Julian Sparber

Università di Urbino “Carlo Bo”

Corso di Laurea in Informatica Applicata

Progetto di Basi di Dati

2014/15

Piattaforma di Aste-on-line

Codice sorgente: https://github.com/jsparber/database

Demo: http://asta.sparber.net.

Indice

1 Analisi dei requisiti

L' obbiettivo del progetto e` di realizzare una piattaforma che gestisce aste online dove utenti privati e professionali possono vendere e acquistare dei beni.

Gli oggetti principali della piattaforma sono gli utenti e i beni.

1.1 Utenti

Un utente può essere sia un venditore che un acquirente, quindi ogni utente può avere sia beni acquistati sia beni venduti.

Un utente ha in oltre la sua fascia. Per semplicità mettiamo gli utenti in 4 categorie privato, silver, gold, platinum.

Esiste un sistema di feedback per gli utenti.

1.2 Beni

Ogni bene ha la sua categoria già' inserita nel sistema.

Il venditore può associare un immagine a suoi prodotti in vendita.

Ogni prodotto può avere delle sottocategorie che vengono create dai venditori.

Ogni prodotto e' associato ad uno o più metodi di pagamento e spedizione.

Le merci possono essere comprati attraverso la vendita diretta (compra subito) oppure attraverso un' asta preimpostata dal venditore.

Le aste hanno una data di scadenza, un prezzo di partenza e opzionalmente un prezzo di riserva da raggiungere.

1.3 Categorie

Ogni vendita prevede una provvigione (in percentuale dell'importo) sulla base della categoria.

Le categorie hanno una provvigione associata.

2 Glossario dei termini e dei concetti fondamentali

2.1 Utente

Qualsiasi persona fisica o giuridica registrata sulla piattaforma.

Sinonimi: venditore, acquirente, user;

2.2 Prodotto

Tutti oggetto che l'utente offre o che può comprare.

Sinonimi: merce, oggetto, bene

2.3 Categoria

Gruppo di prodotti che hanno aspetti comuni.

Sinonimi: gruppo, classe, tipologia

2.4 Sottocategoria

Le parole chiave per la ricerca di un prodotto. A ogni prodotto viene assegnata almeno una sottocategoria.

Sinonimi: parole chiave, tag

2.5 Asta

Vendita pubblica mediante una gara con assegnazione al miglior offerente

Sinonimi: auction

2.6 Compra subito

Vendita pubblica con un prezzo fisso

Sinonimi: vendita diretta

4 Progettazione concettuale

Osservando attentamente le informazioni citate sopra, si possono ora individuare I componenti principali del database in questione.

Entità

Definizione: L’entità è un oggetto (concreto o astratto) che ha un significato anche quando viene considerato in modo isolato, ed è di interesse per la realtà che si vuole modellare.

4.1 Utente

Attributi

E-mail → identifica univocamente l'utente e indirizzo email dove è possibile contattare direttamente l'utente

Nome → nome dell'utente.

Cognome → cognome dell'utente.

Residenza → residenza dell'utente che viene utilizzato come indirizzo di fatturazione

IndirizzoSpedizione → indirizzo a quale vengono spediti le merci

Relazioni

Un utente può effettuare n offerte.

Un utente può effettuare n acquisti.

Un utente può mettere n prodotti in vendita.

Spiegazione

Ciascuna persona che desidera usufruire del sito deve possedere un profilo, visualizzabile anche dagli altri utenti. Un utente può mettere in vendita oggetti o acquistare gli oggetti offerti dagli altri utenti.

Semplificazione

Si potrebbe pensare di creare entità diverse per gli utenti offerenti, acquirenti e interessati in quanto concettualmente hanno obiettivi diversi. Nel nostro caso

però, gli utenti non sono di un solo tipo, in quanto ciascuno può offrire oggetti o acquistarlo di conseguenza la separazione della entità utente non sarebbe producente.

4.2 Credenziali

Attributi

idCredenziali → una id che identifica univocamente i Credenziali.

Utente → utente di riferimento.

Password → chiave segreta del utente.

Username → il nome utente.

Relazioni

Ogni credenziale è associato ad uno solo utente.

Spiegazione

I credenziali di un utente per accedere al suo conto.

4.3 Prodotto

Attributi

IdProdotto → un attributo auto-incrementante che identifica univocamente il prodotto.

Nome → nome del prodotto.

Descrizione→ descrizione del prodotto.

Foto → eventuale foto del prodotto

Categoria → la id della categoria del prodotto.

Stato → stato del prodotto.

Prezzo → il prezzo del prodotto.

Spedizione → le id dei metodi di spedizione

Pagamento → le id dei metodi di pagamento

Proprietario → l' utente che ha inserito questo prodotto.

Relazioni

Ogni prodotto ha una sola categoria.

Ogni prodotto ha un solo proprietario.

Spiegazioni

Lista dei prodotti inseriti dall'utente.

4.4 Categoria

Attributi

IdCategoria → una id che identifica univocamente la categoria.

Nome → nome della categoria.

Relazioni

Ogni categoria contiene n prodotti.

Spiegazione

Elenco delle categorie.

4.5 MetodoPagamento

Attributi

idMetodoPagamento → una id che identifica univocamente il metodo.

Nome → nome del metodo di Pagamento.

Relazioni

Ogni Metodo di Pagamento può avere n prodotti associati.

Spiegazione

Tutti I metodi di pagamento.

4.6 Stato

Attributi

idStato → una id che identifica univocamente il stato.

Nome → nome del stato.

Relazioni

Ogni stato può avere n prodotti associati.

Spiegazione

I stati di un prodotto.

4.7 Spedizione

Attributi

idSpedizone→ un attributo auto-incrementante che identifica univocamente la spedizione.

Nome → nome del corriere.

Descrizione –> descrizione del corriere.

TempoConsegna → il tempo di consegna

Importo → costo della spedizione.

Prodotto → il prodotto di riferimento

Relazioni

Ogni spedizione e' associato ad un solo prodotto.

Spiegazione

Elenco dei metodi di spedizione.

4.8 Pagamento

Attributi

Id → un attributo auto-incrementante che identifica univocamente la spedizione.

Nome → nome del metodo di pagamento.

Relazioni

Ogni pagamento più avere n prodotti.

Spiegazione

Elenco dei metodi di pagamento. Dall'utente non e' possibile di modificare I metodi di pagamento.

4.9 Offerte

Attributi

Id → un attributo auto-incrementante che identifica univocamente l' offerta.

Utente →l' utente che ha effettuato la offerta.

Importo → importo effettuato.

Data → la data e ora dell'offerta.

Prodotto → il prodotto di riferimento.

Relazioni

Una offerta viene effettuata da un singolo utente.

Una offerta è relativa ad un singolo oggetto.

Una offerta può avere più controfferte.

Spiegazione

Oggetto offerto da parte di un determinato utente.

4.10 Feedback

Attributi

Id → un attributo auto-incrementante che identifica univocamente il feedback.

Utente →l' utente di riferimento.

Autore → l' utente che ha scritto il feedback.

Contenuto → l' testo del feedback che ha scritto l' autore.

Valutazione → il voto dato dal autore.

Relazioni

Ogni feedback e' associato a un solo utente.

Ogni utente può scrivere n feedback.

Spiegazione

Il feedback scritto da un utente.

4.11 Asta

Attributi

IdAsta → un attributo auto-incrementante che identifica univocamente l'asta

Prodotto →il prodotto di riferimento.

Scadenza→ la scadenza della asta.

PrezzoPartenza → il prezzo di partenza.

PrezzoRiserva →il prezzo di riserva.

Relazioni

Ogni asta è associato a un solo prodotto.

Spiegazione

La lista dei prodotti in asta.

4.12 VenditaDiretta

Attributi

idVenditaDiretta → un attributo auto-incrementante che identifica univocamente la vendita diretta.

Prodotto → prodotto di riferimento.

Relazioni

Ogni vendita diretta è associato a un solo prodotto.

Spiegazione

La lista dei prodotti in vendita in modalità vendita diretta.

4.13 CronologiaVendite

Attributi

idCronologiaVendite → un attributo auto-incrementante che identifica univocamente vendita.

Prodotto → prodotto di riferimento.

Acquirente → l' utente che ha comprato il prodotto.

Relazioni

Ogni vendita è associato a un solo prodotto.

Spiegazione

La lista dei prodotti venduti.

5 Scelta della strategia di progetto

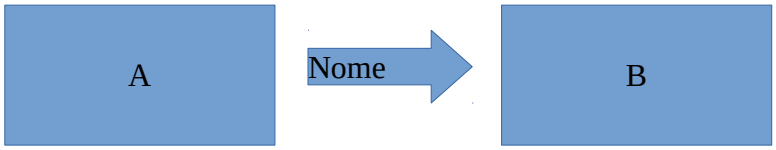
La strategia usata sopra per costruire la struttura di questo database si chiama “bottom-up”, cioè I concetti sono stati costruiti a partire dalle componenti elementari. In poche parole, ho individuato le diverse entità, elencato i loro attributi, poi ragionando, sono arrivato a trovare come questi interagiscono fra di loro, ossia ho definito le loro associazioni.

5.1 Sviluppo dello schema E/

Sfruttando le diverse informazioni descritte per ogni entità, ora sono in grado di sviluppare lo schema E/R (Entity/Relationship che in inglese significa entità/associazione), che è la rappresentazione concettuale del DB, esprimendo quindi la realtà dei dati e le relazioni tra essi.

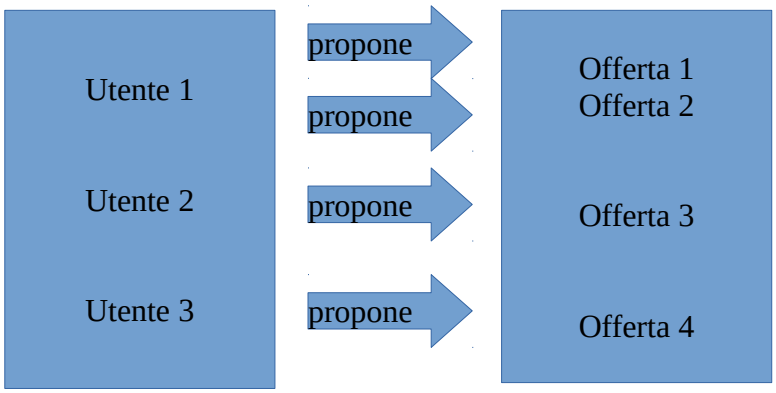
5.2 Le associazioni tra entità

L’associazione è un legame che stabilisce un’interazione tra le entità. Raccogliendo le informazioni precedentemente descritte, descrivo tutte le associazioni necessarie per lo schema E/R. La seguente rappresentazione grafica:

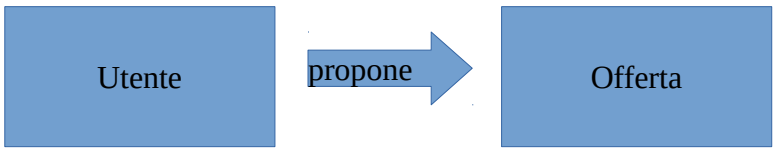


indica che A e B sono due entità e Nome una associazione fra queste; il verso di questo associazione è dato dalla direzione della freccia.

L'associazione tra le entità Utente e Offerta ha il nome propone. Ogni utente può proporre più offerte, ma un'offerta è stata proposta da un solo utente; un esempio potrebbe essere:



Quindi graficamente si traduce:



Questa rappresentazione però considera solo associazioni 1:1, cioè quando ogni utente propone un'offerta, e ogni offerta viene proposta da un singolo utente. Mentre la seconda affermazione è corretta, la prima è troppo restrittiva: infatti un utente può proporre (contemporaneamente o non) più offerte.

Si procede con l'analisi dell'associazione:

molteplicità

obbligatorietà, cioè il numero minimo di possibili istanze

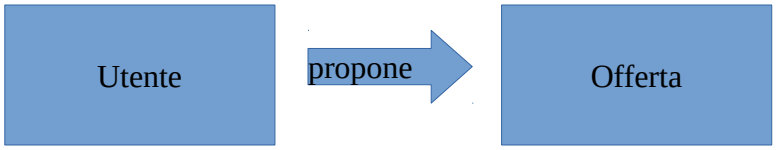
0 → L'utente può offrire facoltativamente offerte

1 → L'offerta è obbligatoriamente proposta da parte utenti

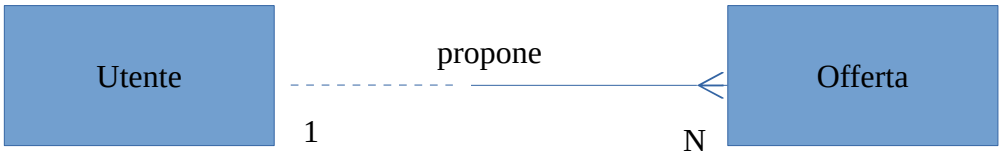
cardinalità, cioè il numero massimo di possibili istanze

n → L'utente può proporre più offerte (a molti)

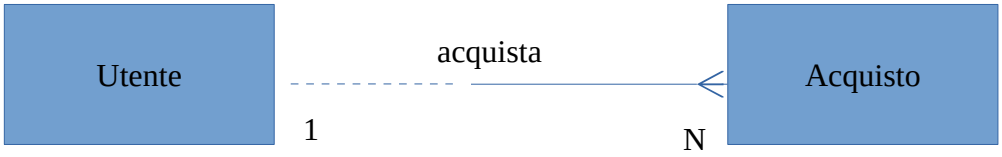
1 → L'offerta viene proposta da parte di un singolo utente (a uno)



Solitamente nel modello E/R si indica solo la cardinalità, mentre la obbligatorietà viene implicata dal tipo di linea (tratteggiata o continua). Il grafico che segue è corretto per tutti i punti analizzati precedentemente.

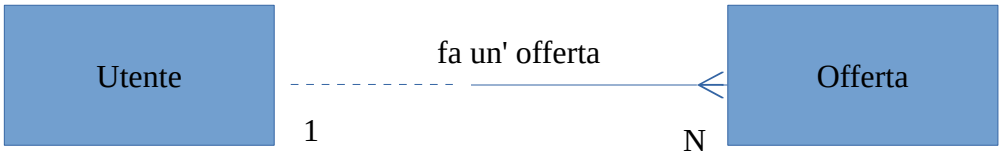


È di conseguenza un'associazione di tipo 1:N.



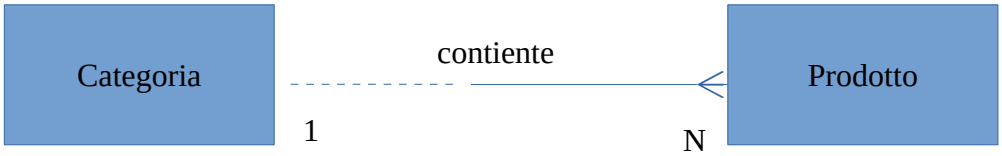
Ogni utente può effettuare più acquisti.

Ogni acquisto viene effettuata da parte di un singolo utente.



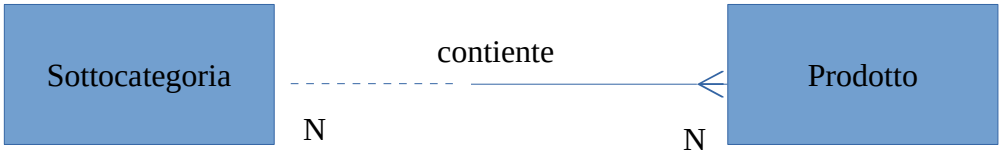
Ogni utente può effettuare più offerta.

Ogni offerta viene effettuata da parte di un singolo utente.



Ogni categoria può contenere più prodotti.

Un prodotto può appartenere ad una sola categoria.



Ogni sottocategoria può contenere più prodotti.

Ogni prodotto può avere più sottocategorie.

5.3 Attributi

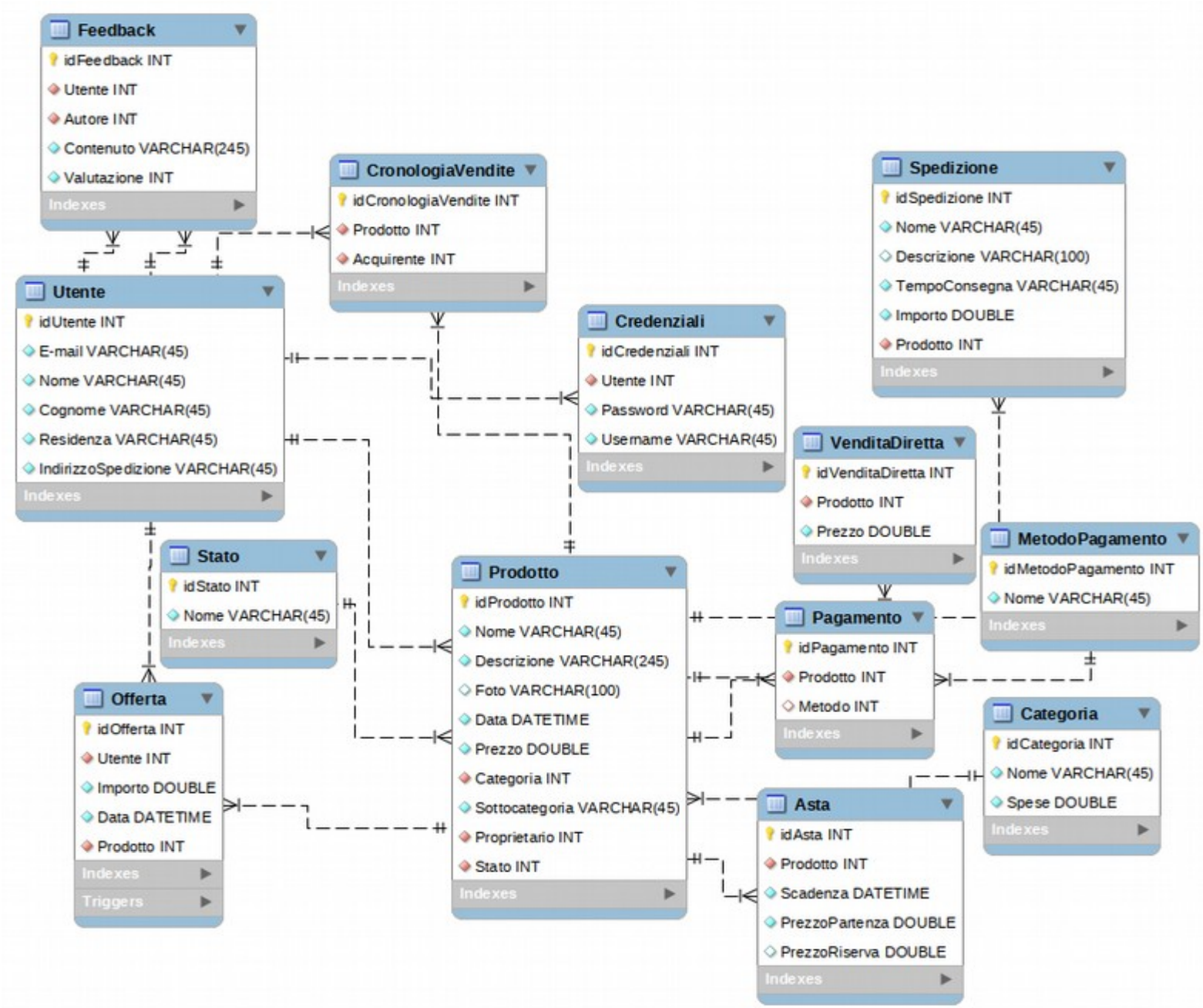
Gli attributi sono elencati nella parte inferiore del rettangolo che rappresenta l'entità, con una linea di demarcazione tra il nome dell'entità e la lista di attributi. E' importante sapere inoltre che pure le associazioni possono avere degli attributi.

5.4 Primary key (o chiavi primarie)

La chiave primaria è un insieme minimale di attributi che permettono di distinguere tra loro le istanze di una stessa entità, cioè quegli attributi che sono in grado di identificare univocamente una istanza di un'entità. Per esempio, l'attributo "Indirizzo email" dell'entità utente è la chiave primaria dell'entità, e viene rappresentato graficamente attraverso la presenza dell'acronimo: {PK} (Primary Key), posto tra parentesi graffe, acanto all'attributo chiave. Nel caso di chiave formata da più attributi, l'acronimo {PPK} è posto accanto a ognuno degli attributi che compongono la chiave.

6 Schema E/R

Raccogliendo tutte le informazioni, ora sono in grado di presentare interamente lo schema E/R:



7 Progettazione logica

7.1 Ottimizzazione(carico di lavoro)

Questa fase consiste nell'individuazione almeno di una scelta progettuale che necessita di analisi del carico di lavoro (esempi: attributo o associazione derivata, porzione di schema ristrutturabile tramite accorporamenti orizzontali/verticali, semplificazione di gerarchia); definire quindi tutte le informazioni (volume dei dati e specifica delle operazioni) che sono utili per calcolare i costi di accesso sulla base dei quali operare la scelta ottimale. In particolare il volume dei dati consiste di misure di cardinalità di entità e associazioni e di un numero medio di ciascuna istanza di entità di un'associazione. Mentre la descrizione delle operazioni consiste di schema di navigazione (indicazione qualitativa della porzione di schema concettuale contenente l dati coinvolti), frequenza di attivazione e tipo dell'operazione. L'ottimizzazione si compone di utilizzare in input e in output le seguenti caratteristiche:

- INPUT: MODELLO E/R
- OUTPUT: MODELLO E/R OTTIMIZZATO

  (tramite ANALISI DEL CARICO DI LAVORO)

7.1.1 Semplificazione di gerarchie di generalizzazione.

Questa fase richiede che le gerarchie vengono sostituite con entità e associazioni. La semplificazione di gerarchie può essere trattata in diversi modi, i quali vengono riportati qui sotto.

• Collassare la gerarchia in una singola entità:

Al termine di questa fase viene introdotto un attributo selettore per discriminare la specializzazione.

• Rimozione dell'entità padre:

Durante questa fase è presente un collasso verso il basso, nel momento in cui viene rimosso il padre. Si ricorda che questo passaggio non è possibile se la gerarchia è parziale. Mentre se la gerarchia è sovrapposta introduce

ridondanza.

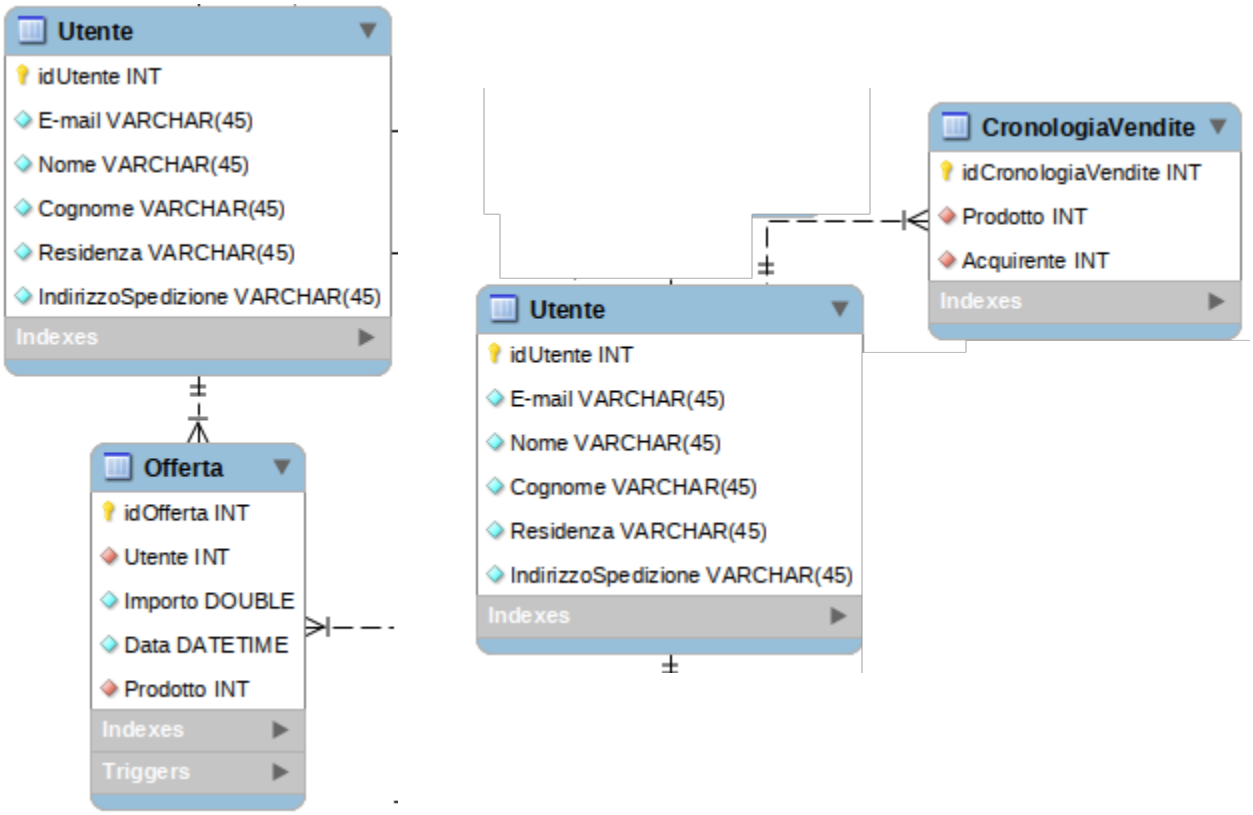
• Sostituzione della gerarchia con associazioni:

Durante questa fase non è presente nessun collasso, anzi è presente una aggiunta di nuove informazioni. Questa fase risulta essere praticabile senza restrizioni. Le scelte progettuali precedenti consideravano le due entità offerta e acquisto separate, anche se entrambe sono figlie dell'entità baratto.

Si consideri però anche la possibilità di lavorare unicamente con l'entità baratto, prendendo tutti gli attributi necessari da offerta e acquisto, considerando il prodotto in offerta o già venduto a seconda di come sono valorizzati (ad esempio il campo idAcquirente).

Si procede con il confronto fra i costi per eseguire un numero di operazioni su questi dati con e senza la gerarchia per vedere se è conveniente oppure no.

7.2 Carico di lavoro senza gerarchia



All'interno del e-commerce, consideriamo:

– 20 utenti registrati

– 100 offerte attive

– 80 acquisti effettuati

In media quindi ogni utente effettua 5 offerte, e procede con un avg di 4 acquisti.

Di conseguenza, ci saranno 100 (20*5) istanze nell'associazione “effettuare offerta” e 80 (20*4) istanze dell'associazione “effettuare acquisto”.

Invece ogni offerta viene effettuata in media da un solo utente, quindi ha avg = 1, e avremo quindi 100*1 istanze (discorso analogo per gli acquisti).

Le operazioni che lavorano su questi dati sono:

– Elenca il numero di offerte effettuate da un determinato utente.

– Trova utente offerente di una determinata offerta.

7.3 OPERAZIONE 1:

Elenca il numero di offerte effettuate da un determinato utente.

Analisi dei dati:

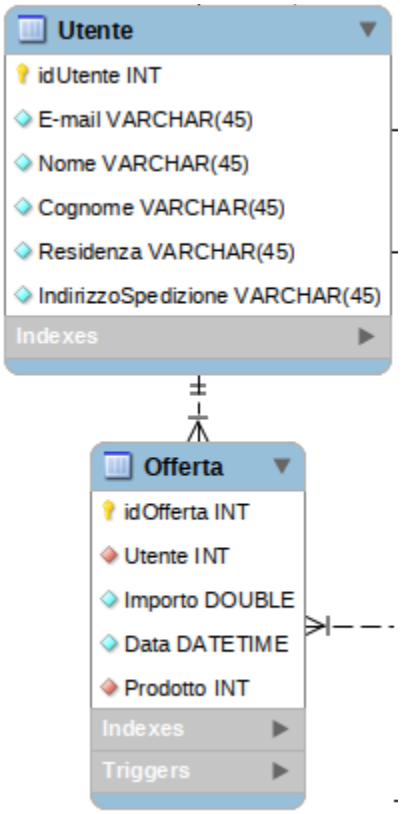
Input:

• Nome = “Julian”

• Cognome = “Sparber”

Output:

• Elenco delle offerte.



Le operazioni:

1. Read Utente

Dobbiamo leggere una istanza dell'entità Utente per trovare l'istanza dove nome = “Julian” e cognome = “Sparber”.

Preleviamo l'attributo “idUtente”.

Nota: in questa fase non viene considerato il tempo di accesso.

2. Read Offerta

Tra le 100 istanza di Offerta, solo 5 in media avranno il campo Utente uguale a quello prelevato precedentemente dall'entità Utente.

E/R	READ	WRITE
Utente	1	-
Offerta	5	-

COSTO OPERAZIONE 1 SENZA GERARCHIA = 1 READ + 5 READ = 6 READ

7.4 OPERAZIONE 2:

Trova utente offerente di una determinata offerta.

Analisi dei dati:

Input:

- id offerta = 5

Output:

- Nome
- Cognome

L'operazione lavora sulle entità “Utente” e “Offerta”

Le operazioni:

1. Read Offerta

Dobbiamo leggere una istanza dell'entità Offerta per trovare l'istanza dove idOfferta = 1.

Preleviamo l'attributo Utente.

Nota: in questa fase non viene considerato il tempo di accesso.

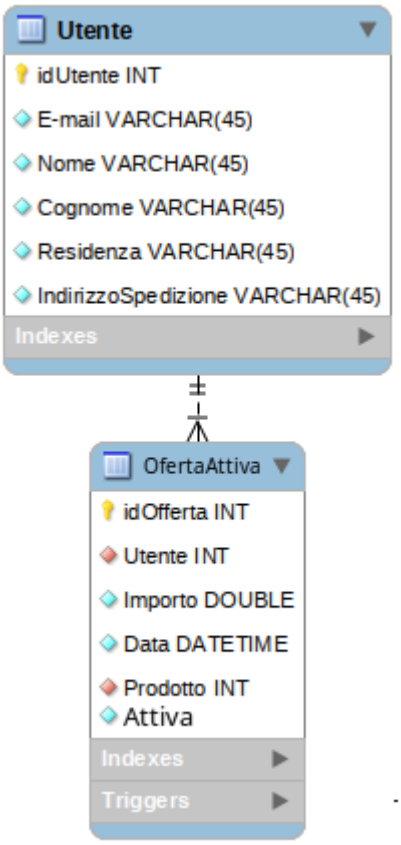
2. Read Utente

Tra le 20 istanza di Utente, solo una avrà il campo mail uguale a quello prelevato precedentemente dall'entità Offerta.

E/R	READ	WRITE
Utente	1	-
Offerta	5	-

COSTO OPERAZIONE 2 SENZA GERARCHIA = 1 READ + 1 READ = 2 READ

7.5 Carico di lavoro con gerarchia



Inoltre nell'entità “offerta Attiva” sarà necessario aggiungere un attributo “acquisto/offerta” che ci indica se il prodotto viene offerto o se è già stato acquistato.

Avremo che:

• in totale ci sono 180 offerte nel DB (100 offerte e 80 acquisti).

• in totale ci sono 20 utenti nel database.

• “avg=9” In media ogni utente effettua 9 offerte.

• “avg=1” In media ogni offerta viene effettuato (offerta attiva o non attiva) da 1 utente.

7.6 OPERAZIONE 1:

Elenca il numero di offerte effettuate da un determinato utente.

Analisi dei dati:

Input:

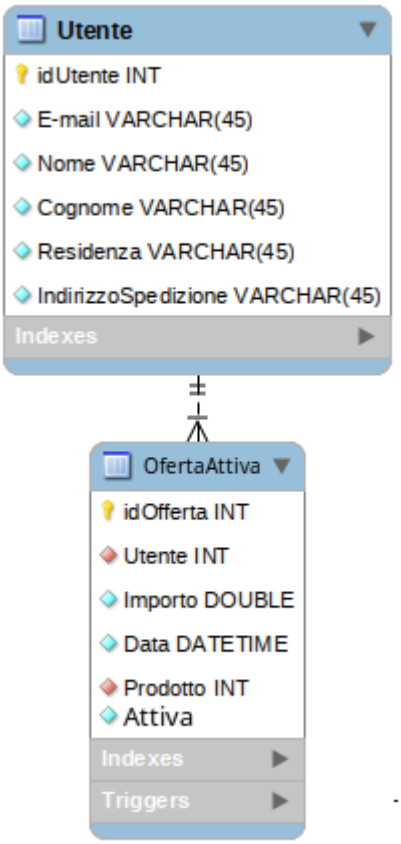
• Nome = “Julian”

• Cognome = “Sparber

Output:

• Elenco delle offerte.

L'operazione lavora sulle entità “Utente” e “OffertaAttiva”



Le operazioni:

1. Read Utente

Dobbiamo leggere una istanza dell'entità Utente per trovare l'istanza dove nome = “Julian” e cognome = “Sparber”.

Preleviamo l'attributo “mail”.

Nota: in questa fase non viene considerato il tempo di accesso.

2. Read Offerta

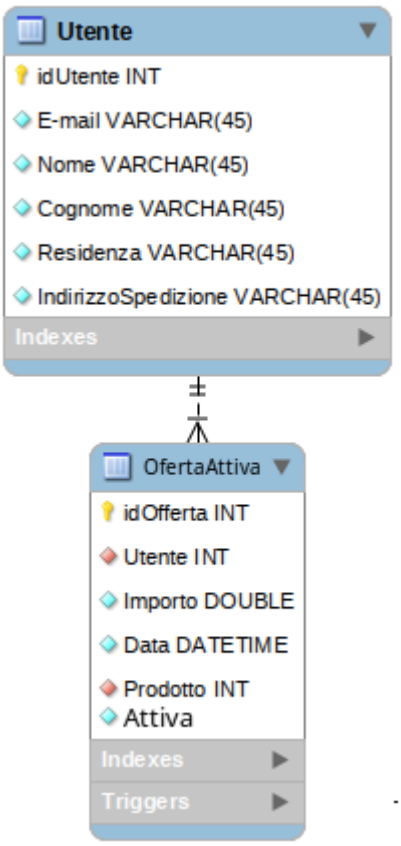
Tra le 100 istanza di Scambi, solo 9 in media avranno il campo mail uguale a quello prelevato precedentemente dall'entità Utente.

E/R	READ	WRITE
Utente	1	-
Offerta	9	-

COSTO OPERAZIONE 2 SENZA GERARCHIA = 1 READ + 9 READ = 10 READ

7.7 OPERAZIONE 2:

Trova utente offerente di una determinata offerta.
Analisi dei dati:
Input:
• id offerta = 5
Output:
• Nome
• Cognome
L'operazione lavora sulle entità “Utente” e “Offerta” e ha il seguente schema di navigazione:



Le operazioni:
1. Read Offerta
Dobbiamo leggere una istanza dell'entità Offerta per trovare l'istanza dove id = 1 e l'attributo offerta/acquisto ci indica che è una offerta in corso.
Preleviamo l'attributo idUtente.
Nota: in questa fase non viene considerato il tempo di accesso.

2. Read Utente

Tra le 20 istanza di Utente, solo una avrà il campo mail uguale a quello prelevato precedentemente dall'entità Offerta.

E/R	READ	WRITE
Utente	1	-
Offerta	1	-

COSTO OPERAZIONE 2 SENZA GERARCHIA = 1 READ + 1 READ = 2 READ

Confrontando i valori ottenuti tramite l'analisi dei costi con e senza gerarchia potrebbe sembrare che utilizzare utilizzare 2 entità o usare la entità padre “scambio” sia relativamente simile dal punto di vista delle performance . E da tenere presente però che questi valori sono valori medi e non rappresentano la realtà.

7.8 Conclusione

I costi di carico delle due operazioni con le due modalità risultano uguali perché parliamo di valori medi. Si è preferito mantenere le 2 entità separate, in quanto “acquisto” e “offerta” sono entrambe 2 oggetti che hanno un significato anche quando vengono considerati in modo isolato, cioè la definizione del termine “entità”.

2)Semplificazione attributi composti:

Questa fase consiste nell'eliminazione dell'attributo composto secondo due criteri fondamentali:

• Eliminazione dell'attributo e considerare i suoi elementi costitutivi come attributi semplici e indipendenti:

Gli attributi che risultano essere composti (caratterizzato a sua volta da ulteriori attributi) vengono eliminati e vengono considerati i suoi attributi come attributi semplici e indipendenti.

• Eliminazione degli elementi costitutivi e considerare l'attributo come semplice:

3)Semplificazione di attributi ripetuti:

Questa fase consiste nell'eliminazione di attributi ripetuti presenti nello schema E/R ottimizzato. Osservando attentamente il mio schema E/R ottimizzato risulta che non sono presenti attributi ripetuti, quindi non è possibile applicare nessuna semplificazione di tali.

4)Semplificazione identificatori esterni:

Questa fase consiste nella risoluzione delle relazioni tra entità importando o esportando le chiavi candidate.

7.9 La derivazione delle relazioni dal modello E/R

La traduzione si compone di utilizzare in input e in output le seguenti caratteristiche:

INPUT: MODELLO E/R RISTRUTTURATO

OUTPUT: MODELLO RELAZIONALE (tramite la TRADUZIONE DEI COSTRUTTI DELLO SCHEMA)

Regole per tradurre uno schema E/R in relazioni:

1. Ogni entità diventa una relazione.

2. Ogni attributo di un'entità diventa un attributo della relazione, cioè il nome di una colonna della tabella.

3. Il formato degli attributi di un'entità rimane lo stesso per gli attributi della relazione.

4. La chiave primaria dell'entità diventa la chiave primaria della relazione.

5. Nell'associazione 1:1 con partecipazione obbligatoria, le 2 entità diventano 1 unica relazione (tabella) che contiene sia gli attributi della prima che della seconda entità.

6. Nell'associazione 1:1 con partecipazione facoltativa, la scelta di creare una sola relazione, composta da tutti gli attributi di entrambe le entità, appare inadeguata in quanto comparirebbero molte righe (istanze) con valori nulli.

In questo caso si tratta l'associazione come un'associazione "uno a molti". Si creano due relazioni una per ogni entità, dove si aggiunge la chiave dell'entità con partecipazione facoltativa in coda agli attributi dell'entità con partecipazione obbligatoria, questo/i attributo/i prende (o prendono) il nome di chiave esterna(Foreign Key {FK}).

7. Per l'associazione 1:N, vedi sopra (1:1 facoltativo).

8. Nell'associazione N:N, le 2 entità diventano 2 relazioni. Poi si crea una terza relazione contenente le chiavi delle 2 entità e gli eventuali attributi dell'associazione.

7.10 Modello relazionale

Usando queste regole, posso ora tradurre lo schema E/R nel modello relazionale:

`Utente` (
 `idUtente`
 `E-mail`
 `Nome`
 `Cognome`
 `Residenza`
 `IndirizzoSpedizione`
 PRIMARY KEY (`idUtente`))

`Credenziali` (
 `idCredenziali`
 `Utente`
 `Password`
 `Username`
 PRIMARY KEY (`idCredenziali`)
 fk: `Utente` > `Utente` (`idUtente`)

`MetodoPagamento` (
 `idMetodoPagamento`
 `Nome`
 PRIMARY KEY (`idMetodoPagamento`))

`Categoria` (

```
  `idCategoria`
  `Nome`
  `Spese`
  PRIMARY KEY (`idCategoria`))

`Stato` (
  `idStato`
  `Nome`
  PRIMARY KEY (`idStato`))

`Prodotto` (
  `idProdotto`
  `Nome`
  `Descrizione`
  `Foto`
  `Data`
  `Prezzo`
  `Categoria`
  `Sottocategoria`
  `Proprietario`
  `Stato`
  PRIMARY KEY (`idProdotto`)
  fk: `Categoria` > `Categoria` (`idCategoria`)
  `Proprietario` > `Utente` (`idUtente`)
`Stato` > `Stato` (`idStato`)
```

`Pagamento` (
  `idPagamento`
  `Prodotto`
  `Metodo`
  PRIMARY KEY (`idPagamento`),
 fk: `Prodotto`
`Metodo`

`Spedizone` (
  `idSpedizone`
  `Nome`
  `Descrizione`
  `TempoConsegna`
  `Importo`
  `Prodotto`
  PRIMARY KEY (`idSpedizone`),
fk: `Prodotto` > `Prodotto` (`idProdotto`)

`Offerta` (
  `idOfferta`
  `Utente`
  `Importo`
  `Data`
  `Prodotto`

```
  PRIMARY KEY (`idOfferta`),
fk: `Prodotto` > `Prodotto` (`idProdotto`)
`Utente` > `Utente` (`idUtente`)

`Asta` (
  `idAsta`
  `Prodotto`
  `Scadenza`
  `PrezzoPartenza`
  `PrezzoRiserva`
  PRIMARY KEY (`idAsta`),
`Prodotto` > `Prodotto` (`idProdotto`)

`VenditaDiretta` (
  `idVeditaDiretta`
  `Prodotto`
  `Prezzo`
  PRIMARY KEY (`idVeditaDiretta`),
  fk: `Prodotto` > `Prodotto` (`idProdotto`)

`CronologiaVendite` (
  `idCronologiaVendite`
  `Prodotto`
  `Acquirente`
  PRIMARY KEY (`idCronologiaVendite`),
```

fk: `Prodotto` > `Prodotto` (`idProdotto`)

`Acquirente` > `Utente` (`idUtente`)

`Feedback` (

`idFeedback`

`Utente`

`Autore`

`Contenuto`

`Valutazione`

PRIMARY KEY (`idFeedback`),

`Autore` > `Utente` (`idUtente`)

`Utente` > `Utente` (`idUtente`)

8 Normalizzazione

8.1 Forme normali

Sono stati definiti opportuni criteri che devono essere soddisfatti dalle relazioni per evitare:

- ridondanza dei dati
- possibili anomalie che ne conseguono

Questi criteri prendono il nome di forme normali.

Esistono diverse forme normali ed in seguito verificheremo che le relazioni del nostro DB soddisfano i loro requisiti.

8.2 Normalizzazione

La normalizzazione consiste in un processo di trasformazione che consente di creare tabelle ben definite che rispettano questi criteri senza perdita di informazioni, consentendo operazioni di aggiunta, modifica e cancellazione delle informazioni in modo chiaro e semplice, e rendendo possibili i cambiamenti nella struttura con

l’evolvere delle esigenze degli utenti del DB.

8.3 1FN

Una relazione è in prima forma normale quando rispetta i requisiti fondamentali del modello relazionale, cioè:

- tutte le righe della tabella contengono lo stesso numero di colonne.
- gli attributi rappresentano informazioni elementari
- i valori che compaiono in una colonna sono dello stesso tipo, cioè appartengono allo stesso dominio.
- ogni riga è diversa da tutte le altre, cioè non ci possono essere 2 righe con gli stessi valori nelle colonne
- l’ordine con il quale le righe compaiono nella tabella è irrilevante

Inoltre anche le relazioni contenute devono rispettare le seguenti regole:

- ogni attributo è elementare (nome dell'utente, cognome ecc...).
- non ci sono righe uguali (sempre verificata grazie all'uso dei {PK} che sono identificatori univoci di tipo auto-incrementale).
- non ci sono attributi ripetitivi (ogni attributo richiede un’informazione indipendente dalle altre.
- Nessun attributo dipende da un altro attributo che non sia chiave).

8.4 2FN

Una relazione è in seconda forma normale quando:

- è in prima forma normale
- non ci sono attributi non-chiave che dipendono parzialmente dalla chiave.

La 2FN elimina la dipendenza parziale degli attributi dalla chiave e riguarda il caso di relazioni con chiavi composte, cioè formate da più attributi (più {PPK}). Il DB che stiamo creando non ha chiavi composte di conseguenza ogni attributo può solo dipendere dalla chiave.

8.5 3FN

Una relazione è in 3FN quando:

- è in 2FN
- tutti gli attributi non-chiave dipendono dalla chiave, cioè non possiede attributi NON-CHIAVE che dipendono da attributi non-chiave.

Osservando attentamente le relazioni del mio DB, affermo che sono tutte in 3FN.

8.6 BCNF (forma normale di Boyce-Codd)

Si dice dipendenza funzionale tra attributi quando il valore di un attributo o un insieme di attributi chiamato (A) determina in modo univoco il singolo valore dell'attributo B e si indica cosi:

$A \rightarrow B$.

In questo caso si dice che B dipende funzionalmente da A, oppure che A è determinante per B.

Una relazione è in BCNF se e solo se ciascun determinante è una chiave candidata.

Equivale a dire che se $A \rightarrow B$ è una dipendenza funzionale nella relazione R, allora B è un sottoinsieme di A, oppure A è una chiave candidata.

8.6.1 Chiave candidata:

E' ogni insieme minimale di 1 o più attributi che possono svolgere la funzione di chiave (ci possono essere molte chiavi candidate, ma solo una può essere {PK}).

Essendo chiave candidata, significa che il determinante ha tutti i requisiti per essere {PK}, quindi non dipende da nessun altro attributo che non sia {PK} o chiave candidata. Vengono così eliminate tutte le dipendenze funzionali, sia quelli parziali sia quelli transitive.

8.6.2 Individuando tutte le chiavi candidate delle nostre relazioni, possiamo affermare che sono tutti in BCNF:

8.6.2.1 utente

Chiavi candidate:

- idUtente {PK}
- E-mail, Nome, Cognome, Residenza, IndirizzoSpedizione

Le dipendenze funzionali sono:

a) {idUtente} → E-mail, Nome, Cognome, Residenza, IndirizzoSpedizione

b) {E-mail, Nome, Cognome, Residenza, IndirizzoSpedizione} → idUtente

La relazione “utente” è in BCFN perché ogni “utente” (a, b) è chiave candidata.

8.6.2.2 Offerta

Chiavi candidate:

idOfferta {PK}

Utente, Importo, Data, Prodotto

Le dipendenze funzionali sono:

a) {idOfferta} → Utente, Importo, Data, Prodotto

b) {Utente, Importo, Data, Prodotto} → idOfferta

La relazione è in BCFN perché a è una chiave candidata.

8.6.2.3 Categoria

Chiavi candidate:

idCategoria {PK}

Nome

Le dipendenze funzionali sono:

a) {idCategoria} → Nome

b) {Nome} → idCategoria

La relazione è in BCFN perché a è una chiave candidata.

8.6.2.4 Feedback

Chiavi candidate:

idFeedback {PK}

Utente, Autore, Contenuto, Valutazione

Le dipendenze funzionali sono:

a) {idFeedback} → Utente, Autore, Contenuto, Valutazione

b) {Utente, Autore, Contenuto, Valutazione} → idFeedback

La relazione è in BCFN perché a è una chiave candidata.

8.6.2.5 Pagamento

Chiavi candidate:

idPagamento {PK}

Nome

Le dipendenze funzionali sono:

a) {idPagamento} → Nome

b) {Nome} → idPagamento

La relazione è in BCFN perché a è una chiave candidata.

8.6.2.6 Prodotto

Chiavi candidate:

idProdotto {PK}

Nome, Descrizione, Foto, Prezzo, Categoria, Proprietario, Acquirente, Pagamento, Spedizione, Sottocategoria

Le dipendenze funzionali sono:

a) {idProdotto} →Nome, Descrizione, Foto, Prezzo, Categoria, Proprietario, Acquirente, Pagamento, Spedizione, Sottocategoria

b) {Nome, Descrizione, Foto, Prezzo, Categoria, Proprietario, Acquirente, Pagamento, Spedizione, Sottocategoria} → idProdotto

La relazione è in BCFN perché a è una chiave candidata.

8.6.2.7 Spedizione

Chiavi candidate:

idPagamento {PK}

Nome, TempoConsegna, Importo

Le dipendenze funzionali sono:

a) {idSpedizone} → Nome, TempoConsegna, Importo

b) {Nome, TempoConsegna, Importo} → idSpedizone

La relazione è in BCFN perché a è una chiave candidata.

Abbiamo dimostrato che tutte le tabelle sono in forma normale Boyce-Codd, che implica anche tutti gli altri libelli (1FN, 2FN e 3FN). Di conseguenza possiamo affermare che tutte le relazioni hanno i requisiti delle forme normali, e proseguiamo con la traduzione dello schema relazionale in linguaggio mySQL.

9 Creazione della database

```
CREATE SCHEMA IF NOT EXISTS `piattaforma` DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci;


CREATE TABLE IF NOT EXISTS `piattaforma`.`Utente` (
  `idUtente` INT NOT NULL AUTO_INCREMENT,
  `E-mail` VARCHAR(45) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `Residenza` VARCHAR(45) NOT NULL,
  `IndirizzoSpedizione` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idUtente`))
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `piattaforma`.`Credenziali` (
  `idCredenziali` INT NOT NULL AUTO_INCREMENT,
  `Utente` INT NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `Username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idCredenziali`),
  INDEX `fk_Credenziali_1_idx` (`Utente` ASC),
  UNIQUE INDEX `Username_UNIQUE` (`Username` ASC),
  CONSTRAINT `fk_Credenziali_1`
    FOREIGN KEY (`Utente`)
    REFERENCES `piattaforma`.`Utente` (`idUtente`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`MetodoPagamento` (
  `idMetodoPagamento` INT NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idMetodoPagamento`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`Categoria` (
  `idCategoria` INT NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Spese` DOUBLE NOT NULL,
  PRIMARY KEY (`idCategoria`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`Stato` (
  `idStato` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idStato`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`Prodotto` (
  `idProdotto` INT NOT NULL AUTO_INCREMENT,
```

```
  `Nome` VARCHAR(45) NOT NULL,
  `Descrizione` VARCHAR(245) NOT NULL,
  `Foto` VARCHAR(100) NULL,
  `Data` DATETIME NOT NULL,
  `Prezzo` DOUBLE NOT NULL,
  `Categoria` INT NOT NULL,
  `Sottocategoria` VARCHAR(45) NOT NULL,
  `Proprietario` INT NOT NULL,
  `Stato` INT NOT NULL,
  PRIMARY KEY (`idProdotto`),
  INDEX `IdCategoria_idx` (`Categoria` ASC),
  INDEX `IdUtente_idx` (`Proprietario` ASC),
  INDEX `fk_Prodotto_1_idx` (`Stato` ASC),
  CONSTRAINT `IdCategoria`
    FOREIGN KEY (`Categoria`)
    REFERENCES `piattaforma`.`Categoria` (`idCategoria`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `IdUtente`
    FOREIGN KEY (`Proprietario`)
    REFERENCES `piattaforma`.`Utente` (`idUtente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Prodotto_1`
    FOREIGN KEY (`Stato`)
```

```
    REFERENCES `piattaforma`.`Stato` (`idStato`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`Pagamento` (
  `idPagamento` INT NOT NULL AUTO_INCREMENT,
  `Prodotto` INT NOT NULL,
  `Metodo` INT NULL,
  PRIMARY KEY (`idPagamento`),
  INDEX `fk_Pagamento_2_idx` (`Prodotto` ASC),
  INDEX `fk_Pagamento_1_idx` (`Metodo` ASC),
  CONSTRAINT `fk_Pagamento_1`
    FOREIGN KEY (`Metodo`)
    REFERENCES `piattaforma`.`MetodoPagamento` (`idMetodoPagamento`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Pagamento_2`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `piattaforma`.`Prodotto` (`idProdotto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`Spedizone` (
```

```
  `idSpedizone` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Descrizione` VARCHAR(100) NULL,
  `TempoConsegna` VARCHAR(45) NOT NULL,
  `Importo` DOUBLE NOT NULL,
  `Prodotto` INT NOT NULL,
  PRIMARY KEY (`idSpedizone`),
  INDEX `fk_Spedizone_1_idx` (`Prodotto` ASC),
  CONSTRAINT `fk_Spedizone_1`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `piattaforma`.`Prodotto` (`idProdotto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`Offerta` (
  `idOfferta` INT NOT NULL AUTO_INCREMENT,
  `Utente` INT NOT NULL,
  `Importo` DOUBLE NOT NULL,
  `Data` DATETIME NOT NULL,
  `Prodotto` INT NOT NULL,
  PRIMARY KEY (`idOfferta`),
  INDEX `IdProdotto_idx` (`Prodotto` ASC),
  INDEX `IdUtente_idx` (`Utente` ASC),
  CONSTRAINT `fk_Offerta_Prodotto`
```

```
    FOREIGN KEY (`Prodotto`)
    REFERENCES `piattaforma`.`Prodotto` (`idProdotto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Offerta_Utente`
    FOREIGN KEY (`Utente`)
    REFERENCES `piattaforma`.`Utente` (`idUtente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`Asta` (
  `idAsta` INT NOT NULL AUTO_INCREMENT,
  `Prodotto` INT NOT NULL,
  `Scadenza` DATETIME NOT NULL,
  `PrezzoPartenza` DOUBLE NOT NULL,
  `PrezzoRiserva` DOUBLE NULL,
  PRIMARY KEY (`idAsta`),
  INDEX `fk_Asta_1_idx` (`Prodotto` ASC),
  CONSTRAINT `fk_Asta_1`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `piattaforma`.`Prodotto` (`idProdotto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `piattaforma`.`VenditaDiretta` (
  `idVeditaDiretta` INT NOT NULL AUTO_INCREMENT,
  `Prodotto` INT NOT NULL,
  `Prezzo` DOUBLE NOT NULL,
  PRIMARY KEY (`idVeditaDiretta`),
  INDEX `fk_VeditaDiretta_1_idx` (`Prodotto` ASC),
  CONSTRAINT `fk_VeditaDiretta_1`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `piattaforma`.`Prodotto` (`idProdotto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`CronologiaVendite` (
  `idCronologiaVendite` INT NOT NULL AUTO_INCREMENT,
  `Prodotto` INT NOT NULL,
  `Acquirente` INT NOT NULL,
  PRIMARY KEY (`idCronologiaVendite`),
  INDEX `fk_CronologiaVendite_1_idx` (`Prodotto` ASC),
  INDEX `fk_CronologiaVendite_2_idx` (`Acquirente` ASC),
  UNIQUE INDEX `Prodotto_UNIQUE` (`Prodotto` ASC),
  CONSTRAINT `fk_CronologiaVendite_1`
    FOREIGN KEY (`Prodotto`)
    REFERENCES `piattaforma`.`Prodotto` (`idProdotto`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_CronologiaVendite_2`
    FOREIGN KEY (`Acquirente`)
    REFERENCES `piattaforma`.`Utente` (`idUtente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `piattaforma`.`Feedback` (
  `idFeedback` INT NOT NULL AUTO_INCREMENT,
  `Utente` INT NOT NULL,
  `Autore` INT NOT NULL,
  `Contenuto` VARCHAR(245) NOT NULL,
  `Valutazione` INT NOT NULL,
  PRIMARY KEY (`idFeedback`),
  INDEX `idAutore_idx` (`Autore` ASC),
  INDEX `idUtende_idx` (`Utente` ASC),
  CONSTRAINT `idUtende`
    FOREIGN KEY (`Utente`)
    REFERENCES `piattaforma`.`Utente` (`idUtente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `idAutore`
    FOREIGN KEY (`Autore`)
```

```
  REFERENCES `piattaforma`.`Utente` (`idUtente`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

10 Trigger

Il trigger è una procedura che viene eseguita in maniera automatica in corrispondenza di un determinato evento. In questo modo si ha disposizione una tecnica per specificare e per mantenere vincoli di integrità anche complessi. Tale procedura è associata ad una tabella e viene automaticamente richiamata dal motore del database quando una certo evento avviene all'interno della tabella. Le modifiche sulla tabella possono includere operazioni di:

- INSERT
- UPDATE
- DELETE

Nel momento in cui viene definito un trigger occorre specificare anche se è definito dopo (AFTER) un certo evento o prima (BEFORE) un certo evento.

La sintassi per la creazione di un trigger risulta essere la seguente:

```
create trigger <nome_trigger>
<before | after> <insert | update | delete>
into <nome_tabella>
for each row
<comando sql>
```

Il trigger utilizzato per aumentare il prezzo del prodotto dopo l' inserimento di un'offerta

```
CREATE DEFINER = CURRENT_USER TRIGGER
`piattaforma`.`Offerta_AFTER_INSERT` AFTER INSERT ON `Offerta` FOR EACH
ROW UPDATE `piattaforma`.`Prodotto` SET Prezzo = (SELECT MAX(Importo) FROM
Offerta WHERE Prodotto = NEW.Prodotto) WHERE idProdotto = NEW.Prodotto;
```

Il trigger che viene inserito per ogni prodotto in asta

```
CREATE EVENT `event_name`
ON SCHEDULE CURRENT_TIMESTAMP + INTERVAL 1 MONTH
DO BEGIN
INSERT INTO `CronologiaVendite` (`Prodotto`, `Acquirente`) VALUES ($idProdotto, (SELECT Utente FROM Offerta WHERE Importo = (SELECT MAX(Importo) FROM Offerta) AND Prodotto = $idProdotto));
DELETE FROM `Asta` WHERE idProdoto = $idProdotto;
UPDATE `Prodotto` SET Stato = 'in attesa' WHERE idProdoto = $idProdotto;
END;
```

11 Inserimento dei dati fondamentali

Tutto questi valori non sono modificabile dei utente

```
INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (001, 'Computer', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (002, 'Phone', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (003, 'Cibo', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (004, 'Mobili', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (005, 'Imobili', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (006, 'Uomo', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (007, 'Dona', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (008, 'Bambini', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (009, 'Sport', '3.00');

INSERT INTO `Categoria` (`idCategoria`, `Nome`, `Spese`) VALUES (010, 'Altro', '3.00');


INSERT INTO `MetodoPagamento` (`idMetodoPagamento`, `Nome`) VALUES (001, 'Carta di Credito');

INSERT INTO `MetodoPagamento` (`idMetodoPagamento`, `Nome`) VALUES (002, 'Bonifcio Bancario');

INSERT INTO `MetodoPagamento` (`idMetodoPagamento`, `Nome`) VALUES (003, 'PayPal');

INSERT INTO `MetodoPagamento` (`idMetodoPagamento`, `Nome`) VALUES (004, 'Contrassegno');


INSERT INTO `Stato` (`Nome`) VALUES ('asta');

INSERT INTO `Stato` (`Nome`) VALUES ('Vendita diretta');

INSERT INTO `Stato` (`Nome`) VALUES ('in attesa');
```

INSERT INTO `Stato` (`Nome`) VALUES ('venduto');

12 Aggiornamenti del database

12.1 Aggiornamenti del DB consento ai non registrati

12.1.1 Inserimento di un' utente

BEGIN;

INSERT INTO `Utente` (`E-mail`, `Nome`, `Cognome`, `Residenza`, `IndirizzoSpedizione`) VALUES ('julian@sparber.net', 'Julian', 'Sparber', 'Via home 2 City 23045', 'Via home 2 City 23045') SELECT DISTINCT Username FROM Credenziali WHERE NOT EXISTS Username = "jsparber";

INSERT INTO `Credenziali` (`Utente`, `Password`, `Username`) VALUES (LAST_INSERT_ID(), 'mysecretpassword', 'jsparber');

COMMIT;

12.2 Aggiornamenti del DB consento solo ad utenti registrati

12.2.1 Inserimento di un' prodotto

BEGIN;

INSERT INTO `Prodotto` (`Nome`, `Descrizione`, `Foto`, `Data`, `Prezzo`, `Categoria`, `Sottocategoria`, `Proprietario`, `Stato`) VALUES ('Notebook', 'portatile', 'https://i.imgur.com/a0UesaP.jpg', NOW(), '350.50', '001', 'Notebook, old', '002', '001');

INSERT INTO `Pagamento` (`Prodotto`, `Metodo`) VALUES (LAST_INSERT_ID(), 001);

INSERT INTO `Asta` (`Prodotto`, `Scadenza`, `PrezzoPartenza`, `PrezzoRiserva`) VALUES (LAST_INSERT_ID(), NOW() , '3.00', '10.00');

INSERT INTO `VenditaDiretta` (`Prodotto`, `Prezzo`) VALUES (LAST_INSERT_ID(), '40.00');

INSERT INTO `Spedizone` (`Nome`, `Descrizione`, `TempoConsegna`, `Importo`, `Prodotto`) VALUES ('SDA', 'Spedizione normale', '3 Giorni', '3.50', LAST_INSERT_ID());

COMMIT;

12.2.2 Inserire un feedback

INSERT INTO `piattaforma`.`Feedback` (`idFeedback`, `Utente`, `Autore`, `Contenuto`, `Valutazione`) VALUES (NULL, NULL, NULL, NULL, NULL);

12.3 Aggiornamenti del DB consento sui propri prodotti

12.3.1 Modificare il profilo dell'utente

Profilo dell Utente:

UPDATE `Utente` SET E-mail=$E-mail, Nome=$nome, Cognome=$cognome, Residenza=$residezna, IndirizzoSpedizione=$spedizione WHERE idUtente=$idUtente;

Password e username:

UPDATE `Credenziali` SET Password=$password, Username=$username WHERE Utente = $idUtente;

12.3.2 Modificare i dettagli di un prodotto

UPDATE `Prodotto` SET Nome=$nome, Descrizione=$Descrizione, Foto=$Foto, Prezzo=$prezzo, Categoria=$categoria, Sottocategoria=$sottocategoria WHERE idUtente = $idUtente AND idProdotto = $idProdotto;

12.4 Aggiornamenti del DB consento su tutti i prodotti in vendita

12.4.1 fare una offerta per un prodotto in asta

INSERT INTO `piattaforma`.`Offerta` (`idOfferte`, `Utente`, `Importo`, `Data`, `Prodotto`) VALUES (NULL, NULL, NULL, NULL, NULL);

12.5 Interrogazioni al DB consento a tutti

12.5.1 Elencare tutti i prodotti

SELECT * FROM Prodotto

12.5.2 Elencare i prodotti di una categoria

SELECT * FROM Prodotto p, Categoria c WHERE p.Categoria = c.idCategoria

AND c.Nome = $Categoria;

12.5.3 Elencare tutti i prodotti inseriti dopo di una certa data

SELECT * FROM Prodotto WHERE Data > '2013-01-28 21:00:00';

12.5.4 Elencare tutti prodotti di un venditore

SELECT * FROM Prodotto WHERE Proprietario = (SELECT Utente FROM Credenziali WHERE Username = $Veditore);

12.5.5 Cercare un prodotto con varie parole chiave

SELECT p.idProdotto, p.Nome, p.Descrizione, p.Foto, p.Prezzo, c.Nome AS Categoria, p.Sottocategoria FROM Prodotto p, Categoria c WHERE p.Categoria = c.IdCategoria AND (p.Descrizione LIKE '%keyword%' OR p.Nome LIKE '%keyword%');

12.5.6 Stampare i dettagli di un prodotto

SELECT * FROM Prodotto WHERE idProdotto = $idProdotto;

12.5.7 Stampare i dettagli del venditore

SELECT Nome, Cognome, Residenza FROM Utente WHERE idUtente = (SELECT Utente FROM Credenziali WHERE Username = $Utente);

12.6 Interrogazioni al DB consento solo ad utenti registrati

Elencare tutti prodotti comprati dell'utente

SELECT * FROM CronologiaVendite WHERE Acquirente = $Self;

13 Progettazione Fisica

La progettazione fisica è l'ultima fase della progettazione di una base di dati. Essa rappresenta l'effettiva installazione degli archivi elettronici: essa indica l'ubicazione dei dati nelle memorie di masse (dischi).

La progettazione fisica è quindi l'implementazione della progettazione logica sui supporti per la registrazione fisica dei dati.

Un compito importante della progettazione fisica è quello di definire:

1. Le strutture di memorizzazione delle tabelle (si parla di organizzazione primaria dei dati).

2. Le strutture ausiliarie di accesso ai dati (organizzazione secondaria dei dati).

Una struttura di memorizzazione per una tabella è una struttura di dati che consente di salvare in memoria i dati della tabella. In poche parole dobbiamo decidere dove salvare le tabelle del nostro DB. Di solito un DB viene memorizzato in memoria secondaria, in particolare sui HDD.

Si sceglie la memoria secondaria al posto della memoria principale (RAM, cache), perché I DB sono di solito di grandi dimensioni e non ci starebbe dentro la RAM, gli HDD invece riescono a memorizzare permanentemente i dati ed inoltre costa molto di meno comprare memoria secondaria che memoria principale.

Quando implementiamo il nostro DB su un HDD:

- una tabella viene chiamata file e può essere memorizzata su un insieme di blocchi del disco
- un attributo viene chiamato campo
- una riga viene chiamata record

13.1 Organizzazione dei record nel file

Di solito i record sono inseriti nel file nell'ordine in cui sono inseriti nel DB, quindi in generale, senza un ordine preciso. Per cercare un record (cioè un'istanza di una

tabella) è necessaria una scansione sequenziale del file con costo lineare. L'ideale sarebbe che il file fosse già ordinato secondo un certo campo (attributo) prima di eseguire la ricerca, così riducendo drasticamente il costo e tempo di ricerca. Tuttavia ordinare un file ha un costo molto elevato, perciò si fa ricorso a una struttura ausiliaria di accesso, più comunemente chiamata indice.

13.2 Indici

L'indicizzazione è un metodo per consentire un accesso ordinato a un file disordinato. Un indice è un file da parte di piccole dimensioni che è indicizzato secondo 1 o più attributi di una tabella. In pratica, quel file piccolo contiene in memoria tutti gli indirizzi dei blocchi che contengono i record del file in questione, ordinati secondo quel attributo.

Gli indici portano una maggiore efficienza nelle interrogazioni (SELECT) che coinvolgono gli attributi indice. Infatti, senza l'utilizzo dell'indice avremmo dovuto scandire tutto il file per trovare il record richiesto dall'interrogazione.

Gli indici sono molto efficienti nelle operazioni di interrogazione al DB, soprattutto per la selezione di tuple (SELECT) che per l'operazione di join tra 2 tabelle.

Gli indici però sono molto sfavorevoli per le operazioni di aggiornamento, in quanto se ho bisogno di aggiornare la mia tabella (modifiche, inserimenti, cancellazioni), oltre a effettuare queste operazioni sul file vero e proprio, dovrò anche aggiornare l'indice.

Come richiesto, ora sceglierò una tabella del mio DB oggetto di interrogazioni frequenti e verificherò se e quali tecniche di organizzazione dei dati consentono di ottimizzare gli accessi.

L'interfaccia web da me sviluppato che consente agli utenti di interagire col DB, permette determinate operazioni sul DB. Le operazioni di interrogazioni (SELECT) prevalgono pesantemente sulle operazioni di aggiornamento, in quanto in uno e-commerce prevalgono le query per la visualizzazione dei prodotti all'acquisto proprio.

La maggior parte delle query lavorano sulla tabella "prodotti" e sono soprattutto operazioni di join.

Per questo motivo opterei per la scelta degli indici sulla tabella "prodotti" (le probabilmente pure su tutte le altre tabelle del DB).

13.3 Rappresentazione relazionale della tabella “Prodotto”:

• prodotto (`idProdotto`, `Nome`, `Descrizione`, `Foto`, `Data`, `Prezzo`, `Categoria`, `Sottocategoria`, `Proprietario`, `Stato`)

FK:

`Categoria`, Proprietario`, `Stato`

1) Un tipico esempio di select è la seguente:

SELECT * FROM prodotto WHERE idProdotto = 3;

Supponendo che sulla tabella prodotto non vi siano indici, la ricerca del prodotto con id=3 deve necessariamente procedere in modo sequenziale, leggendo nel caso peggiore tutti I blocchi (pagine del disco) occupati dalla tabella.

Supponendo invece che vi sia un indice sul campo id (chiave primaria), ora è possibile

trovare velocemente il record (istanza) della tabella dove l’id=3. Nella situazione peggiore, cioè che l’id cercato sia l’ultimo di quelli possibili, bisognerebbe scorrere tutto l’indice leggendo i blocchi occupati dall’indice (pochi in quanto il file è piccolo) più il blocco che contiene il record (istanza) del risultato della query.

Perciò come dichiarato prima, è molto conveniente definire un indice su un attributo di chiave primaria nel caso della SELECT.

Dal “MySQL 5.6 reference manual” possiamo vedere quanti byte sono utilizzati per i diversi datatypes(tipi di dati); con questi possiamo calcolare che per memorizzare una istanza della relazione prodotto necessitiamo

4 byte + 1*45 byte + 1*45 byte + 4 byte = 95 byte

Visto che attualmente la tabella “prodotto” contiene 7 istanze:

NTprodotti = 7

Supponendo che le pagine sul HDD siano di 1KB, il numero di pagine occupate dalla tabella sono:

NP =7 * 95/1024 = meno di uno = 1 per approssimazione

Visto che id è chiave primaria {PK} e ci sono 7 istanze:

NKid = 7

13.4 Ricerca sequenziale

Se non vi fossero indici, la ricerca sarebbe sequenziale. Nel caso peggiore, cioè che l'istanza della relazione prodotti con id=3, sia l'ultima istanza, bisogna leggere tutti i blocchi (pagine) occupati dalla tabella:

Costo = NpProdotti =4

13.5 Ricerca attraverso una primary B+ tree avendo indice clustered su id

Una primary B+ tree è in pratica un indice a forma di albero, dove solo i nodi foglia contengono i dati (i valori di chiave associati ai record (istanze) della relazione) ed i nodi interni rappresentano solo una mappa contenente separatori che indirizzano la ricerca fino alla foglia contenente il valore di chiave cercato.

Un indice clustered è un indice su un attributo secondo i cui valori il file dati è ordinato.

Supponiamo che l'utilizzazione delle foglie del primary B+ tree sia del 70% e che il grado sia uguale a 2.

u = 0.70

g=2

Sapendo che quando portiamo un DB su HDD, ogni istanza della relazione diventa un record del file:

NT = 7

NR = 7

Numero di foglie in un primary B+ tree:

$$NL = \left[\left(NR * (len(k) + len(p)) \right) / (D * u) \right]$$

len(k) → lunghezza della chiave (in byte)

len(p) → lunghezza dei puntatori ai record dati

len(k) + len(p) → occupazione della relazione prodotti (95 byte).

D → Dimensione delle foglie del B+ tree (cioè dimensione delle pagine dell'HDD = 1KB).

U → Utilizzazione delle foglie (0.70).

Quindi:

NL = (7 * 95) / (1024 * 0.70) = 665 / 716.8 =0.9

approssimiamo per eccesso:

NL = 1

L'altezza minima di un B+ tree è data dalla formula:

h = Log base (2g+1) di (NL) + 1= Log2*2+1(1) + 1 = Log base 5 di 1 + 1 = 1

Costo di accesso con indice clustered sull'attributo 'id':

CAid_prodotto = h -1 + [EK/NK * NL] + [EK/NK * NP]

- visto che id è {PK} NK = 7

- visto che vogliamo solo l'istanza dove id=3

Caid_prodotto = 2 – 1 + [1/7 * 6] + [1/7 * 4]= 37/27 =1

13.6 Ricerca attraverso una primary B+ tree avendo indice unclustered su id

Un indice unclustered è un indice su un attributo secondo i cui valori, il file dati non è ordinato.

CAid_prodotto = h -1 + [EK/NK * NL] + [EK * Φ (NR/NK , NP)]

= 1 – 1 + [1/7 * 1] + [1 * Φ (7/7 , 4)]= 6/7 + Φ (1 , 4)= 6/27 + [4 * (1- (1 - 1⁄4)1)]

= 6/27 + [4 * (1 – 3⁄4)]= 6/27 + [4 * 1⁄4] = 6/27 + 1 =2

I calcoli allora confermano la teoria. La ricerca attraverso un indice (su una {Pk}) è molto più conveniente di quella sequenziale.

Nel nostro caso la strategia vincente è quella di mettere un indice clustered sull'attributo 'id'.

2) Osserviamo ora invece l'operazione di join, eseguita molto frequentemente nel mio DB, soprattutto sulla tabella "prodotto".

Elencare i nomi dei Prodotti che vengono offerti:

SELECT Prodotto.nome FROM Prodotto join Offerta WHERE Offerta.idProdotto = Prodotto.id;

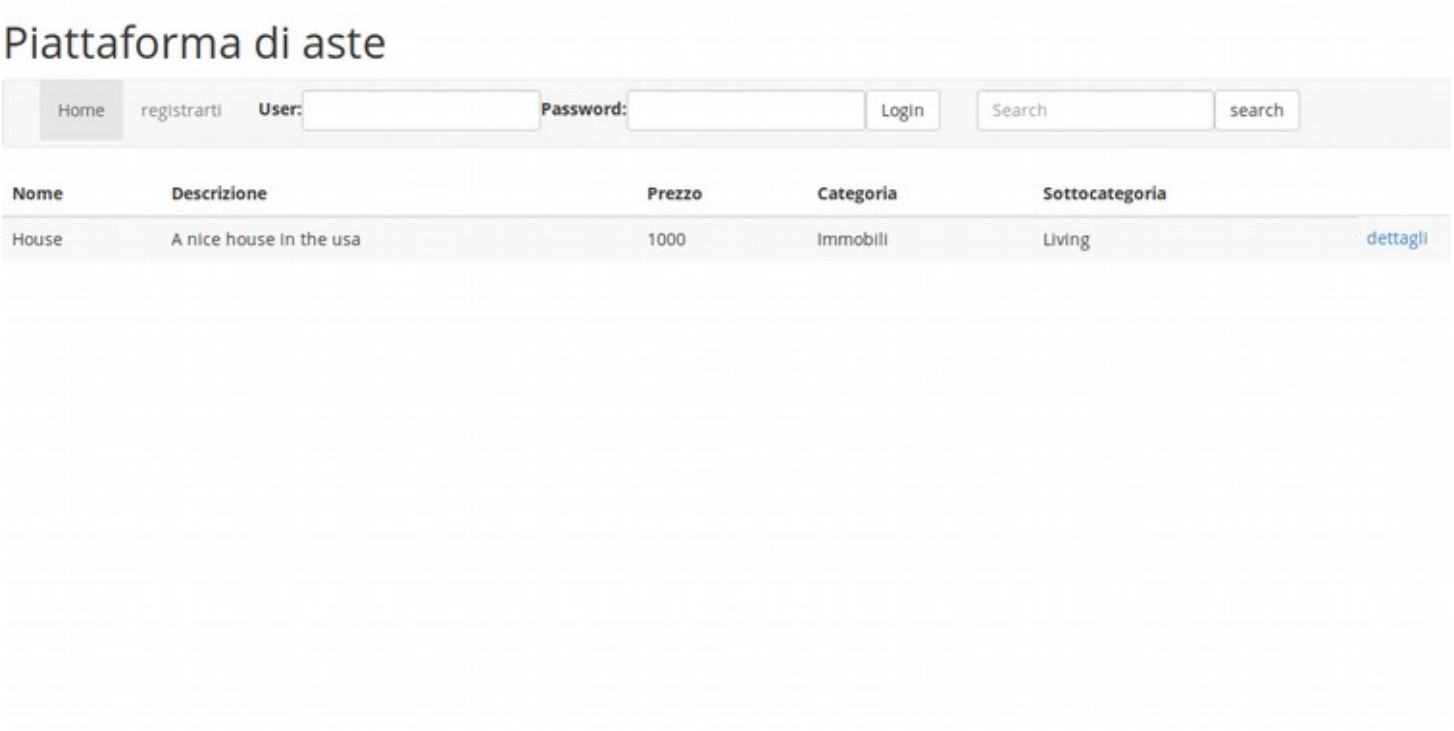
Supponendo che non vi siano indici sulla tabella "prodotto" e sulla tabella "offerte", bisognerebbe scorrere linearmente la tabella "Offerta" per ogni riga della tabella "prodotto" cercando la riga corrispondente. In questo caso allora, occorre leggere tutti i blocchi (pagine) di entrambe le tabelle.

Avendo un indice sulla chiave primaria della tabella offerta e uno sulla chiave esterna della tabella prodotto avrei una riduzione notevole degli accessi al disco. Ciò avviene perché in questo caso nell'eseguire il join occorre scorrere i 2 indici invece di scandire le 2 tabelle memorizzate sul disco e accedere ai blocchi delle 2 tabelle solo per i record che soddisfano la condizione di join.

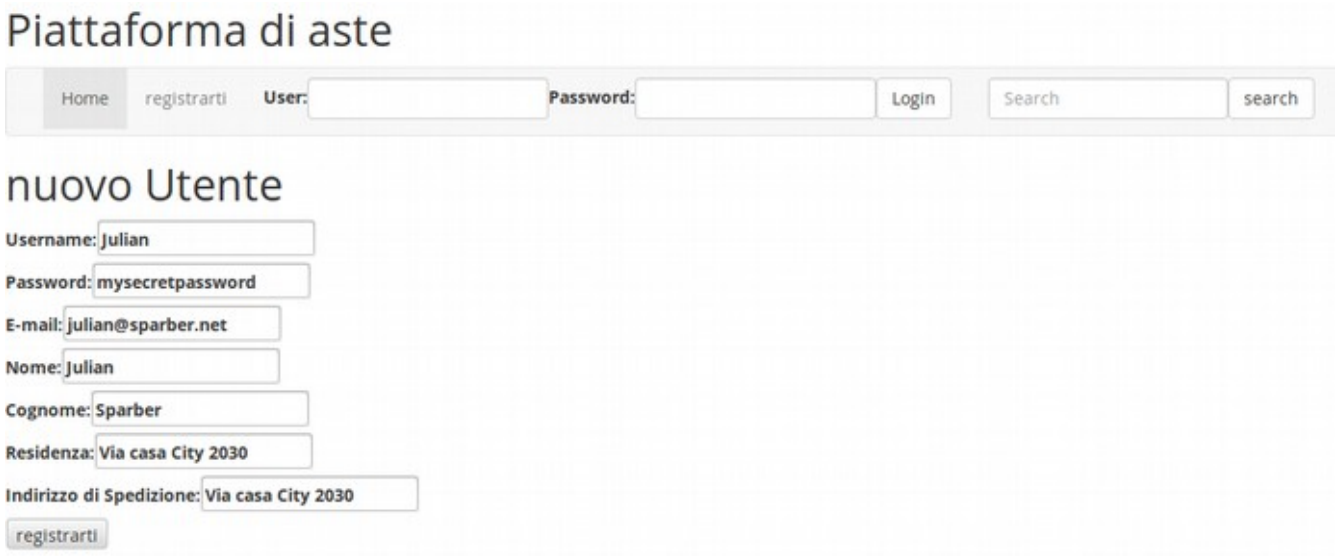
Allora concludendo possiamo dire che è molto più conveniente l'utilizzo degli indici che la ricerca sequenziale in quanto favoriscono le interrogazioni di ricerca anche se sfavoriscono quelle di aggiornamento.

14 Interfaccia Web – Implementazione fisica

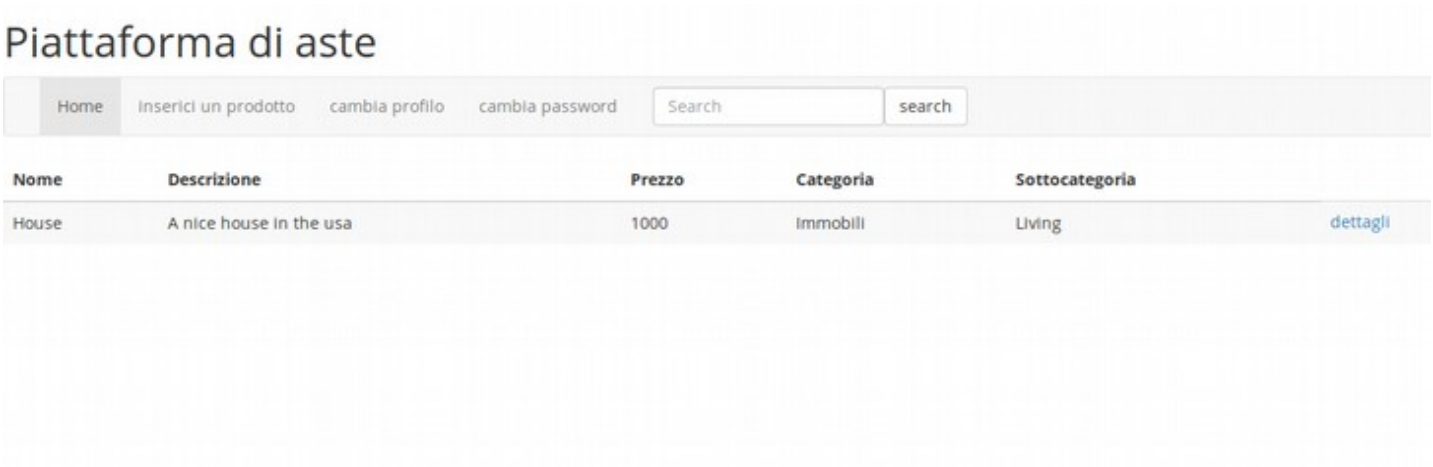
L’ultima parte del progetto comprende lo sviluppo di una interfaccia web, che grazie al linguaggio HTML e Javascript(nodejs) consente agli utenti di interagire col DB, interrogandolo e aggiornandolo. La demo e pubblicato su http://asta.sparber.net e il codice sorgente e pubblicato su https://github.com/jsparber/database.



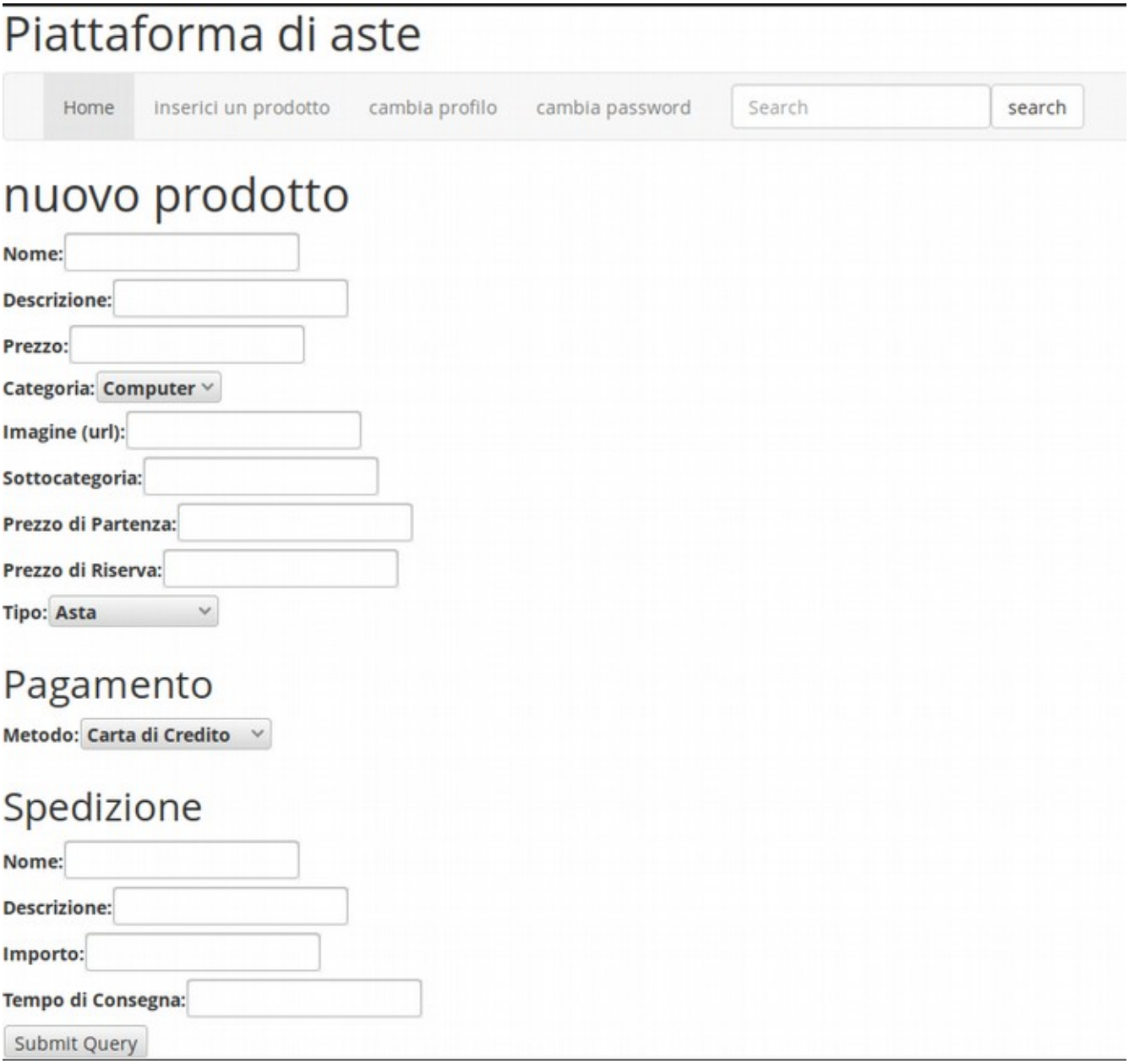
Pagina iniziale.



Registrazione nuovo utente.

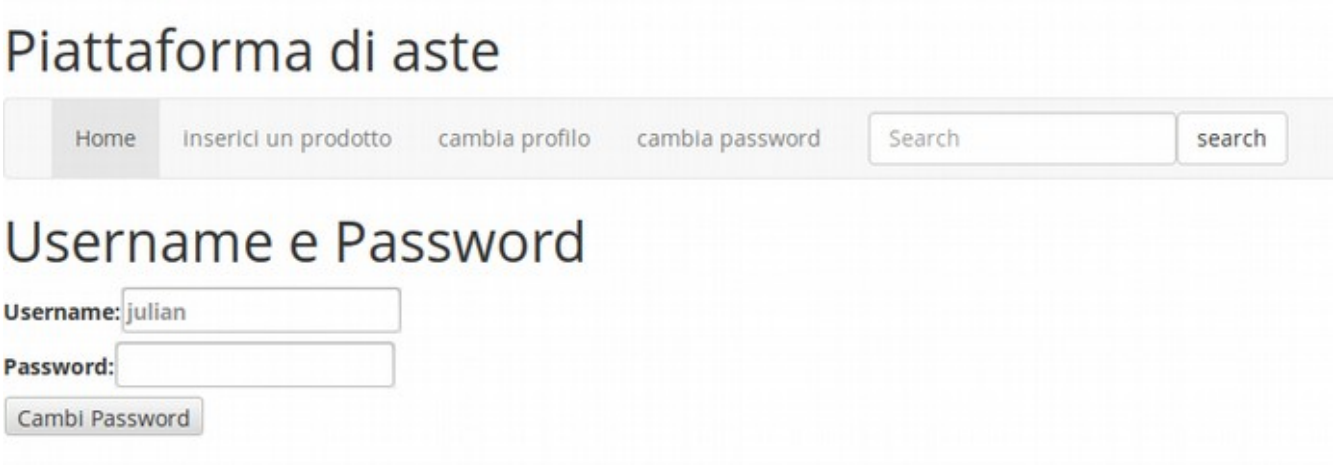


Sito dopo il login.



Inserimento di un nuovo prodotto.

Cambiare il suo profilo



Cambiare il suo password o username