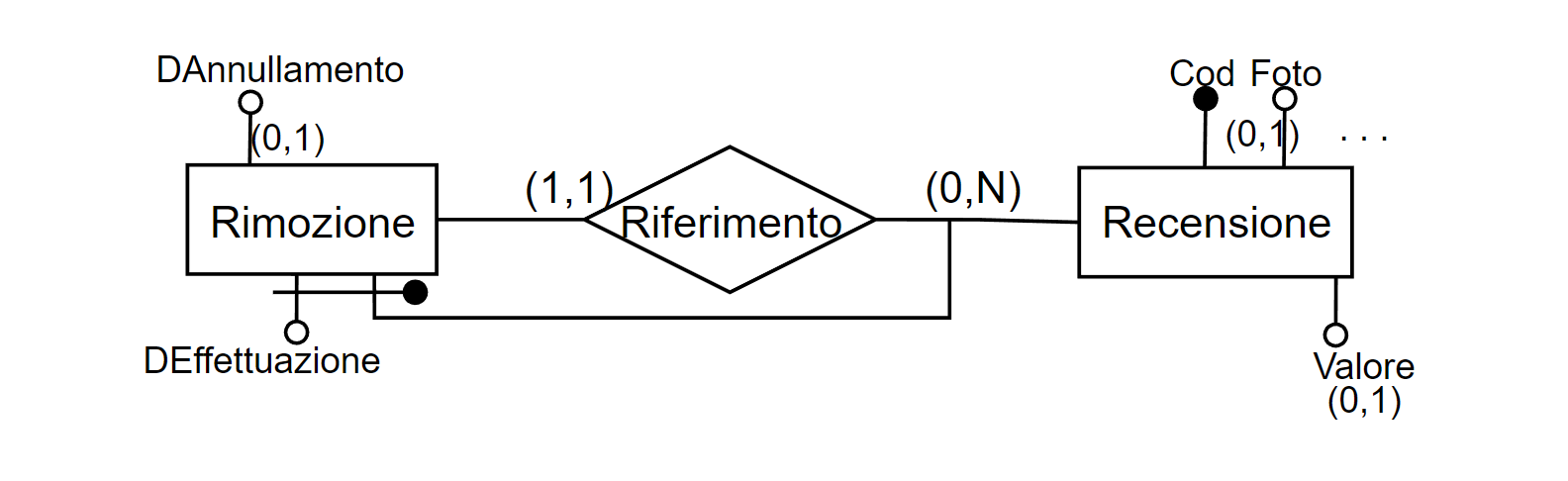


Tabella dei volumi:

|  |  |  |
| --- | --- | --- |
| Concetto | Tipo | Volume dati |
| Rimozione | E | 50,000 |
| Recensione | E | 1,000,000 |

Tabella delle operazioni:

|  |  |  |
| --- | --- | --- |
| Operazione | Tipo | Frequenza |
| Rimozione Recensione | I | 100/G |
| Annullamento Rimozione | I | 8/G |
| Accesso Recensione | I | 2,000,000/G |

Può sembrare a prima vista che la presenza del dato derivato sia estremamente vantaggiosa; tuttavia, per un’analisi rigorosa è opportuno verificare concretamente.

Con l’utilizzo dei trigger è possibile introdurre un vincolo che non permetta un’altra rimozione di una recensione che sia già stata rimossa (cioè per la quale esista un record in “rimozione” che abbia data annullamento nulla). Si ricorda che l’associazione Riferimento è stata tradotta con un accorpamento a Recensione, dunque nello studio del dato derivato non si contano gli accessi di lettura/scrittura in Riferimento perché i suoi dati sono già contenuti all’interno di Rimozione.

Inoltre, si suppone di essere già a conoscenza del codice della recensione in caso di rimozione e di accesso, e nell’operazione di annullamento, oltre al codice della recensione si è a conoscenza della data di effettuazione dell’ultima rimozione (quella da annullare).

Con l’utilizzo del dato derivato si ottengono i seguenti risultati:

Rimozione recensione:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Rimozione | 1 | S |
| Recensione | 1 | S |

2\*2 \* 100 = 400/G

Annullamento rimozione:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Rimozione | 1 | S |
| Recensione | 1 | S |

2\*2 \* 8 = 32/G

Accesso recensione:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Recensione | 1 | L |

1\* 2,000,000 = 2,000,000/G

Si ha quindi un totale di 2,000,432/G.

Per quanto riguarda il caso in cui non si utilizzi il dato derivato:

Rimozione recensione:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Rimozione | 1 | S |

1\*2 \* 100 = 200/G

Annullamento rimozione:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Rimozione | 1 | S |

1\*2 \* 8 = 16/G

Accesso recensione:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Rimozione | 50,000/1,000,000 | L |
| Recensione | 1 | L |

(1+50,000/1,000,000) \* 2,000,000 = 2,100,000/G

In realtà l’accesso a recensione si verifica soltanto nel caso in cui sia visibile, altrimenti l’accesso a tale recensione viene annullato. Ipotizzando che, solitamente, su 100 recensioni a cui si vuole accedere, 95 siano visibili, il numero di accessi giornaliero si avvicina di più al seguente risultato:

(95/100+50,000/1,000,000) \* 2,000,000 = 2,000,000/G

Si ha quindi un totale di 2,000,216/G.

Dunque, al contrario di quanto ipotizzato, il dato derivato non conviene in questo caso, anche se la differenza del numero di accessi giornalieri nei due casi è trascurabile. La conclusione potrebbe quindi cambiare con una piccola variazione dei dati e delle supposizioni fatte. È anche da esplicitare il fatto che per un’analisi più accurata sarebbe adatto effettuare misurazioni nella pratica, e il risultato potrebbe variare in presenza di dati che rappresentano più accuratamente un social network di medie dimensioni. Tuttavia per lo scopo di questo progetto si suppone che queste approssimazioni rappresentino la realtà con un errore sufficientemente ridotto.

Una richiesta di cui si è parlato precedentemente è la limitazione di un numero massimo di recensioni e di commenti che un utente può effettuare in un’ora. Questo dev’essere senz’altro gestito con l’utilizzo di trigger, tuttavia quest’operazione di controllo (che dev’essere eseguita ad ogni pubblicazione di un commento e di una recensione) può essere molto costosa, motivo per il quale può essere vantaggioso l’utilizzo di un dato derivato che per ogni utente memorizza il numero di pubblicazioni effettuate, e che viene azzerato per tutti gli utenti allo scattare di xx:00. Si studia di seguito il numero di operazioni di lettura e scrittura in entrambi i casi. È chiaro che, nel momento in cui si voglia aggiungere una recensione oppure un commento, è noto il codice dell’utente che la vuole pubblicare.

Tabella dei volumi:

|  |  |  |
| --- | --- | --- |
| Concetto | Tipo | Volume dati |
| Utente | E | 400,000 |
| Commento | E | 1,450,000 |
| Recensione | E | 1,000,000 |

Tabella delle operazioni:

|  |  |  |
| --- | --- | --- |
| Operazione | Tipo | Frequenza |
| Aggiunta Recensione | I | 80,000/G |
| Aggiunta Commento | I | 120,000/G |
| Azzeramento contatore | I | 24/G |

L’ultima operazione è soltanto presente nel caso in cui sia memorizzato nel database il dato derivato, la cui frequenza giornaliera è esattamente 24, una volta per ogni ora (da 00:00 a 23:00).

Con l’utilizzo del dato derivato, ad ogni aggiunta occorre aggiornare il contatore del rispettivo utente che ha intenzione di pubblicarla, conseguentemente si ottengono i seguenti risultati:

Aggiunta Recensione:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Recensione | 1 | S |
| Utente | 1 | L |
| Utente | 1 | S |

(2\*2 + 1) \* 80,000 = 400,000/G

Aggiunta Commento:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Commento | 1 | S |
| Utente | 1 | L |
| Utente | 1 | S |

(2\*2 + 1) \* 120,000 = 650,000/G

Azzeramento Contatore:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Utente | 400,000 | S |

400,000\*2 \* 24 = 19,200,000/G

Si ha quindi un totale di 20,250,000/G.

Il numero di accessi di scrittura in Utente per l’azzeramento del contatore è pari a 400,000 ovvero pari al numero di utenti, e in generale aumenta con il passare del tempo, in un ambiente in continua crescita come quello di un social network.

Per quanto riguarda il caso in cui non si utilizzi il dato derivato, ad ogni aggiunta o rimozione occorre andare a contare tutte le pubblicazioni che ha effettuato l’utente (quindi recensioni + commenti), perciò:

Aggiunta Recensione:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Recensione | 1 | S |
| Recensione | 1,000,000/400,000 | L |
| Commento | 1,450,000/400,000 | L |

(1\*2 + 1,000,000/400,000 + 1,450,000/400,000) \* 80,000 = 650,000/G

Aggiunta Commento:

|  |  |  |
| --- | --- | --- |
| Concetto | Accessi | Tipo |
| Commento | 1 | S |
| Recensione | 1,000,000/400,000 | L |
| Commento | 1,450,000/400,000 | L |

(1\*2 + 1,000,000/400,000 + 1,450,000/400,000) \* 80,000 = 650,000/G

Si ha quindi un totale di 1,300,000/G.

In quest’ultimo caso, in generale nelle due operazioni di inserimento, la prima operazione di scrittura viene in realtà omessa nel caso molto raro in cui l’utente abbia effettuato già 1,000 pubblicazioni nell’attuale ora. Le operazioni di lettura, che hanno lo scopo di contare il numero di recensioni e di commenti, vengono effettuate anche per ricavare l’ora di pubblicazione. Se le recensioni pubblicate dall’utente che hanno ora di pubblicazione maggiore di hh:00, dove hh è l’ora corrente, sono in numero maggiori di 1,000 allora a quel punto si impedisce la pubblicazione e di conseguenza la scrittura su Commento/Recensione.

In questo caso specifico si può osservare che il dato derivato semplifica le due operazioni di inserimento, tuttavia il costo di mantenimento dello stesso sorpassa notevolmente il risparmio che esso porta. Nel caso in cui il rapporto tra il volume dei commenti o delle recensioni e il volume degli utenti sia molto maggiore, si può ipotizzare che l’analisi possa portare a risultati differenti.

Lo schema logico complessivo è il seguente:

Moderatore(Email, Username, Nome, Cognome, Password)

PK: Username

AK: Email

Utente(Email, Username, Nome, Cognome, Password, Foto, IP)

PK: Username

AK: Email

UtentePremium(CodUtente, IBAN)

PK: CodUtente

FK: CodUtente REFERENCES Utente

Amicizia(Seguito, Seguente, Data)

PK: (Seguito, Seguente)

FK: Seguito REFERENCES Utente

FK: Seguente REFERENCES Utente

Oggetto(CodO, Nome, Descrizione, CodC, Categoria)

PK: CodO

Recensione(CodR, Foto, DataVisionePubblica, Titolo, Valore,

Descrizione, DataPubblicazione, CodO, CodUtente)

PK: CodR

FK: CodO REFERENCES Oggetto

FK: CodUtente REFERENCES Utente

Medaglia(CodUtente, CodR, Timestamp)

PK: (CodUtente, CodR)

FK: CodUtente REFERENCES Utente

FK: CodR REFERENCES Recensione

Report(CodUtente, CodR, Tipo, Altro)

PK: (CodUtente, CodR)

FK: CodUtente REFERENCES Utente

FK: CodR REFERENCES Recensione

Commento(CodC, CodR, CodU, Timestamp, Testo, CodCRisposta)

PK: (CodC, CodR)

FK: CodR REFERENCES Recensione

FK: CodU REFERENCES Utente

FK: (CodCRisposta, CodR) REFERENCES Commento

Messaggio(CodM, CodMittente, CodDestinatario, Letto, Timestamp)

PK: (CodM, CodMittente, CodDestinatario)

FK: CodMittente REFERENCES Utente

FK: CodDestinatario REFERENCES Utente

MTesto(CodM, CodMittente, CodDestinatario, Contenuto)

PK: (CodM, CodMittente, CodDestinatario)

FK: (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio

MImmagine(CodM, CodMittente, CodDestinatario, Immagine, Descrizione)

PK: (CodM, CodMittente, CodDestinatario)

FK: (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio

MRecensione(CodM, CodMittente, CodDestinatario, CodR)

PK: (CodM, CodMittente, CodDestinatario)

FK: (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio

FK: CodR REFERENCES Recensione

CartaCredito(Numero, DScadenza, EnteEmittente)

PK: Numero

CartaUtente(CodCarta, CodU)

PK: (CodCarta, CodU)

FK: CodU REFERENCES Utente

FK: CodCarta REFERENCES Carta

Piano(CodP, Quantita, Periodo, Attivo, CodUtentePremium)

PK: CodP

FK: CodUtentePremium REFERENCES UtentePremium

Esclusivita(DAnticipata, CodP, CodR)

PK: (CodP, CodR)

FK: CodP REFERENCES Piano

FK: CodR REFERENCES Recensione

Iscrizione(CodP, CodUtente, DIscrizione, DAbbandono)

PK: (CodP, CodUtente, DIscrizione)

FK: CodUtente REFERENCES Utente

FK: CodP REFERENCES Piano

TransazioneAutomatica(CRO, Annullata, Data, CodPiano, CodUtente,

NumeroCartaDiCredito)

PK: CRO

FK: CodPiano REFERENCES Piano

FK: CodUtente REFERENCES Utente

FK: NumeroCartaDiCredito REFERENCES CartaCredito

AK: (Data, CodPiano, CodUtente)

TransazioneManuale(CRO, Annullata, Data, CodMittente, CodDestinatario,

NumeroCartaDiCredito)

PK: CRO

FK: CodMittente REFERENCES Utente

FK: CodDestinatario REFERENCES Utente

FK: NumeroCartaDiCredito REFERENCES CartaCredito

AK: (Data, CodMittente, CodDestinatario)

Ban(CodUtente, CodModeratore)

PK: CodUtente

FK: CodUtente REFERENCES Utente

FK: CodModeratore REFERENCES Moderatore

Rimozione(DataEffettuazione, CodR, DAnnullamento, CodModeratore)

PK: (DataEffettuazione, CodR)

FK: CodR REFERENCES Recensione

FK: CodModeratore REFERENCES Moderatore

**Operazioni previste dalla base di dati – Descrizione e relativo codice SQL**

Vi sono alcuni vincoli che vengono gestiti con appositi CHECK, in particolare: il formato delle email, degli username e degli indirizzi IP; il vincolo del valore della recensione che deve essere compreso nell’intervallo 0-100; la data di annullamento di una rimozione e la data di abbandono di un piano, che devono essere rispettivamente maggiore della data di effettuazione della rimozione e della data di iscrizione a tale piano; infine l’esclusività del tipo di un report e del campo “Altro” (la presenza di uno esclude la presenza dell’altro). I rimanenti verranno gestiti con appositi TRIGGER.

**Query di creazione**

Si hanno le seguenti query di inserimento:

CREATE TABLE Moderatore(

Email varchar(320) UNIQUE,

CHECK(Email ~\* '^[A-Za-z0-9.\_%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'),

Username varchar(55) PRIMARY KEY,

CHECK(Username NOT LIKE '%[^a-zA-Z0-9]%'),

Nome varchar(55) NOT NULL,

Cognome varchar(55) NOT NULL,

Psw char(64) NOT NULL

);

CREATE TABLE Utente(

Email varchar(320) UNIQUE,

Username varchar(55) PRIMARY KEY,

Nome varchar(55) NOT NULL,

Cognome varchar(55) NOT NULL,

IP char(15) NOT NULL,

CHECK(IP LIKE '###.###.###.###'),

Psw char(64) NOT NULL,

Foto varchar(55)

);

CREATE TABLE UtentePremium(

CodUtente varchar(55) PRIMARY KEY,

FOREIGN KEY (CodUtente) REFERENCES Utente(Username),

IBAN char(27) NOT NULL

);

CREATE TABLE Amicizia(

Seguente varchar(55),

Seguito varchar(55),

FOREIGN KEY (Seguente) REFERENCES Utente(Username),

FOREIGN KEY (Seguito) REFERENCES Utente(Username),

PRIMARY KEY (Seguente, Seguito),

Data timestamp NOT NULL

);

CREATE TABLE Oggetto(

CodO int PRIMARY KEY,

Nome varchar(55) NOT NULL,

Descrizione varchar(320) NOT NULL,

Categoria varchar(55) NOT NULL

);

CREATE TABLE Recensione(

CodR int PRIMARY KEY,

Foto varchar(55),

DataPubblicazione date NOT NULL,

DataVisionePubblica date,

Titolo varchar(155) NOT NULL,

Valore int,

Descrizione varchar(1555) NOT NULL,

CHECK(Valore IS NULL OR (Valore >= 0 AND Valore <= 100)),

CodUtente varchar(55) NOT NULL,

CodO int NOT NULL,

FOREIGN KEY (CodO) REFERENCES Oggetto(CodO),

FOREIGN KEY (CodUtente) REFERENCES Utente(Username)

);

CREATE TABLE Medaglia(

CodUtente varchar(55),

CodR int,

Data timestamp NOT NULL,

FOREIGN KEY (CodUtente) REFERENCES Utente(Username),

FOREIGN KEY (CodR) REFERENCES Recensione(CodR),

PRIMARY KEY (CodUtente, CodR)

);

CREATE TYPE tipoR AS ENUM('Contenuti di natura sessuale', 'Contenuti violenti o ripugnanti', 'Azioni dannose o pericolose', 'Spam o ingannevole');

CREATE TABLE Report(

CodUtente varchar(55),

CodR int,

TipoReport tipoR,

Altro varchar(55),

CHECK((TipoReport IS NULL AND Altro IS NOT NULL) OR (TipoReport IS NOT NULL AND Altro IS NULL)),

FOREIGN KEY (CodUtente) REFERENCES Utente(Username),

FOREIGN KEY (CodR) REFERENCES Recensione(CodR),

PRIMARY KEY (CodUtente, CodR)

);

CREATE TABLE Commento(

CodC int,

CodR int,

CodUtente varchar(55) NOT NULL,

Data timestamp NOT NULL,

Testo varchar(1555),

CodCRisposta int,

FOREIGN KEY (CodUtente) REFERENCES Utente(Username),

FOREIGN KEY (CodR) REFERENCES Recensione(CodR),

FOREIGN KEY (CodCRisposta, CodR) REFERENCES Commento,

PRIMARY KEY (CodC, CodR)

);

CREATE TABLE Messaggio(

CodM int,

CodMittente varchar(55),

CodDestinatario varchar(55),

Letto boolean NOT NULL,

Data timestamp NOT NULL,

FOREIGN KEY (CodDestinatario) REFERENCES Utente(Username),

FOREIGN KEY (CodMittente) REFERENCES Utente(Username),

PRIMARY KEY (CodM, CodMittente, CodDestinatario)

);

CREATE TABLE MTesto(

CodM int,

CodMittente varchar(55),

CodDestinatario varchar(55),

Contenuto varchar(1555),

FOREIGN KEY (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio(CodM, CodMittente, CodDestinatario),

PRIMARY KEY (CodM, CodMittente, CodDestinatario)

);

CREATE TABLE MImmagine(

CodM int,

CodMittente varchar(55),

CodDestinatario varchar(55),

Immagine varchar(55),

Descrizione varchar(1555),

FOREIGN KEY (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio(CodM, CodMittente, CodDestinatario),

PRIMARY KEY (CodM, CodMittente, CodDestinatario)

);

CREATE TABLE MRecensione(

CodM int,

CodMittente varchar(55),

CodDestinatario varchar(55),

codR int NOT NULL,

Descrizione varchar(1555),

FOREIGN KEY (CodM, CodMittente, CodDestinatario) REFERENCES Messaggio(CodM, CodMittente, CodDestinatario),

FOREIGN KEY (CodR) REFERENCES Recensione(CodR),

PRIMARY KEY (CodM, CodMittente, CodDestinatario)

);

CREATE TABLE CartaCredito(

Numero char(16) PRIMARY KEY,

DScadenza char(5) NOT NULL,

EnteEmittente varchar(30) NOT NULL

);

CREATE TABLE CartaUtente(

CodU varchar(55),

NumeroC char(16),

PRIMARY KEY(CodU, NumeroC),

FOREIGN KEY (CodU) REFERENCES Utente(Username),

FOREIGN KEY (NumeroC) REFERENCES CartaCredito(Numero)

);

CREATE TYPE periodoPiano AS ENUM('settimana', 'mese', 'trimestre', 'semestre', 'anno');

CREATE TABLE Piano(

CodP int PRIMARY KEY,

CodR int NOT NULL,

CodUtentePremium varchar(55) NOT NULL,

Quantita int NOT NULL,

Periodo periodoPiano NOT NULL,

Attivo bool NOT NULL,

FOREIGN KEY (CodUtentePremium) REFERENCES UtentePremium(CodUtente),

FOREIGN KEY (CodR) REFERENCES Recensione(CodR)

);

CREATE TABLE Esclusivita(

DataAnticipata date NOT NULL,

CodP int,

CodR int,

FOREIGN KEY (CodR) REFERENCES Recensione(CodR),

FOREIGN KEY (CodP) REFERENCES Piano(CodP),

PRIMARY KEY (CodP, CodR)

);

CREATE TABLE Iscrizione(

CodP int,

CodUtente varchar(55),

DIscrizione date,

DAbbandono date,

CHECK(DAbbandono IS NULL OR (DAbbandono >= DIscrizione)),

FOREIGN KEY (CodUtente) REFERENCES Utente(Username),

FOREIGN KEY (CodP) REFERENCES Piano(CodP),

PRIMARY KEY (CodP, CodUtente, DIscrizione)

);

CREATE TABLE TransazioneAutomatica(

TRN char(30) PRIMARY KEY,

Annullata bool NOT NULL,

Data date NOT NULL,

CodPiano int NOT NULL,

CodUtente varchar(55) NOT NULL,

NumeroCartaDiCredito char(16) NOT NULL,

FOREIGN KEY (NumeroCartaDiCredito) REFERENCES CartaCredito(Numero),

FOREIGN KEY (CodPiano) REFERENCES Piano(CodP),

FOREIGN KEY (CodUtente) REFERENCES Utente(Username),

UNIQUE(Data, CodPiano, CodUtente)

);

CREATE TABLE TransazioneManuale(

TRN char(30) PRIMARY KEY,

Annullata bool NOT NULL,

Data date NOT NULL,

CodMittente varchar(55) NOT NULL,

CodDestinatario varchar(55) NOT NULL,

NumeroCartaDiCredito char(16) NOT NULL,

FOREIGN KEY (NumeroCartaDiCredito) REFERENCES CartaCredito(Numero),

FOREIGN KEY (CodMittente) REFERENCES Utente(Username),

FOREIGN KEY (CodDestinatario) REFERENCES Utente(Username),

UNIQUE(Data, CodMittente, CodDestinatario)

);

CREATE TABLE Ban(

CodUtente varchar(55) PRIMARY KEY,

CodModeratore varchar(55) NOT NULL,

FOREIGN KEY (CodUtente) REFERENCES Utente(Username),

FOREIGN KEY (CodModeratore) REFERENCES Utente(Username)

);

CREATE TABLE Rimozione(

DataEffettuazione date,

CodR int,

DAnnullamento date,

CHECK(DAnnullamento IS NULL OR (DataEffettuazione <= DAnnullamento)),

CodModeratore varchar(55) NOT NULL,

PRIMARY KEY(DataEffettuazione, CodR),

FOREIGN KEY (CodR) REFERENCES Recensione(CodR),

FOREIGN KEY (CodModeratore) REFERENCES Utente(Username)

);

**Viste**

Vista contenente tutti i piani attivi:

CREATE VIEW PianiAttivi AS

SELECT \*

FROM Piano

WHERE Attivo = TRUE

Vista contenente tutte le recensioni visibili:

CREATE VIEW RecensioniVisibili AS

SELECT R1.\*

FROM Recensione AS R1

WHERE R1.CodR NOT IN (

SELECT R2.CodR

FROM Recensione AS R2, Rimozione

WHERE R2.CodR = Rimozione.CodR

AND Rimozione.DAnnullamento IS NULL

)

Vista contenente tutte le recensioni disponibili per ciascun utente:

CREATE VIEW RecDisponibiliPerUtente AS

SELECT Utente.Username, R.CodR

FROM Utente, RecensioniVisibili AS R, UtentePremium

WHERE R.DataVisionePubblica <= CURRENT\_DATE OR

(UtentePremium.CodUtente = R.CodUtente AND

EXISTS (

SELECT \*

FROM Iscrizione, Piano, Esclusivita

WHERE Piano.CodUtentePremium = UtentePremium.CodUtente AND

Iscrizione.CodUtente = Utente.Username AND

Iscrizione.CodP = Piano.CodP AND

Iscrizione.DAbbandono IS NULL AND

Piano.CodP = Esclusivita.CodP AND

Esclusivita.CodR = R.CodR AND

DataAnticipata <= CURRENT\_DATE

))

**Trigger – Imposizione di vincoli [da testare, e volendo si può aggiungere una piccola descrizione]**

Totalità della gerarchia di Transazione:

/\* Il primo argomento è vero se si tenta di inserire una transazione manuale, falso altrimenti \*/

CREATE FUNCTION TotalitaGerarchiaTrans() RETURNS TRIGGER AS $$

DECLARE

Cond BOOL;

BEGIN

IF OLD.TRN = NEW.TRN THEN RETURN NEW;

END IF;

IF TG\_NARGS <> 1 THEN RAISE EXCEPTION 'Argomenti forniti erroneamente';

END IF;

IF TG\_ARGV[0] = TRUE

THEN

Cond = EXISTS(SELECT TRN

FROM TransazioneAutomatica

WHERE TRN = NEW.TRN);

ELSE

Cond = EXISTS(SELECT TRN

FROM TransazioneManuale

WHERE TRN = NEW.TRN);

END IF;

IF Cond

THEN RAISE EXCEPTION 'Questo codice TRN è già presente nel database.';

END IF;

RETURN NEW;

END;

$$ LANGUAGE 'plpgsql';

CREATE TRIGGER GestioneInserTransManuale

AFTER INSERT OR UPDATE ON TransazioneManuale

FOR EACH ROW EXECUTE PROCEDURE TotalitaGerarchiaTrans(TRUE);

CREATE TRIGGER GestioneInserTransAuto

AFTER INSERT OR UPDATE ON TransazioneAutomatica

FOR EACH ROW EXECUTE PROCEDURE TotalitaGerarchiaTrans(FALSE);

Totalità della gerarchia di Messaggio:

CREATE FUNCTION TotalitaGerarchiaMessaggio() RETURNS TRIGGER AS $$

BEGIN

IF (OLD.CodM = NEW.CodM AND

OLD.CodMittente = NEW.CodMittente AND

OLD.CodDestinatario = NEW.CodDestinatario) THEN RETURN NEW;

END IF;

IF (SELECT COUNT(\*)

FROM (SELECT CodM, CodMittente, CodDestinatario

FROM MTesto

UNION

SELECT CodM, CodMittente, CodDestinatario

FROM MImmagine

UNION

SELECT CodM, CodMittente, CodDestinatario

FROM MRecensione) AS CodMessaggi

WHERE CodM = NEW.CodM

AND CodMittente = NEW.CodMittente

AND CodDestinatario = NEW.CodDestinatario) = 1

THEN RAISE EXCEPTION 'La tripla (CodM, CodMittente, CodDestinatario) esiste

già in un''altra specializzazione di Messaggio';

END IF;

RETURN NEW;

END;

$$ LANGUAGE 'plpgsql';

CREATE TRIGGER GestioneInserMTesto

AFTER INSERT OR UPDATE ON MTesto

FOR EACH ROW EXECUTE PROCEDURE TotalitaGerarchiaMessaggio();

CREATE TRIGGER GestioneInserMImmagine

AFTER INSERT OR UPDATE ON MImmagine

FOR EACH ROW EXECUTE PROCEDURE TotalitaGerarchiaMessaggio();

CREATE TRIGGER GestioneInserMRecensione

AFTER INSERT OR UPDATE ON MRecensione

FOR EACH ROW EXECUTE PROCEDURE TotalitaGerarchiaMessaggio();

Vincolo esclusività:

CREATE FUNCTION VincoloEsclusivita() RETURNS TRIGGER AS $$

DECLARE

CodUtenteP varchar(55);

BEGIN

IF (OLD.CodP = NEW.CodP AND

OLD.CodR = NEW.CodR) THEN RETURN NEW;

END IF;

SELECT CodUtentePremium INTO CodUtenteP

FROM Piano WHERE CodP = NEW.CodP;

IF NOT EXISTS (SELECT \*

FROM Recensione

WHERE CodR = NEW.CodR AND CodUtente = CodUtenteP)

THEN RAISE EXCEPTION 'La tripla (CodM, CodMittente, CodDestinatario) esiste

già in un''altra specializzazione di Messaggio';

END IF;

RETURN NEW;

END;

$$ LANGUAGE 'plpgsql';

CREATE TRIGGER GestioneEsclusivita

AFTER INSERT OR UPDATE ON Esclusivita

FOR EACH ROW EXECUTE PROCEDURE VincoloEsclusivita();

Vincolo carta di credito:

CREATE FUNCTION VincoloCartaCredito() RETURNS TRIGGER AS $$

BEGIN

IF (OLD.CodPiano = NEW.CodPiano AND

OLD.CodUtente = NEW.CodUtente) THEN RETURN NEW;

END IF;

IF NOT EXISTS (SELECT \*

FROM CartaUtente

WHERE CodU = NEW.CodUtente AND NumeroC = NEW.NumeroCartaDiCredito)

THEN RAISE EXCEPTION 'L''utente non può utilizzare la carta di credito in questione';

END IF;

RETURN NEW;

END;

$$ LANGUAGE 'plpgsql';

CREATE TRIGGER GestioneTransManCarta

AFTER INSERT OR UPDATE ON TransazioneManuale

FOR EACH ROW EXECUTE PROCEDURE VincoloCartaCredito();

CREATE TRIGGER GestioneTransAutoCarta

AFTER INSERT OR UPDATE ON TransazioneAutomatica

FOR EACH ROW EXECUTE PROCEDURE VincoloCartaCredito();

Vincolo rimozione:

CREATE FUNCTION VincoloRimozione() RETURNS TRIGGER AS $$

BEGIN

IF OLD.DAnnullamento = NEW.DAnnullamento THEN RETURN NEW;

END IF;

IF EXISTS (SELECT \*

FROM Rimozione

WHERE CodR = NEW.CodR AND DAnnullamento IS NULL)

THEN RAISE EXCEPTION 'La recensione è già attualmente invisibile';

END IF;

RETURN NEW;

END;

$$ LANGUAGE 'plpgsql';

CREATE TRIGGER GestioneRimozione

AFTER INSERT OR UPDATE ON Rimozione

FOR EACH ROW EXECUTE PROCEDURE VincoloRimozione();

Vincolo iscrizione:

CREATE FUNCTION VincoloIscrizione() RETURNS TRIGGER AS $$

BEGIN

IF OLD.DAbbandono = NEW.DAbbandono THEN RETURN NEW;

END IF;

IF EXISTS (SELECT \*

FROM Iscrizione

WHERE CodP = NEW.CodP AND

CodUtente = NEW.CodUtente AND

DAbbandono IS NULL)

THEN RAISE EXCEPTION 'L''utente è già attualmente iscritto al piano';

END IF;

RETURN NEW;

END;

$$ LANGUAGE 'plpgsql';

CREATE TRIGGER GestioneIscrizione

AFTER INSERT OR UPDATE ON Iscrizione

FOR EACH ROW EXECUTE PROCEDURE VincoloIscrizione();

**Trigger – Mantenimento dati derivati**

Non sono presenti dati derivati all’interno della base di dati in quanto lo studio dei dati derivati ha portato alla conclusione che entrambi quelli considerati sono svantaggiosi.

**Procedure**

Procedure da fare: Inserimento/Cancellazione di una recensione/commento, forse la roba degli inserimenti/elevazione die privilegi

**Query di inserimento**

Mancano.

**Query di interrogazione**

Mancano.

**Query di modifica**

Mancano.

**Query di eliminazione**

Mancano.

**Progettazione Fisica**

Si vuole effettuare l’analisi del costo delle query, per comprendere se conviene l’utilizzo o non di un indice. In seguito vengono riportati alcuni esempi di casi di studio.

Un primo caso di studio puó essere rappresentato dalla seguente query: dato un utente, visualizzarne gli abbonamenti attivi.

**SELECT**

**Iscrizione.DIscrizione,**

**Piano.CodUtentePremium,**

**Piano.Periodo,**

**Piano.Quantita**

**FROM**

**Iscrizione, Piano**

**WHERE**

**Iscrizione.CodP = Piano.CodP AND**

**Iscrizione.CodUtente = 'jimmy' AND**

**Iscrizione.DAbbandono = NULL.**

Si supponga di avere i seguenti dati relativi alle tabelle

|  |  |  |
| --- | --- | --- |
| **Tabella** | **NT** | **NB** |
| **Iscrizione** | **100000** | **10000** |
| **Piano** | **20000** | **1000** |

Si vuole studiare i seguenti indici e si considera che siano tutti unclustered.

|  |  |  |
| --- | --- | --- |
| **Indice** | **NF** | **NK** |
| **Iscrizione.CodP** | **80** | **20000** |
| **Iscrizione.CodUtente** | **50** | **6000** |
| **Piano.CodP** | **60** | **20000** |

Si effettua l’analisi della selettivitá degli indici e degli predicati rimanenti

Fiscrizione.CodP = 1/20000

Fiscrizione.CodUtente = 1/6000

Fpiano.CodP = 1/20000

Fpiano.Dabbondono = 1/10

Siccome all’interno della query è presente un join occorre suddividere l’analisi del costo in due casi, andando ad analizzare come varia il costo della query al variare della direzione di join.

Caso 1: Iscrizione -> Piano

Iscrizione

Cseq =10000

Ciscrizione.CodUtente = ceil(50/6000) + ceil(10000/6000) = 3

Eiscrizione = ceil(100000/(50 \* 6000) = 4

Piano

Cseq = 1000

Cpiano.CodP = ceil(60/20000) + ceil(20000/20000) = 2

CostTot = 3 + 4 \* 2 = 11

Caso 2: Piano -> Iscrizione

Piano

Cseq = 1000

Epiano = ceil(20000 / 10) = 2000

Iscrizione

Cseq = 10000

Ciscrizione.CodUtente = 3

Ciscrizione.CodP = ceil(80/20000) + ceil(100000/20000) = 7

CostTot = 1000 + 2000 \* 3 = 7000

In questo primo caso di studio possiamo concludere che per velocizzare la query occorre considerare il caso Iscrizione -> Piano e creare gli indici su Ciscrizione.CodUtente e Cpiano.CodP.

Un secondo caso di studio puó essere rappresentato dalla query che dato un utente, ritorna tutti i messaggi che non ha letto.

**SELECT CodMittente**

**FROM Messaggio**

**WHERE CodDestinatario = 'sara' AND Letto = 'FALSE'**

Si supponga di avere i seguenti dati relativi alle tabella Messaggio.

|  |  |  |
| --- | --- | --- |
| **Tabella** | **NT** | **NB** |
| Messaggio | 1000000 | 40000 |

Si vuole studiare il seguente indice unclustered.

|  |  |  |
| --- | --- | --- |
| **Indice** | **NF** | **NK** |
| CodDestinatario | 150 | 20000 |

Si considera che il predicato **Letto =** **‘FALSE’** abbia una selettività pari a 1/10. Dall’analisi della selettività dell’indice si ottiene che FCodDestinatario è pari a 1/20000.

Si puó procedere con l’analisi del costo della query.

Cseq = 40000

CCodDestinatario = ceil(150/20000) + ceil(1000000/20000) = 51

E = ceil(100000/(20000\*10) = 5

Infine. si puó concludere che per questo secondo caso di studio conviene la creazione dell’indice CodDestinatario e il numero di tuple attese è pari a 5.

**Analisi della sicurezza**

Un’aspetto importante della progettazione fisica è riguardante l’analisi della sicurezza del database in questione e lo studio dei permessi da attribuire ad un utente che non sia amministratore.

L’elemento critico del sistema informativo che si vuole studiare è l’autenticazione dell’utente, che permette di fruire dei servizi offerti dal social network. Nel database in questione vengono memorizzati inoltre alcune informazioni sensibili riguardanti all’utente, ad esempio la carta di credito. Tali informazioni possono essere rese inaccessibili se si progetta un adeguato meccanismo di autenticazione al sistema.

Per l’autenticazione di un utente, occorre memorizzare nel sistema le credenziali di accesso, che sono l’username e la password. Quest’ultima non puó essere memorizzata in chiaro nel database, in quanto permetterebbe l’accesso all’account di un utente nel caso in cui vi sia una fuga di dati. Il metodo che si vuole adottare è quello di poter cifrare la password e memorizzarne nel database la versione cifrata.

Per poter utiilizzare le funzionalitá di cifratura fornite dal DBMS POSTGRESQL, occorre abilitare l’estensione pgcrypto.

CREATE EXTENSION IF NOT EXISTS pgcrypto;

A fronte delle analisi effettuate si è deciso di creare un trigger, che permettesse di sostituire la password con la sua versione cifrata a seguito dell’aggiornamento delle credenziali memorizzate nella tabella ‘utente’. È stata prevista inoltre una procedura che permettesse di convalidare le credenziali di un utente. Le precedenti funzionalitá sono state implementate anche per il moderatore.

CREATE FUNCTION moderatore\_login(IN user VARCHAR(55), IN pswd VARCHAR(64)) RETURNS BOOLEAN AS $$

DECLARE

successful BOOLEAN

BEGIN

SELECT (psw = crypt(pswd, psw)) INTO successful

FROM moderatore

WHERE username = user;

RETURN successful;

END;

$$ LANGUAGE 'plpgsql';

CREATE FUNCTION user\_login(IN user VARCHAR(55), IN pswd VARCHAR(64)) RETURNS BOOLEAN AS $$

DECLARE

successful BOOLEAN

BEGIN

SELECT (psw = crypt(pswd, psw)) INTO successful

FROM utente

WHERE username = user;

RETURN successful;

END;

$$ LANGUAGE 'plpgsql';

CREATE FUNCTION GestionePassword() RETURNS TRIGGER AS $$

BEGIN

IF OLD.psw = NEW.psw THEN RETURN NEW;

END IF;

NEW.psw = crypt(pswd, gen\_salt('bf'));

RETURN NEW;

END;

$$ LANGUAGE 'plpgsql';

CREATE TRIGGER GestionePasswordUtente

BEFORE INSERT OR UPDATE ON utente

FOR EACH ROW EXECUTE PROCEDURE GestionePassword();

CREATE TRIGGER GestionePasswordModeratore

BEFORE INSERT OR UPDATE ON moderatore

FOR EACH ROW EXECUTE PROCEDURE GestionePassword();

Effettuando un analisi piú approndita, sono state individuate delle criticitá nelle operazioni di eliminazione di un utente, e di modifica di password. Tali operazioni devono essere protette da un meccanismo di un autenticazione, per la verifica dell’identitá di un utente.

Innanzitutto occorre creare un nuovo utente del database, che a differenza dell’amministratore sará in possesso di permessi diversi.

CREATE USER client WITH ENCRYPTED PASSWORD 'topolino2022';

A tale utente si decide di assegnare i permessi dI SELECT, INSERT, UPDATE sulla tabella utente. E in particolare sulle colonne ‘username’ e ‘psw’ il nuovo utente avrá solo i permessi di SELECT, INSERT. In questo modo si impedisce l’eliminazione di un utente oppure il cambiamento di una password. Tali operazioni potranno essere eseguite tramite procedure, che le quali prima di eseguire le operazioni in questione effettuano un controllo di autenticazione. In quanto l’utente del database ‘client’ non puó effettuare l’eliminazione di una tupla della tabella utente oppure aggiornare la password, tali procedure devono poter essere eseguite con i permessi del utente che le ha create(ció si ottiene mediante la direttiva SECURITY DEFINER).

GRANT SELECT, INSERT, UPDATE ON utente to client;

GRANT SELECT(email, username, nome, cognome, ip, foto, psw),

INSERT(email, username, nome, cognome, ip, foto, psw),

UPDATE(email, nome, cognome, ip, foto) ON utente to client;

CREATE FUNCTION delete\_user(IN user VARCHAR(55), IN pswd VARCHAR(64)) RETURNS BOOLEAN SECURITY DEFINER AS $$

BEGIN

IF NOT (SELECT user\_login(user, pswd)) = TRUE

THEN

RETURN FALSE;

END IF;

DELETE FROM utente WHERE username = user;

RETURN TRUE;

END

$$ LANGUAGE 'plpgsql';

CREATE FUNCTION change\_psw(IN user VARCHAR(55), IN new\_pswd VARCHAR(64), IN old\_pswd VARCHAR(64))

RETURNS BOOLEAN SECURITY DEFINER AS $$

BEGIN

IF NOT (SELECT user\_login(user, old\_pswd)) = TRUE

THEN

RETURN FALSE;

END IF;

UPDATE utente SET psw = new\_pswd WHERE psw = crypt(old\_pswd, psw);

RETURN TRUE;

END

$$ LANGUAGE 'plpgsql';

Il moderatore é una figura che controlla le varie interazioni tra i vari utenti per poter preservare la sicurezza della piattaforma e la pubblicazione di contenuti coerenti con le policy del social network. L’inserimento di un moderatore all’interno del social network, e in questo caso nel database, deve poter essere effettuato solo dall’utente amministratore, in quanto i moderatori sono delle persone su cui si fa affidamento per far rispettare le regole del social network, e inoltre deve poter avere accesso a tutte le segnalazioni e recensioni pubblicate nel database. Per queste ragioni occorre regolamentare la creazione degli utenti moderatori, in quanto l’account moderatore puó essere usato in modo non convenzionale. Perció l’utente ‘client’ avrá solo il permesso di SELECT sulla tabella moderatore.

GRANT SELECT ON moderatore to client;

L’utente ‘client’ sará l’utente che verrá utilizzato dall’applicazione o sito web dell’applicazione per connettersi al database.