Progetto Basi Di Dati

Orlando V. M. Ferazzani - 2058653

Filippo Rizzolo - 2042377

Progetto Basi di Dati - Relazione

Abstract

ML Tech è un e-commerce di PC e tastiere custom. Il sito permette di ordinare prodotti solo ad utenti che sono registrati. Al momento della registrazione, un utente inserisce il codice fiscale, il proprio nominativo, l'indirizzo per la spedizione, l'indirizzo email e il numero di cellulare. Inoltre, ad ogni utente viene assegnato un ID univoco. Ogni utente poi possiede un carrello che è reso unico dal codice identificativo del carrello, che è collegato all'utente. Inoltre, ogni carrello viene aggiornato con la data di ultima modifica e con la quantità di prodotti. I prodotti che possono essere aggiunti al carrello sono tutti identificati da un codice univoco detto SKU, da un nome, una descrizione e un prezzo. I prodotti si dividono in PC e Tastiere.

Questi sono contenuti in magazzini, che hanno un ID e un indirizzo.
Una volta che un prodotto è aggiunto al carrello, si può procedere al pagamento dei prodotti. Al momento del pagamento, l'utente sceglie il metodo con cui pagare tra bonifico, carta o PayPal dopo aver visualizzato il totale compreso di spedizione. Ogni transazione è identificata da un ID numerico che è unico per ogni transazione. Dal pagamento si procede al riepilogo dell'ordine dove si può visualizzare il codice di questo e la data in cui è stato effettuato. Quando un ordine è spedito, l'utente viene notificato. Ogni spedizione ha un tracking code univoco, una data di spedizione, una data di prevista consegna e il peso del collo. Si utilizzano i dati inseriti durante la registrazione dell'utente per la consegna.

Entità

Utente: persona che utilizza il sistema di e-commerce		
Id	Int primary key	Codice identificativo dell'utente
Nome	Varchar(255)	Nominativo dell'utente
Indirizzo	Varchar(120)	Indirizzo a cui spedire gli ordini
Telefono	Varchar(20)	Numero di telefono per contattare l'utente
Email	Varchar(255)	Email per contattare l'utente
Azienda	Varchar(255)	Se azienda: nome dell'azienda
pIva	Varchar(255)	Se azienda: pIva dell'azienda
CF	Varchar(255)	Se privato: codice fiscale della persona

Magazzino: deposito dei vari prodotti		
Id	Int primary key	Codice identificativo del magazzino
Indirizzo	Varchar(255)	Indirizzo in cui è situato il magazzino

Prodotto: oggetto venduto nell'e-commerce		
SKU	Int primary key	Codice identificativo del prodotto
Nome	Varchar(255)	Nome del prodotto
Prezzo	Float	Prezzo del prodotto
Descrizione	Text	Una descrizione del prodotto
Tipo	Varchar(255)	Tipologia di prodotto che varia tra pc o tastiera

Carrello: riepilogo di tutti i prodotti selezionati prima dell'ordine		
Id	Int primary key	Codice identificativo del carrello
DataAggiunta	Timestamp	Data dell'ultima aggiunta di un prodotto
Quantità	Int	Quantità di prodotti nel carrello

Pagamento: pagamento dei prodotti nel carrello		
Id	Int primary key	Codice identificativo del pagamento
DataPagamento	Timestamp	Data di avvenuto pagamento
Metodo	Enum	Metodologia del pagamento: carta, bonifico, PayPal
PrezzoSpedizione	Float	Costo della spedizione

Spedizione: dati della spedizione		
Id	Int primary key	Codice identificativo della spedizione
DataSpedizione	Timestamp	Data dell'avvenuta spedizione
DataConsegna	Timestamp	Data della prevista consegna
Peso	Int	Peso dei prodotti da consegnare

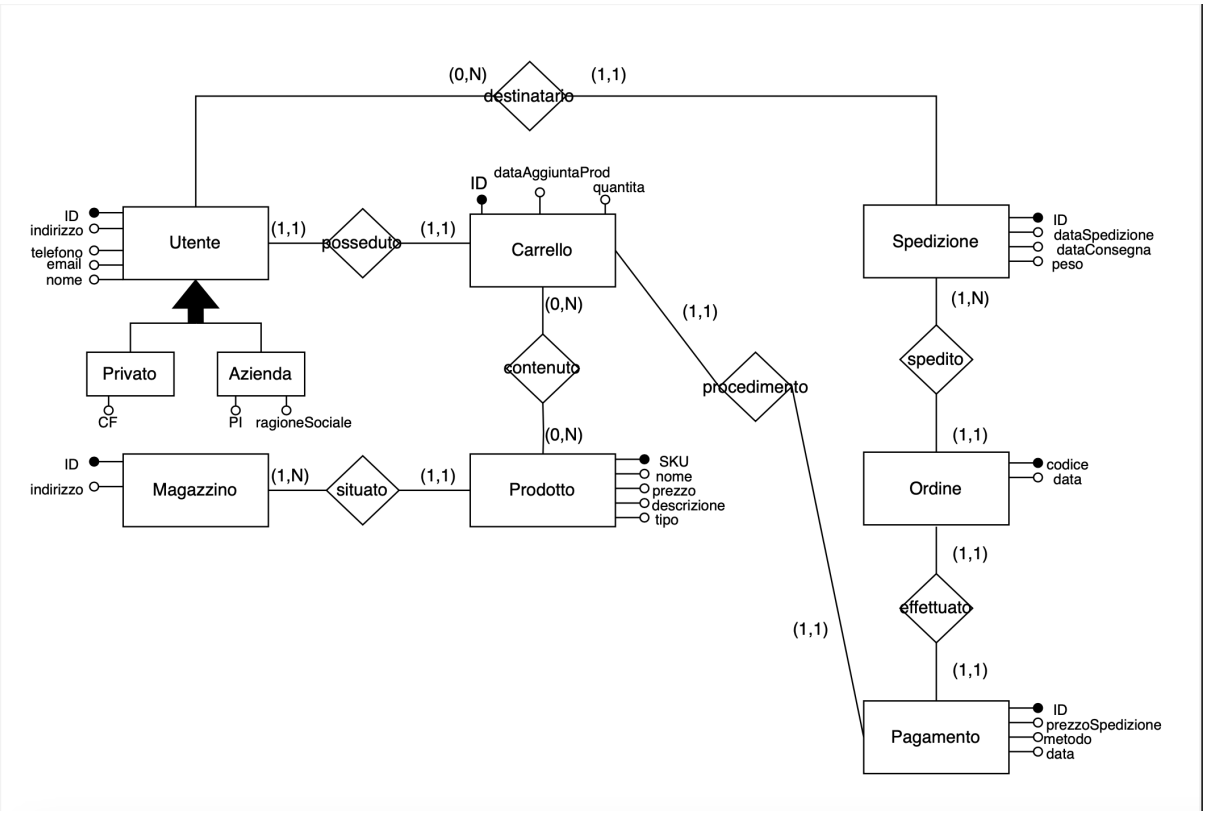
Ordine: informazioni dell'ordine		
Id	Int primary key	Codice identificativo dell'ordine
DataOrdine	Timestamp	Data dell'ordine

Tabella delle relazioni

Relazione	Entità coinvolte	Descrizione
Posseduto	Utente (1,1) Carrello (1,1)	Un utente possiede un solo carrello, un carrello è posseduto da un solo utente
Situato	Magazzino (1,N) Prodotto (1,1)	In un magazzino sono situati più prodotti, un prodotto è situato in un singolo magazzino
Contenuto	Carrello (0,N) Prodotto (0,N)	Un carrello può contenere uno o più prodotti o non contenerne nessuno ,un prodotto può essere contenuto in uno o più carrelli o non essere contenuto in nessuno
Procedimento	Carrello (1,1) Pagamento (1,1)	Da un carrello si procede ad un singolo pagamento, ad un pagamento si procede da un solo carrello
Effettuato	Pagamento (1,1) Ordine (1,1)	Con un singolo pagamento si effettua un solo ordine, un ordine è effettuato da un solo pagamento
Spedito	Ordine (1,1) Spedizione (1,N)	Un ordine viene spedito con una singola spedizione, una spedizione spedisce diversi ordini
Destinatario	Spedizione (1,1) Utente (0,N)	Una spedizione ha come destinatario un sono utente, un utente può avere una o più spedizioni come non averne nessuna

Schema ER

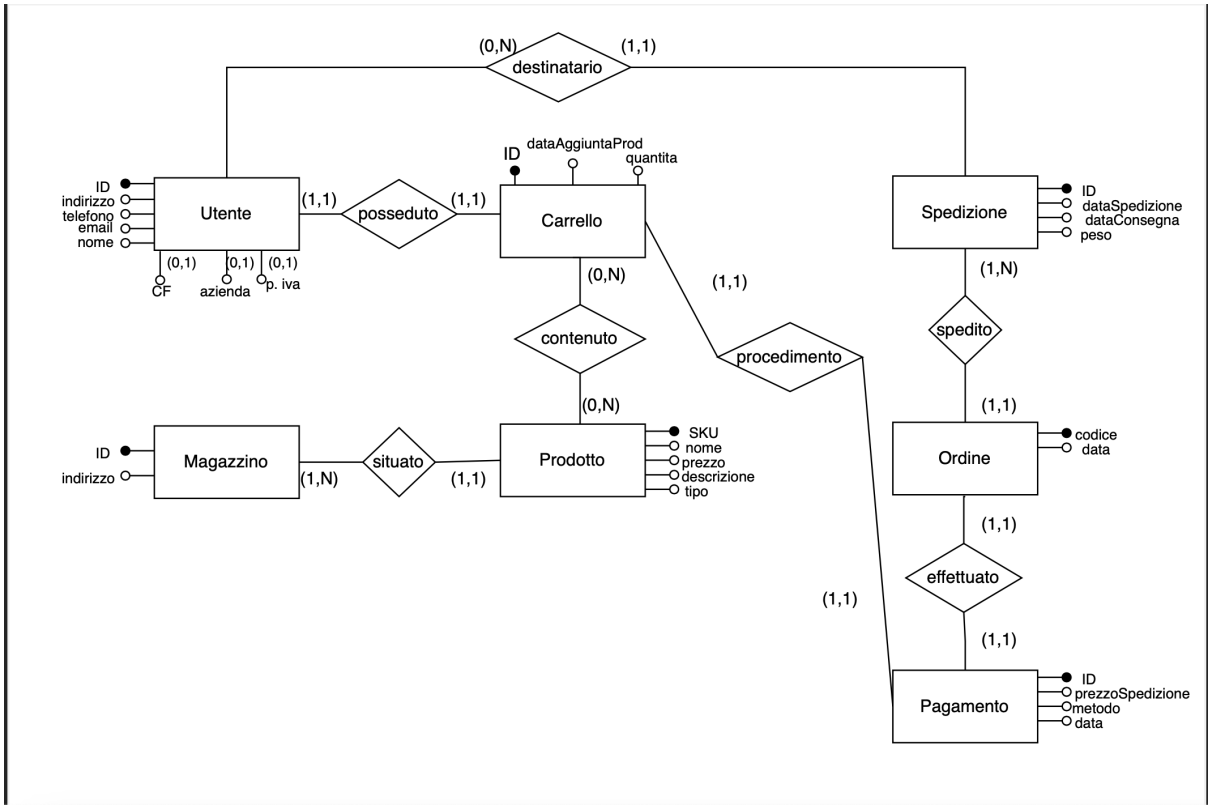
Seguendo l’indicazione della base dati descritta, si può dunque ottenere lo schema Entità-Relazione della suddetta:



Tuttavia lo schema sopra contiene ancora le generalizzazioni, quindi si procede alla sua ristrutturazione in modo che il passaggio a relazionale sia più semplice, e in modo che lo schema stesso sia più leggibile.

Schema ER Ristrutturato

Lo schema ER del database ristrutturato, ovvero senza generalizzazioni:



Come si può notare, le due generalizzazioni (di utente e di prodotto) sono state accorpate dalle loro entità padre come attributi. In particolare, l'entità utente ha ottenuto i seguenti attributi (con cardinalità 0 o 1 a significare che un utente può essere un privato o un'azienda) `CF, p.iva` e `azienda` .

Vincoli: In questo Database, si assume che un Utente non può avere contemporaneamente `p.iva` e `CF`, quindi una query che chiede tutti gli utenti che hanno codice fiscale e partita iva, dovrà restituire `NULL`

Passaggio da ER a Relazionale

Per effettuare il passaggio da schema E-R a Relazionale si utilizzano le regole per le associazioni.

Le associazioni sono di tre tipi e sono categorizzate dalla cardinalità delle relazioni nello schema.

In questo caso, seguendo quindi le suddette regole, si ottiene il seguente schema relazionale:

```
Utente(Id, Nome, Indirizzo, Telefono, Email, Azienda, pIva, CF)
Magazzino(Id, Indirizzo)
Prodotto(SKU, Nome, Prezzo, Descrizione, Tipo)
Situato(Magazzino*, Prodotto*)
```

```
Carrello(Id, DataAgiunta, Quantità, Utente*)
Contenuto(Prodotto*, Carrello*)
Pagamento(Id, DataPagamento, Metodo, PrezzoSpedizione, PrezzoTotale, Carrello*)
Spedizione(Id, DataSpedizione, DataConsegna, Peso)
Ordine(Id, DataOrdine, Pagamento*, Spedizione*)
Destinatario(Utente*, Spedizione*)
```

Codice C++ per connessione al database

Per connettersi al database, si utilizza uno script in C++. Per sfruttare al massimo la programmazione modulare, abbiamo creato un file header con le funzioni adibite alla connessione in modo che, includendo semplicemente l'header non si deve scrivere ogni volta le stesse linee di codice, ma basta chiamare la funzione `connection`. (tutti i file cpp e .h saranno inclusi nella consegna)

Il file `header.h` contiene la funzione di connessioni e altre funzioni utili che vengono utilizzate successivamente:

```
#include <iostream>
#include <fstream>
#include <cstdio>
#include "pass.h" //altro file .h che contiene solo la password
#include "dependencies/include/libpq-fe.h"

#define PG_HOST "127.0.0.1" // indirizzo del localhost
#define PG_USER "postgres"  // il vostro nome utente
#define PG_DB "Progetto"    // il nome del database
#define PG_PASS pass        // la password per accedere a pgadmin contenuta nel
                            // file pass.h
#define PG_PORT 5432

using namespace std;

PGconn *connection(char *conninfo)
{

    sprintf(conninfo, " user =%s password =%s dbname =%s hostaddr =%s port =%d",
    PG_USER, PG_PASS, PG_DB, PG_HOST, PG_PORT);

    PGconn *conn = PQconnectdb(conninfo);

    if (PQstatus(conn) != CONNECTION_OK)
    {
        cout << " Errore di connessione \n"
             << PQerrorMessage(conn);
        PQfinish(conn);
        exit(1);
```

```
    }

    cout << " Connessione avvenuta correttamente " << endl;

    return conn;
}

void checkResults(PGresult *res, const PGconn *conn)
{
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        cout << " Risultati inconsistenti " << PQerrorMessage(conn);
        PQclear(res);
        exit(1);
    }
}

//funzioni per query
```

Nel file .cpp invece viene dichiarata una variabile `char conninfo` che viene poi passata alla funzione `connection`.

```
#include "header.h"

int main()
{
    char conninfo[250];
    PGconn *con = connection(conninfo);
    //resto del codice con chiamate di funzioni query
}
```

Codice C++ per query

Sempre per sfruttare la programmazione modulare, tutte le query sono effettuate in funzioni apposite, create nel file `header.h` che è già incluso nel file principale `query.cpp` che avrà quindi al suo interno solo la chiamata di funzione.

Qua sotto un esempio di funzione all'interno del file `header.h` per la query n.1:

```
void UtentiAzienda(PGconn *conn, PGresult *res) // query 1
{
```

```
    res = PQexec(conn, "SELECT u.nome, u.azienda, u.pIva FROM Utente as u WHERE pIva I
S NOT NULL AND u.pIva <> '' GROUP BY u.nome, u.azienda, u.pIva");
    checkResults(res, conn);
    int tuple = PQntuples(res);
    int campi = PQnfields(res);
    printIntestazione(campi, res);
    printValue(tuple, campi, res);
}
```

INDEX

Viene creato un index con la seguente sintassi:

```
CREATE INDEX Prodotto_Magazzino ON Prodotto(Magazzino);
```

Più precisamente viene creato un indice sulla colonna `MAGAZZINO` della tabella `PRODOTTO` che velocizzerà lo svolgimento delle query, in particolare quella della ricerca dei prodotti all'interno di un magazzino dato il suo `ID` o `Indirizzo`.

Nel nostro caso, dato l'esimio numero di dati all'interno della nostra base di dati, la differenza di tempo di esecuzione della query seguente è pari a 0.

```
SELECT p.nome
FROM prodotto as p
WHERE p.magazzino = 'id_magazzino' //dove id_magazzino indica l'id del magazzino che s
i vuole ricercare
```

Tuttavia, nel caso in cui il numero di tuple all'interno della base di dati fosse esponenzialmente più grande, il tempo di esecuzione sarebbe drasticamente diminuito, in quanto l'indice permette alla query di fare una ricerca efficiente dei magazzini contenuti nella base di dati riducendo il numero di record che questa deve scansionare.

In generale, più il volume di dati all'interno della base di dati è grande, più l'indice velocizzerà il processo di esecuzione della query.
Anche la complessità della query stessa influisce sull'utilità dell'indice.

Qui si può notare, seppur infinitesimale, la differenza di tempo tra l'utilizzo di un indice (foto 1) e non (foto 2)

Query con index

query senza index

Conclusioni

Il database progettato supporta le funzionalità necessarie per l'ecommerce di PC e tastiere, come la gestione del carrello, degli ordini, dei pagamenti e delle spedizioni.