ShoePal

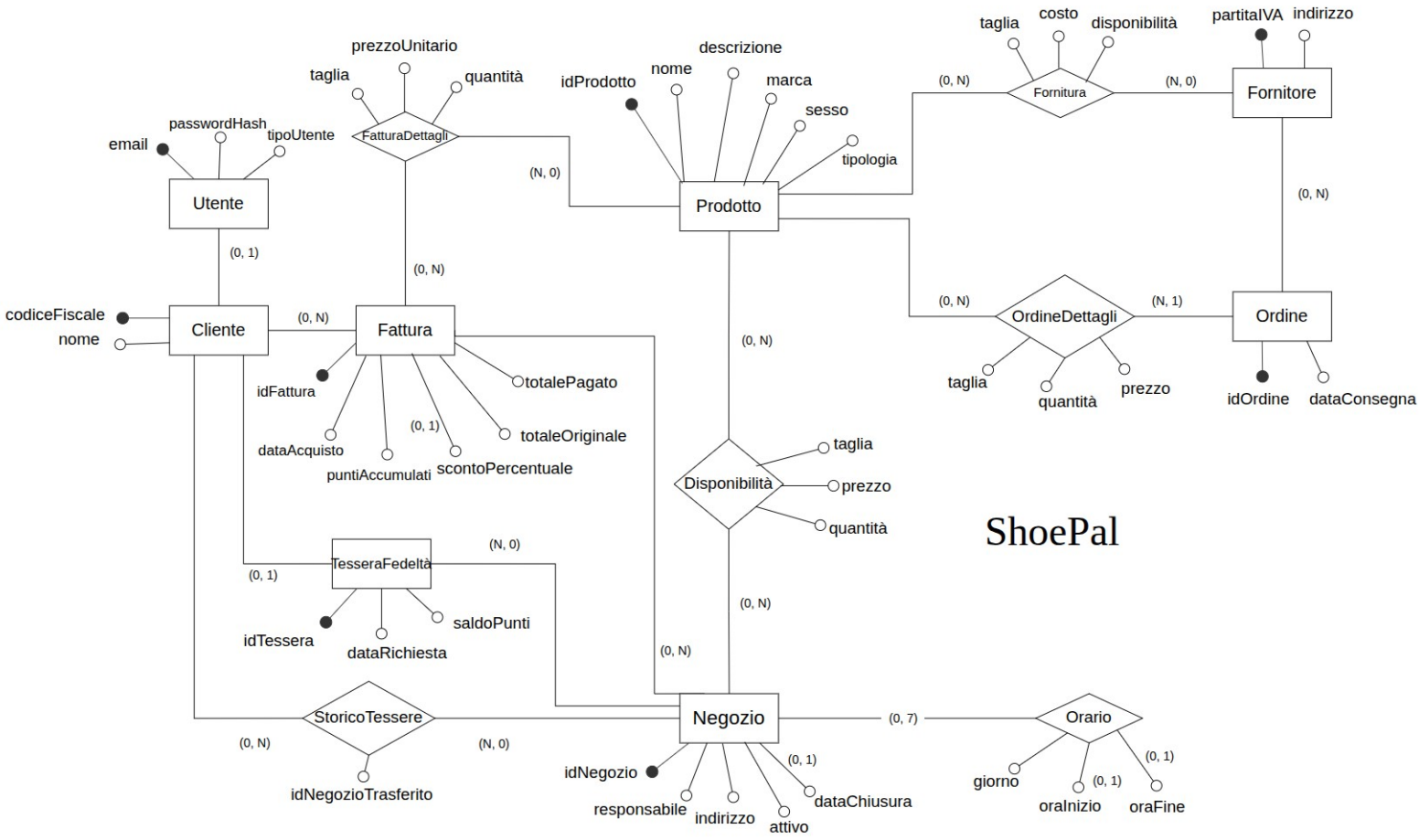
29407A Corrado Francesco

RELAZIONE PROGETTO BASI DI DATI

1. Descrizione progetto e schema ER
2. Schema Relazionale Logico
3. Viste, trigger e funzioni
4. Schermate Applicazione
5. Prove di funzionamento DB

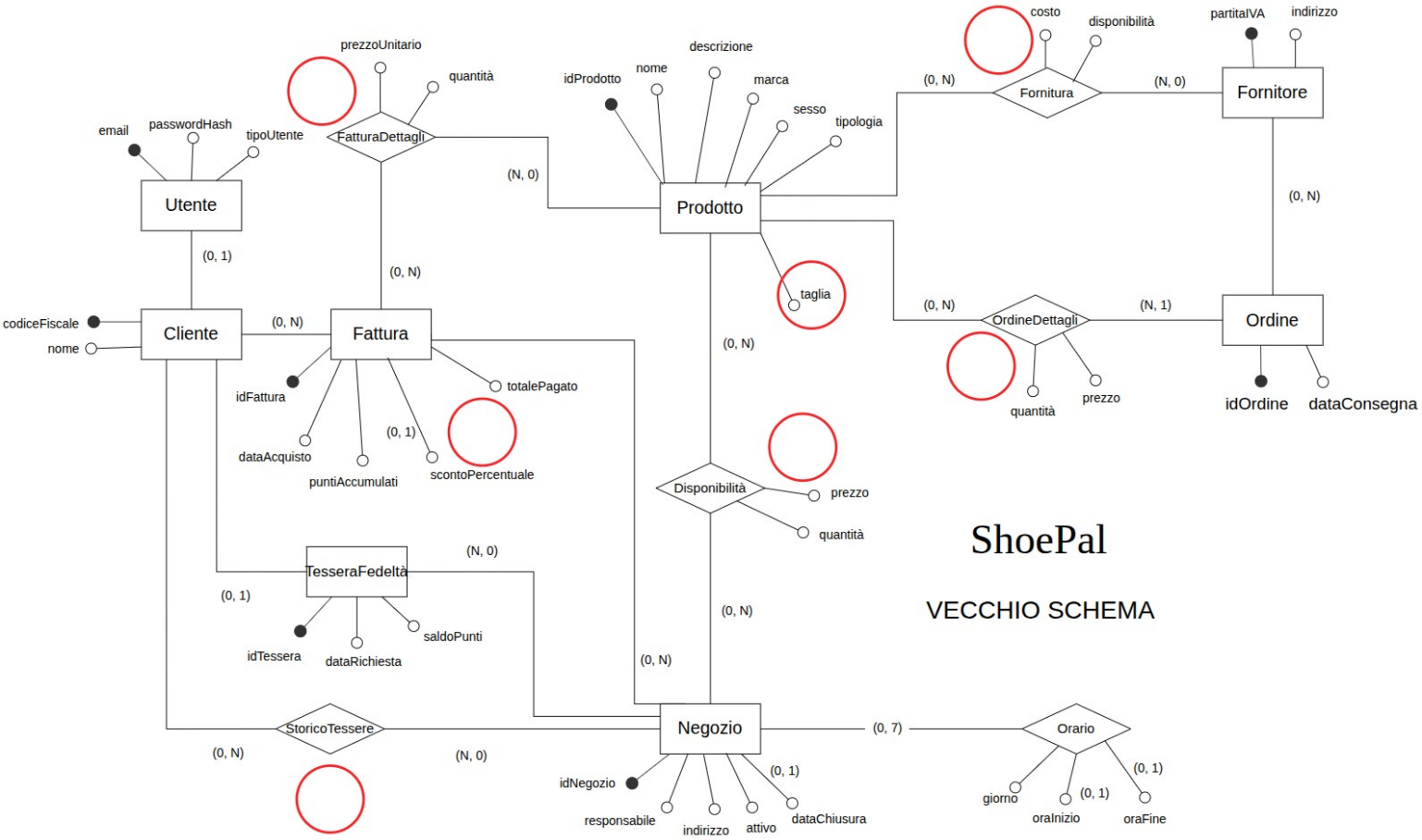
1. Descrizione Progetto e schema ER

ShoePal è un negozio online che vende scarpe. I clienti possono acquistare scarpe di molte diverse marche, taglie e tipologie. I clienti possono richiedere delle tessere fedeltà (associate ad un negozio) tramite le quali possono ottenere sconti spendendo soldi nei negozi. I Manager invece gestiscono i negozi, possono visualizzare informazioni varie sullo stato dei negozi, ordinare rifornimenti, chiudere ed aprire negozi, visualizzare e modificare informazioni sui clienti e sulle tessere dei negozi. Lo schema ER del progetto è il seguente:



Ogni utente (identificato da un'email e una password) può essere di due tipi, manager o cliente. Se l'utente è un manager esso può accedere liberamente all'interfaccia manager e avere pieno controllo. Se l'utente invece è un cliente, deve prima associare un cliente (codiceFiscale, nome) all'utente cliente così da poter accedere all'applicazione come cliente. Ogni negozio è legato ai suoi prodotti tramite una relazione di disponibilità dove sono salvati per ogni prodotto e taglia a quale prezzo sono venduti e in che quantità. Inoltre ogni negozio può avere degli orari, che rappresentano i giorni della settimana. I prodotti vengono riforniti dai manager tramite i fornitori: ogni fornitore ha una relazione fornitura con i prodotti che rappresenta i prodotti che ha a disposizione da vendere all'ingrosso, a che costo e in che disponibilità. Quando un manager effettua un ordine da un fornitore, viene salvato con un id e la data di consegna, mentre i dettagli dell'ordine sono salvati nella relazione specifica (salvando taglia quantità e prezzo). Se invece un cliente effettua un ordine tramite l'applicazione del negozio, viene creata una Fattura con tutte le informazioni (data, punti accumulati, totale pagato, ed eventualmente sconto e totale scontato) e i dettagli dell'acquisto sono salvati nella relazione fatturaDettagli salvando per ogni prodotto e taglia il prezzo e la quantità acquistata. Ogni cliente può richiedere ad un negozio una TesseraFedeltà (max 1 per cliente) sulla quale accumulare punti dopo ogni acquisto. Se un negozio viene chiuso, la tessera viene trasferita in uno storicotessere dove viene salvata, insieme all'eventuale id del negozio alla quale viene poi trasferita se ripristinata

Lo schema ER ha subito diverse modifiche e ristrutturazioni in corso d'opera perché realizzando l'applicazione mi sono reso conto di diverse mancanze o errori progettuali. Inizialmente lo schema era il seguente:



Nei cerchi rossi sono evidenziate le differenze. La differenza più sostanziale riguarda la gestione della taglia: inizialmente avevo inserito la taglia dentro prodotto come un qualsiasi attributo. Mi sono reso conto dopo però che questo era incredibilmente inefficiente, in quanto ad esempio per salvare un paio di scarpe in 6 taglie diverse, avrei dovuto ripetere per 6 volte le informazioni sulla

scarpa (nome, descrizione, prezzo ecc) solo per modificare la taglia. Ho preferito quindi gestire la taglia come una proprietà identificativa del prodotto: nello schema definitivo ho spostato la taglia all'interno di disponibilità, e di conseguenza anche nel resto dello schema come in ordineDettagli, Fornitura, Fattura e fatturaDettagli. In questo modo la logica è coerente in tutto lo schema ed è la seguente: nei prodotti ho una riga per ogni prodotto, mentre quando salvo le disponibilità di quel prodotto, o gli acquisti fatti per quel prodotto, uso anche la taglia per identificarli. Es se ho disponibili 5 paia di scarpe in taglia 39 40 e 41, il prodotto nella tabella Prodotto comparirà una volta sola, mentre in disponibilità ci saranno 3 righe che indicano la taglia e la disponibilità per la taglia.

Un'altra modifica è stata quella di inserire l'IDnegoziotrasferito nello storico tessere, così da fare in modo che quando una tessera viene ripristinata posso sapere dove è stata spostata

2. Schema Logico Relazionale

Lo schema logico del progetto è il seguente:

Negozio (**idNegozio**, responsabile, indirizzo, attivo, dataChiusura*)

- Data chiusura è annullabile perché serve solo se il negozio ha chiuso. Ogni negozio oltre ad un ID possiede un responsabile, un indirizzo ed uno stato attuale

Orario (**idNegozio**, **giorno**, oraInizio*, oraFine*)

- L'orario è identificato da idNegozio e giorno. OraInizio e oraFine sono annullabili perché se sono NULL significa che il negozio è chiuso.

Prodotto (**idProdotto**, nome, descrizione*, marca*, sesso*, tipologia*)

- I prodotti sono identificati da un ID e devono avere un nome: gli altri attributi come descrizione marca sesso e tipologia sono opzionali.

Disponibilità **(idNegozio, idProdotto, taglia**, prezzo, quantità)

- Disponibilità è usata per tenere traccia del magazzino per ogni negozio. Ad ogni negozio è associato un prodotto ed una taglia, così da poter tenere conto di prezzo e quantità di esso

Fornitore (**partitaIVA**, indirizzo)

- Ogni fornitore ha una partitaIVA ed un indirizzo

Fornitura (**partitaIVA, idProdotto, taglia**, costo, disponibilità)

- Fornitura tiene traccia del magazzino per ogni fornitore. Allo stesso modo di disponibilità, ad ogni fornitore è associato un prodotto ed una taglia, così da poter tenere conto di prezzo e quantità di esso

Ordine (**idOrdine**, partitaIVA, idNegozio, dataConsegna)

- Ogni ordine è identificato da un ID ed associa un negozio al fornitore presso il quale è stato effettuato l'ordine. Possiede anche una data di consegna

OrdineDettagli (**idOrdine, idProdotto, taglia**, quantità, prezzo)

- OrdineDettagli mantiene per ogni ordine i dettagli sui prodotti, quindi associa ad ogni ordine un prodotto in una taglia descrivendo la quantità ordinata ed a che prezzo

Utente (**email**, passwordHash, tipoUtente)

- Un utente identificato da un email possiede una password e un tipoUtente, per identificare se si tratta di un manager o un cliente

Cliente (**codiceFiscale**, nome, email)

- Ogni cliente identificato da un CodiceFiscale possiede un nome ed un email

TesseraFedeltà (**idTessera**, codiceFiscale, dataRichiesta, idNegozio, saldoPunti)

- Ogni TesseraFedeltà, identificata da un ID, è collegata ad un cliente tramite codiceFiscale ed a un negozio. Mantiene la data in cui è stata richiesta ed il saldo punti totale

StoricoTessere (**idTessera**, codiceFiscale, dataRichiesta, idNegozio, saldoPunti, idNegozioTrasferito*)

- Lo storico tessera è usato per mantenere le informazioni sulle tessere passate che ora sono archiviate. Mantiene le stesse informazioni della tessera fedeltà, ma ci associa anche eventualmente l'IdNegozioTrasferito per identificare se la tessera è stata ripristinata in un altro negozio

Fattura (**idFattura**, codiceFiscale, idNegozio, dataAcquisto, puntiAccumulati* scontoPercentuale*, totaleOriginale, totalePagato)

- Una fattura rappresenta un acquisto effettuato da un cliente. Identificata da un id associa un cliente ad un negozio, salvando la data dell'acquisto. Se vengono guadagnati punti sono salvati, se viene applicato uno sconto viene salvato anch'esso. Se lo sconto non viene applicato, totaleOriginale e totalePagato saranno uguali

FatturaDettagli (**idFattura, idProdotto, taglia**, quantità, prezzoUnitario)

- Mantiene i dettagli per ogni fattura: ad ogni fattura associa un prodotto in una taglia e tiene conto della quantità e del prezzo d'acquisto

DIPENDENZE FUNZIONALI

Le dipendenze funzionali dello schema sono le seguenti:

idNegozio → responsabile, indirizzo, attivo, dataChiusura
(idNegozio, giorno) → oraInizio, oraFine
idProdotto → nome, descrizione, marca, sesso, tipologia
(idNegozio, idProdotto, taglia) → prezzo, quantità
partitaIVA → indirizzo
(partitaIVA, idProdotto, taglia) → costo, disponibilità
idOrdine → partitaIVA, idNegozio, dataConsegna
(idOrdine, idProdotto, taglia) → quantità, prezzo
email → passwordHash, tipoUtente

codiceFiscale → nome, email
email → codiceFiscale (dato che è UNIQUE)
idTessera → codiceFiscale, dataRichiesta, idNegozio, saldoPunti
codiceFiscale → idTessera (dato chce è UNIQUE)
idTessera → codiceFiscale, dataRichiesta, idNegozio, saldoPunti, idNegozioTrasferito
idFattura → codiceFiscale, idNegozio, dataAcquisto, puntiAccumulati, scontoPercentuale, totaleOriginale, totalePagato
(idFattura, idProdotto, taglia) → quantità, prezzoUnitario

Date queste dipendenze, possiamo affermare che lo schema si trova in BCNF: infatti, tutte le dipendenze funzionali hanno come determinanti solo chiavi primarie o candidate, non ci sono dipendenze parziali o transitive, e i determinanti sono solo chiavi.

3. Viste, Trigger e Funzioni

Ho implementato all'interno del progetto diverse funzioni e trigger attivi, oltre ad alcune Viste.

VISTE

- **TesserePremium**: per mantenere tutti i clienti con più di 300 punti sulla tessera. Si tratta di una vista normale in quanto può essere facilmente calcolata al momento restando sempre aggiornata, lavorando con una query semplice
- **StatisticheVenditePerGiorno**: per mantenere il numero di fatture e l'incasso totale per ogni giorno. In questo caso si tratta di una lista materializzata, dato che esegue operazioni pesanti (SUM, COUNT) e su dati che non vengono aggiornati spesso (1 volta al giorno potenzialmente) quindi conviene fare un refresh della lista materializzata piuttosto che ricalcolare tutto ogni volta
- **StoricoOrdiniAlFornitore**: per mantenere per ogni fornitore gli ordini effettuati presso di lui. Si tratta di una vista normale in quanto può essere facilmente calcolata al momento e la query non è pesante

TRIGGER

- **trg_aggiorna_saldo_punti**: utilizza una funziona aggiorna saldo punti per gestire i punti durante un acquisto: verifica quanti punti vengono guadagnati, quanti punti vengono spesi per uno sconto e calcola il totale, lanciando un'eccezione in caso di errori (es. spendi più punti di quelli che hai)
- **trg_copia_tessere_in_storico**: utilizza una funzione copia_tessere_in_storico per copiare le tessere di un negozio in StoricoTessere quando quel negozio viene chiuso
- **trg_verifica_tessere**: utilizza una funzione trasferisci_punti_auto per trasferire automaticamente i punti da una tessera nello storico alla nuova tessera quando viene inserita una nuova tessera per lo stesso cliente

FUNZIONI

Alcune funzioni sono state implementate per modificare attivamente i dati del db o per calcolare funzioni di utilità quando necessario:

- **calcola_sconto**: calcola lo sconto in base ai punti
- **ripristina_tessere**: ripristina la tessera nello storico come tessere attiva in un negozio
- **sposta_prodotto_tra_negozi**: permette di spostare prodotti da un negozio ad un altro
- le funzioni già descritte nei trigger

Ho inoltre implementato molte funzioni di utilità (principalmente per visualizzare informazioni del db) che sono utilizzate dalla applicazione web. Ho preferito dividere la logica del db da quella dell'applicazione, quindi il più possibile ho cercato di creare funzioni interne che vengono chiamate da php piuttosto che inviare direttamente query tramite php. Alcune funzioni di utilità per ottenete dati sono ad esempio:

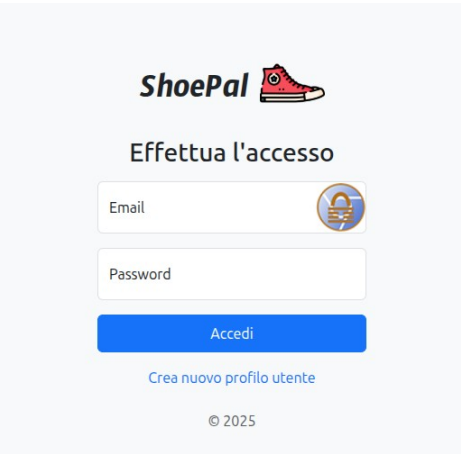
- **Funzioni per prodotti e fornitori**: es. get_all_prodotti_manager per ottenere informazioni sui prodotti, get_costo_minimo_prodotto per ottenere il costo minimo presso un fornitore, get_disponibilità_massima_prodotto per ottenere la massima disponibilità di un prodotto dai fornitori
- **Funzioni per disponibilità e negozi**: es. get_disponibilità_by_negozio ottiene tutte le disponibilità organizzate per negozio con i dettagli dei prodotti
- **Funzioni per fornitori e forniture**: es. get_all_fornitori ottiene tutti i fornitori, get_prodotti_più_ordinati_fornitore per ottenere informazioni su i prodotti più ordinati, get_furniture_by_fornitore per ottenere la fornitura di un fornitore specifico, get_forniture_per_taglia ottiene le forniture di un fornitore raggruppate per taglia, get_cronologia_ordini_fornitore ottiene la cronologia completa degli ordini del fornitore
- **Funzioni per gli ordini dei clienti**: es. get_order_preview per ottenere un'anteprima dell'ordine, get_ordini_raggruppati_per_ordine ottiene tutti gli ordini con info aggregate, get_dettagli_ordine ottiene i dettagli di un ordine specifico
- **Funzioni per clienti ed utenti**: es. get_all_clienti per ottenere tutti i clienti e le loro informazioni, get_all_utenti uguale ma per gli utenti, get_user_typer per verificare il tipo di utente, count_managers per contare il numero di managers nel sistema
- **Funzioni per tessere fedeltà**: es. get_tessere_attive per ottenere tutte le tessere attive, get_storico_tessere per ottenere tutte le tessere non più attive, get_clienti_tessere_per_negozio ottiene i clienti del negozio che possiedono tessere
- **Funzioni per statistiche e bilanci**: get_statistiche_vendite ottiene le vendite degli ultimi 30 giorni, get_tessere_premium ottiene le tessere premium, get_fatture_vendita_per_date ottiene le fatture per il periodo specificato, allo stesso modo get_rifornimenti_magazzino ma per gli ordini

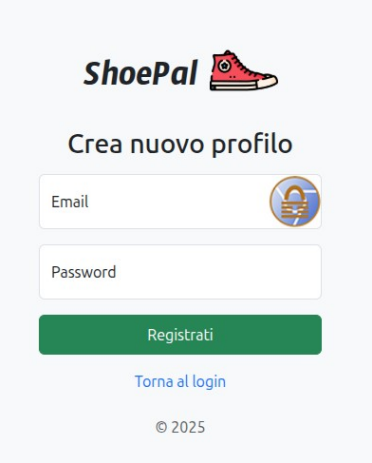
La maggior parte di queste funzioni sono solo di utilità per la applicazione web per visualizzare informazioni, ma ho preferito dichiararle direttamente del db per una questione di divisione logica come ho spiegato. Tutte le funzioni sono visualizzabili nel dump del database

4. Schermate Applicazione

LOGIN

Quando si effettua il login vi è la possibilità di loggarsi con il proprio account oppure creare un nuovo account. Quando si crea un nuovo account, occorre collegarlo ad un cliente: verrà chiesto al primo accesso

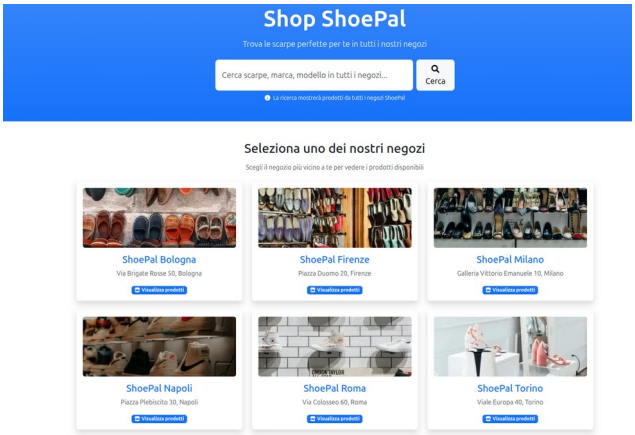


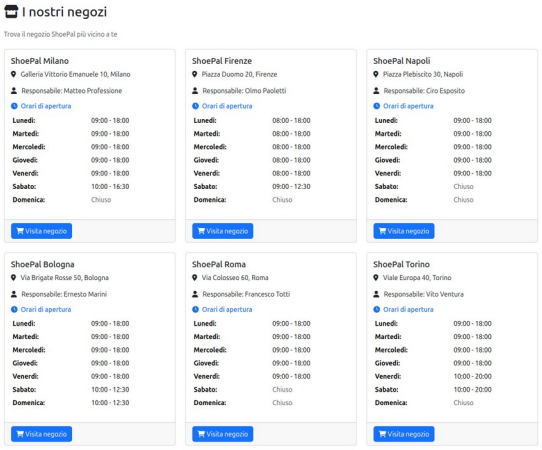


CLIENTE

- Visualizzazione dati

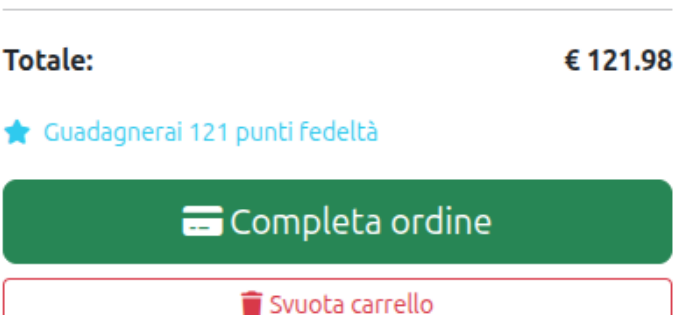
Le informazioni vengono correttamente estratte e visualizzate nell'applicazione, come ad esempio i negozi disponibili o gli orari di essi





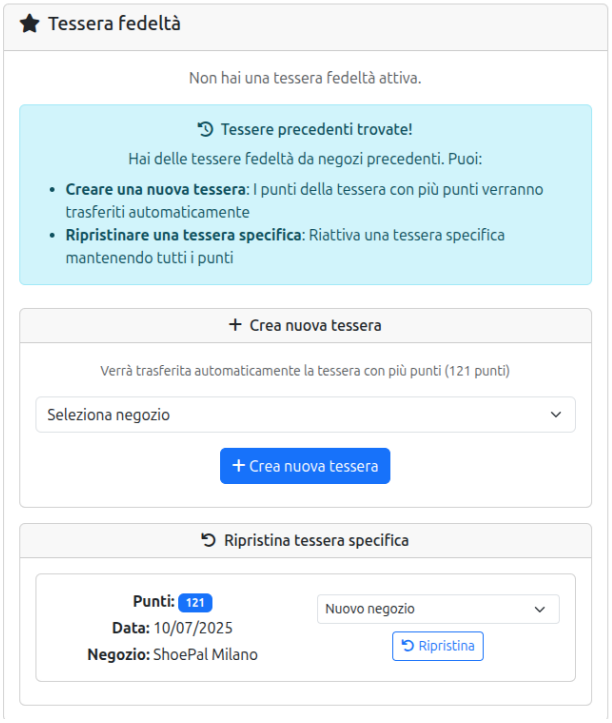
- Tessera Fedeltà

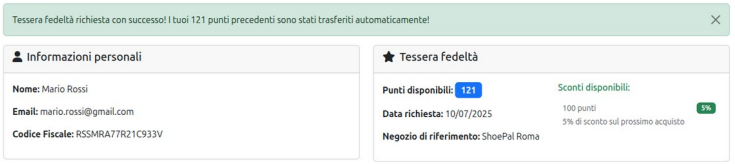
Quando una tessera non è ancora stata associata all'utente, comparirà la possibilità di richiederla. Una volta richiesta, dopo ogni acquisto i punti verranno aggiunti alla carta e saranno visualizzabili, oltre alla possibilità di vedere a quali sconti il cliente può accedere



- Ripristino Tessera

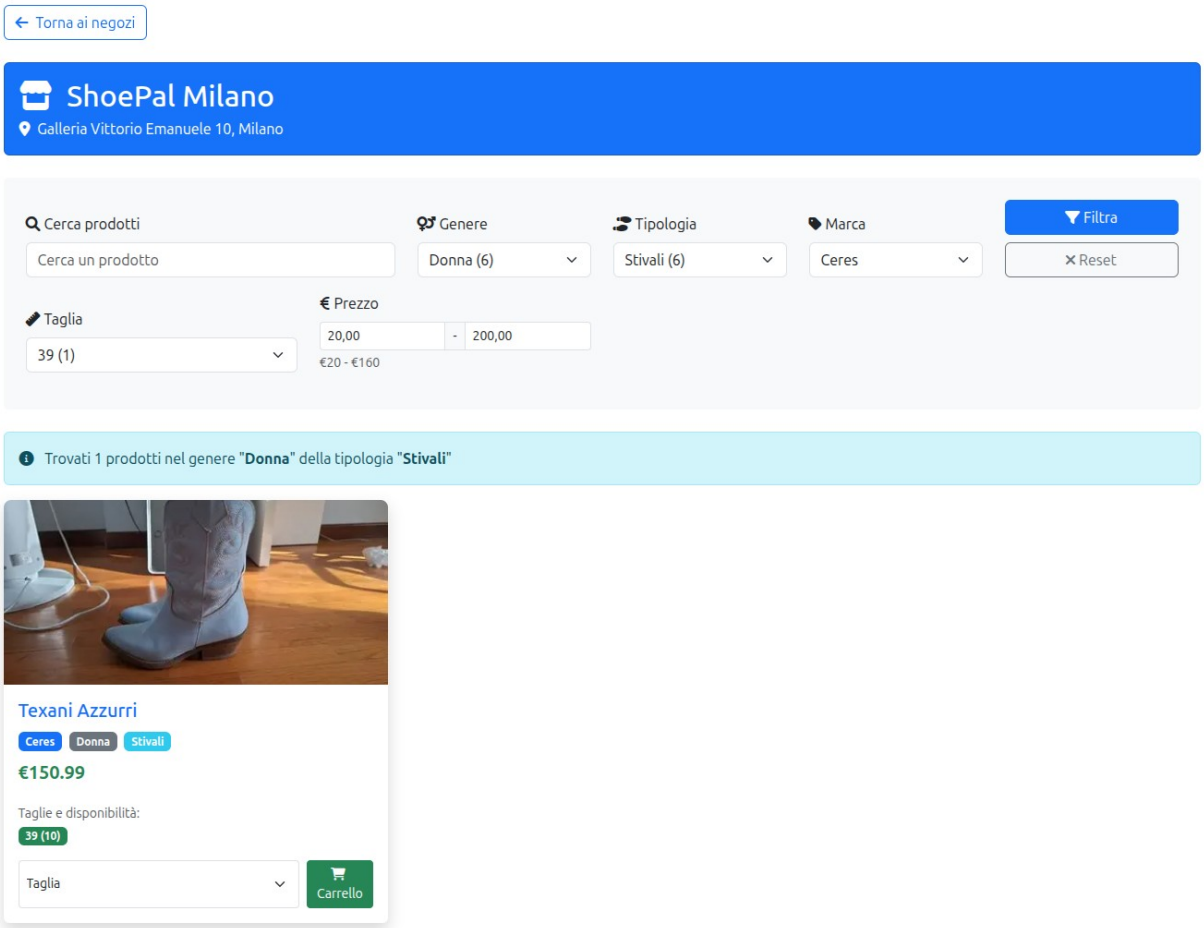
Se il cliente ha la tessera in un negozio ed esso viene chiuso, compare un messaggio nel suo profilo con due opzioni: o si crea una nuova tessera trasferendoci sopra i punti, oppure si ripristina la tessera in un altro negozio



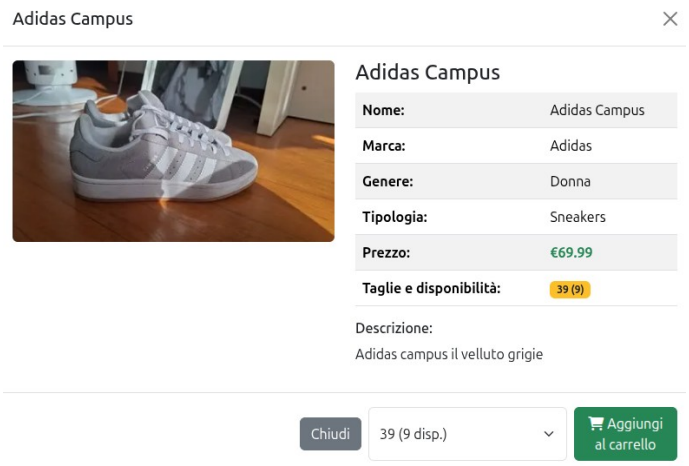


- Acquisti

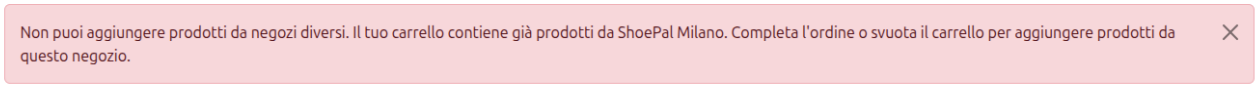
I prodotti possono essere cercati utilizzando i filtri per taglia marca genere ecc.



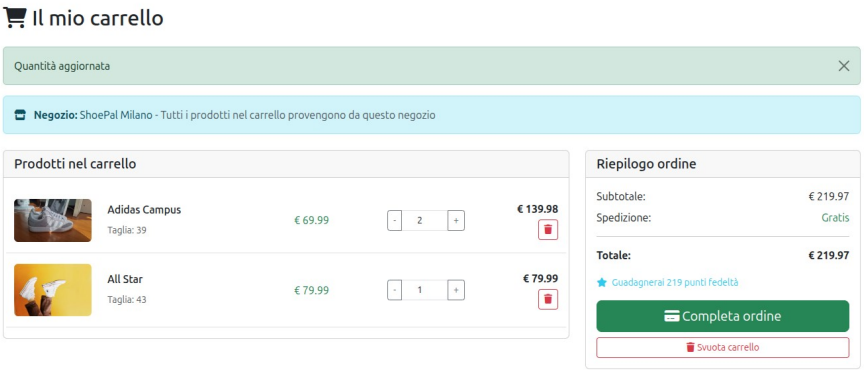
Quando clicco su un prodotto posso visualizzare le sue informazioni ed aggiungerlo al carrello

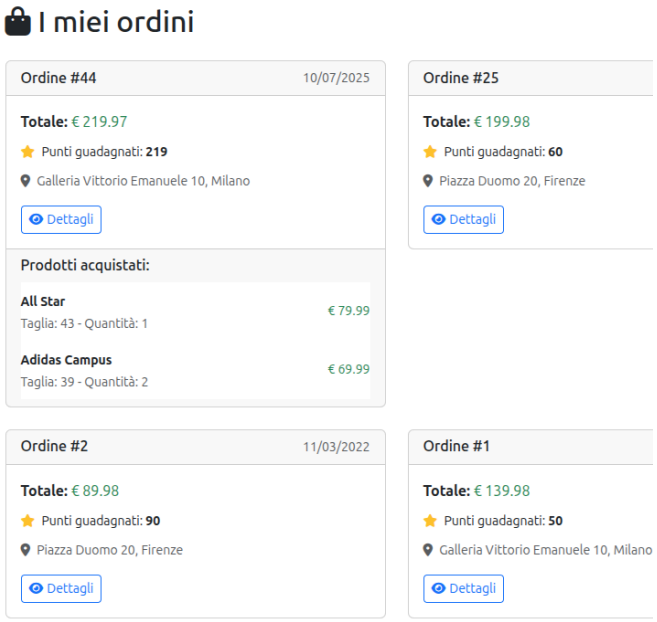


Dato che posso ordinare prodotti solo da un negozio alla volta, se provo ad ordinare un prodotto da un negozio quando ho già prodotti di un altro negozio nel carrello otterrò un errore

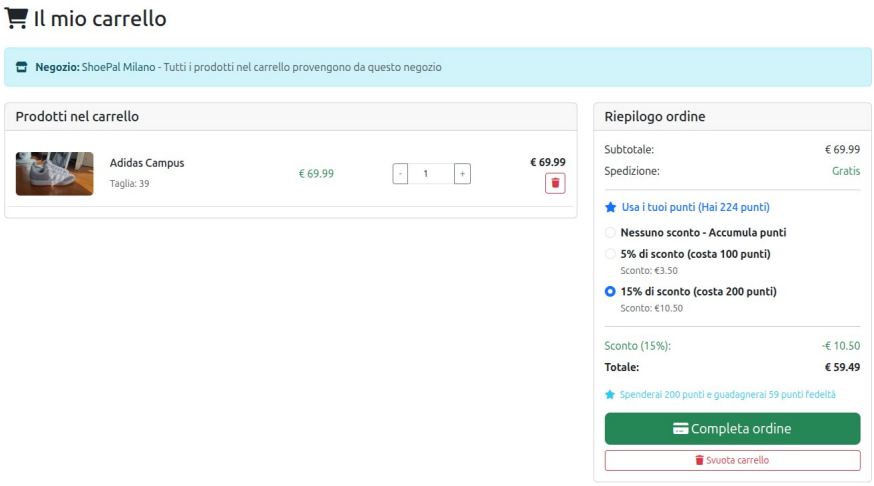


Una volta riempito il carrello, ed eventualmente rimosso o aggiornato quantità da esso, si può procedere al checkout visualizzando il totale e i punti guadagnati. Una volta che l'ordine viene completato, sarà visualizzabile (come i precedenti) nella sezione apposita





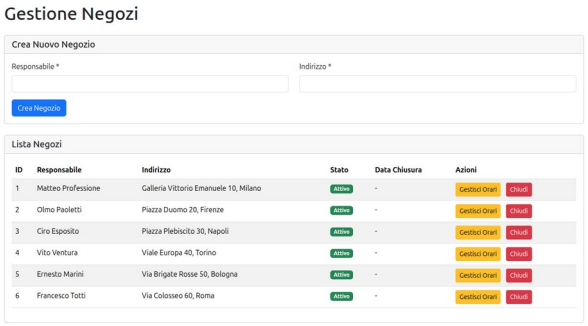
Se voglio effettuare un acquisto spendendo i punti della mia tessera, avendo i punti disponibili, al checkout comparirà l'opzione per utilizzarli. Utilizzandolo i punti saranno rimossi dalla carta e il prezzo totale sarà scontato

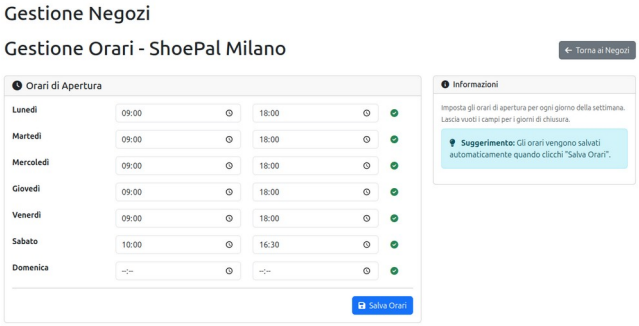


MANAGER

- Negozi

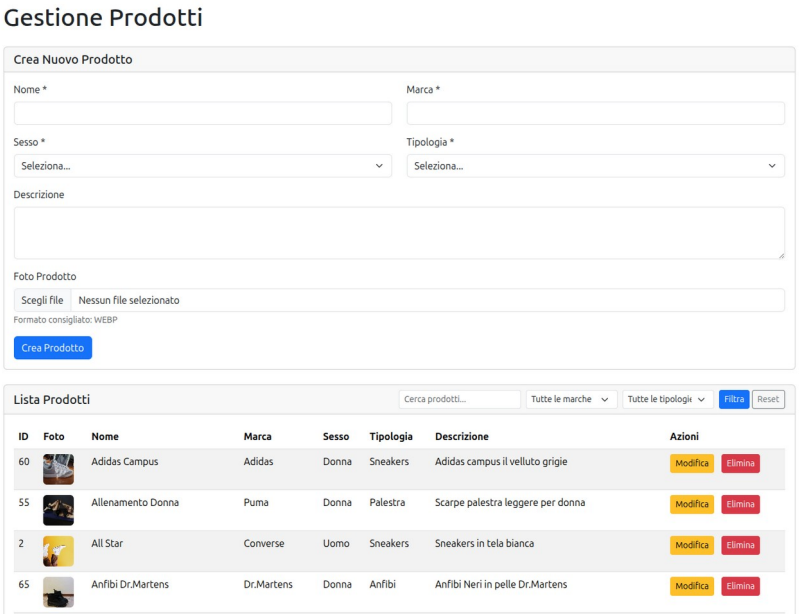
Nella sezione negozi i manager possono gestire aperture/chiusure dei negozi e i loro orari, oltre ad aggiungerne di nuovi





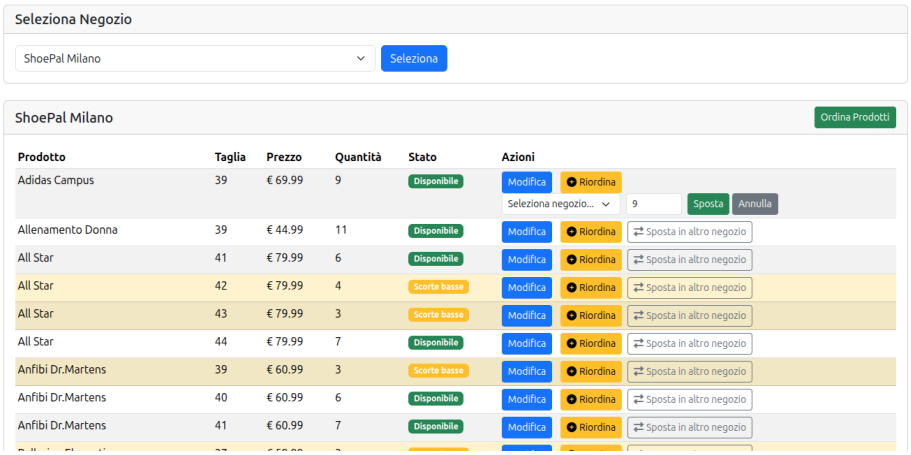
- Prodotti

Nella sezione prodotti i manager possono gestire le informazioni di tutti i prodotti ed aggiungerne altri



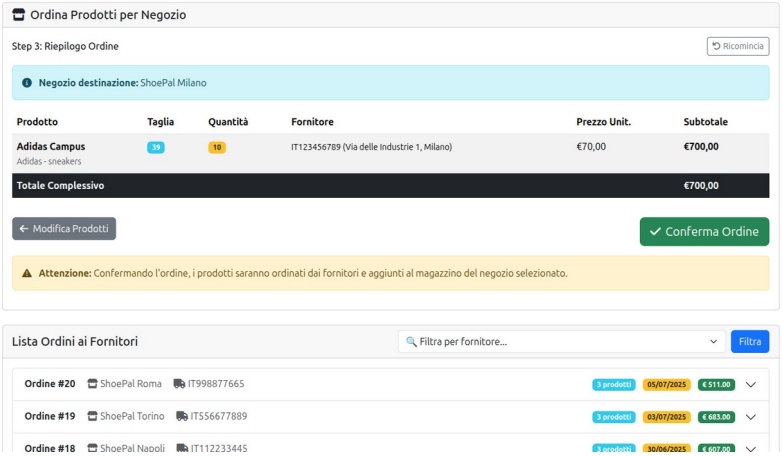
- Disponibilità

Nella sezione disponibilità i manager possono gestire per ogni negozio il magazzino, ordinando prodotti, modificandoli e spostandoli tra i negozi



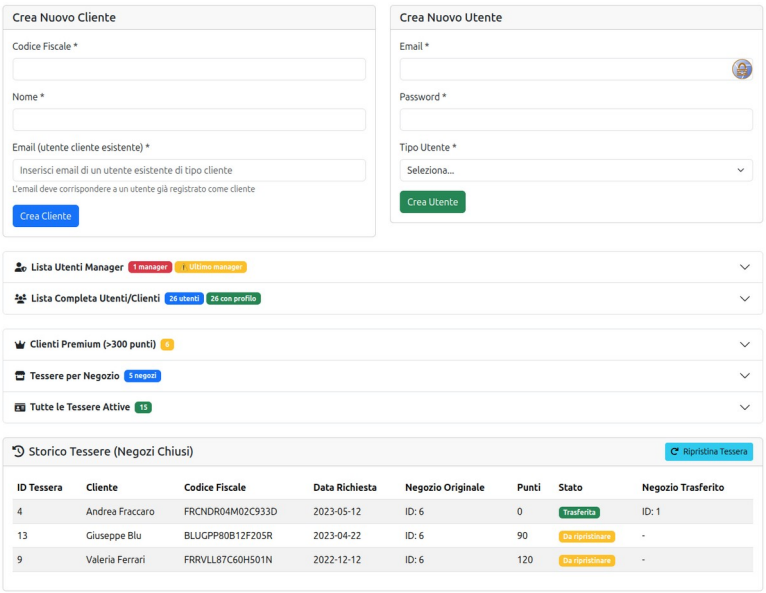
- Ordini

Nella sezione ordini i manager possono ordinare rifornimenti per i prodotti nei vari negozi e visualizzare gli ordini passati



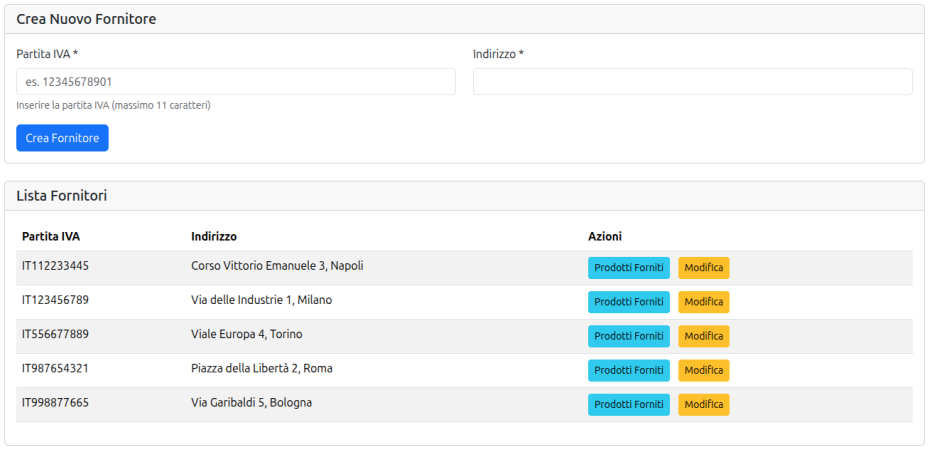
- Clienti

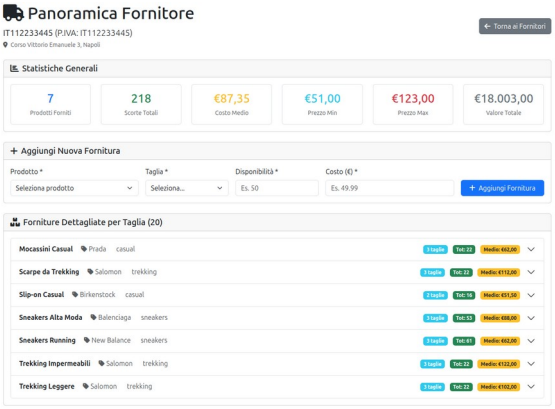
Nella sezione clienti i manager possono visualizzare le informazioni su tutti i clienti/utenti (oltre a poterli aggiungere e rimuovere) e possono modificarli



- Fornitori

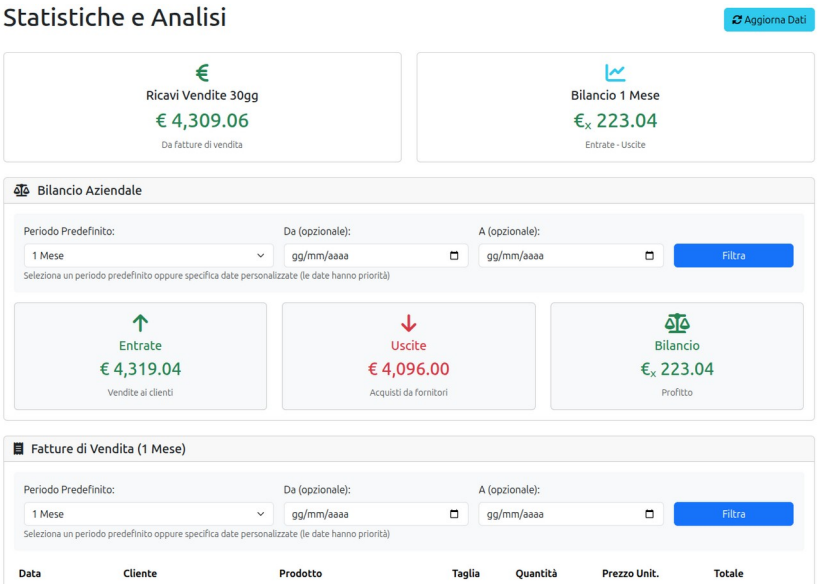
Nella sezione fornitori i manager possono visualizzare e modificate tutti i fornitori e per ognuno di essi informazioni sulla loro fornitura





- Statistiche

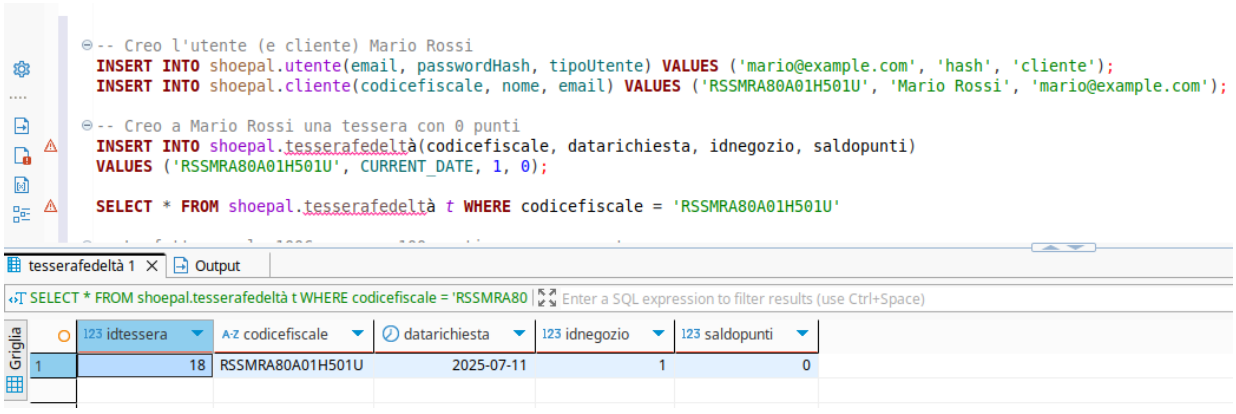
Nella sezione statistiche i manager possono visualizzare statistiche sulle vendite ed acquisti recenti o per un periodo specificato



5. Prove di Funzionamento DB

Vediamo alcuni esempi di esecuzione di alcune funzionalità e trigger precedentemente descritti:

- Trigger che aggiorna saldo punti dopo una fattura



```
-- Inserisco una fattura che vale 100€, genera 100 punti, nessuno sconto
INSERT INTO shoepal.fattura(codicefiscale, idnegozio, dataacquisto, puntiaccumulati, scontoPercentuale, totaleOriginale, totalePagato)
VALUES ('RSSMRA80A01H501U', 1, CURRENT_DATE, 100, NULL, 100, 100);

-- Verifico che il saldo punti sia 100
SELECT * FROM shoepal.tesserafedeltà WHERE codicefiscale = 'RSSMRA80A01H501U';
```

tesserafedeltà 1 | Output

SELECT * FROM shoepal.tesserafedeltà WHERE codicefiscale = 'RSSMRA80A | Enter a SQL expression to filter results (use Ctrl+Space)

Griglia

	idtessera	codicefiscale	datarichiesta	idnegozio	saldopunti
1	18	RSSMRA80A01H501U	2025-07-11	1	100

Provo un inserimento con sconto. Ho una tessera con 120 punti:

```
UPDATE shoepal.tesserafedeltà SET saldopunti = 120 WHERE codicefiscale = 'RSSMRA80A01H501U';

-- Acquisto da 60€, con 5% di sconto (3€), paga 57€, ottiene 57 punti
```

tesserafedeltà 1 | Output

SELECT * FROM shoepal.tesserafedeltà t WHERE codicefiscale = 'RSSMRA80 | Enter a SQL expression to filter results (use Ctrl+Space)

Griglia

	idtessera	codicefiscale	datarichiesta	idnegozio	saldopunti
1	18	RSSMRA80A01H501U	2025-07-11	1	120

```
-- Acquisto da 60€, con 5% di sconto (3€), paga 57€, ottiene 57 punti
INSERT INTO shoepal.fattura(codicefiscale, idnegozio, dataacquisto, puntiaccumulati, scontoPercentuale, totaleOriginale, totalePagato)
VALUES ('RSSMRA80A01H501U', 1, CURRENT_DATE, 57, 5, 60, 57);

-- Risultato: 120 - 100 + 57 = 77
SELECT * FROM shoepal.tesserafedeltà WHERE codicefiscale = 'RSSMRA80A01H501U';
```

tesserafedeltà 1 | Statistics 1 | Output

SELECT * FROM shoepal.tesserafedeltà WHERE codicefiscale = 'RSSMRA80A | Enter a SQL expression to filter results (use Ctrl+Space)

Griglia

	idtessera	codicefiscale	datarichiesta	idnegozio	saldopunti
1	18	RSSMRA80A01H501U	2025-07-11	1	77

Provo a fare in inserimento errato (uno sconto del 15% con solo 50 punti sulla carta):

```
UPDATE shoepal.tesserafedeltà SET saldopunti = 50 WHERE codicefiscale = 'RSSMRA80A01H501U';

-- Provo un inserimento con sconto del 15% (richiede 200 punti)
INSERT INTO shoepal.fattura(codicefiscale, idnegozio, dataacquisto, puntiaccumulati, scontoPercentuale, totaleOriginale, totalePagato)
VALUES ('RSSMRA80A01H501U', 1, CURRENT_DATE, 59, 15, 70, 59.5);
```

Statistics 1 | Output

```
Errore SQL [P0001]: ERROR: Punti insufficienti per applicare lo sconto. Servono 200 punti ma ne hai solo
50
  Dove: PL/pgSQL function aggiorna_saldo_punti() line 33 at RAISE

Error position:
```

- Trigger che trasferisce le tessere in storicotessere quando un negozio chiude

Attualmente la situazione tessere è la seguente: (in particolare le tessere del negozio 1)

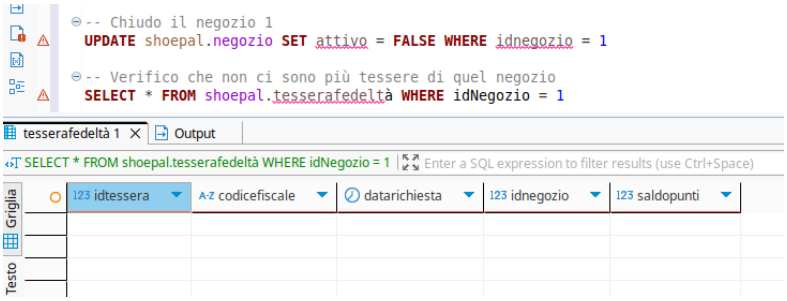
```
-- Vedo tutte le tessere associate al negozio 1
SELECT * FROM shoepal.tesserafedeltà WHERE idNegozio = 1
```

serafedeltà 1 | Output

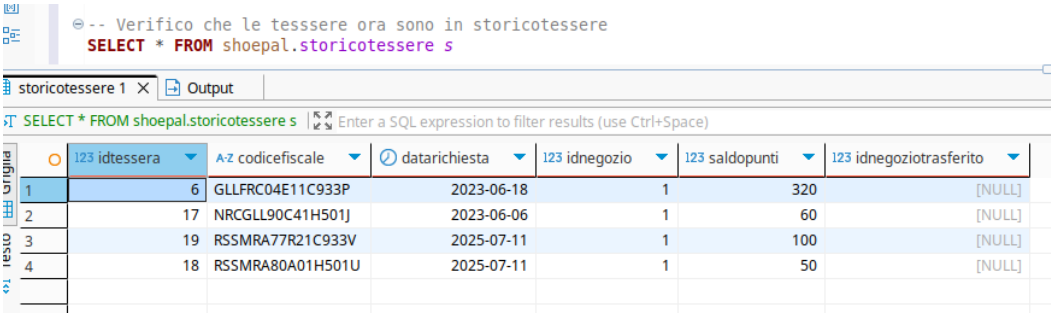
ECT * FROM shoepal.tesserafedeltà WHERE idNegozio = 1 | Enter a SQL expression to filter results (use Ctrl+Space)

idtessera	codicefiscale	datarichiesta	idnegozio	saldopunti
6	GLLFRC04E11C933P	2023-06-18	1	320
17	NRCGLL90C41H501J	2023-06-06	1	60
19	RSSMRA77R21C933V	2025-07-11	1	100
18	RSSMRA80A01H501U	2025-07-11	1	50

Chiudo il negozio 1 e verifico che non ci sono più tessere

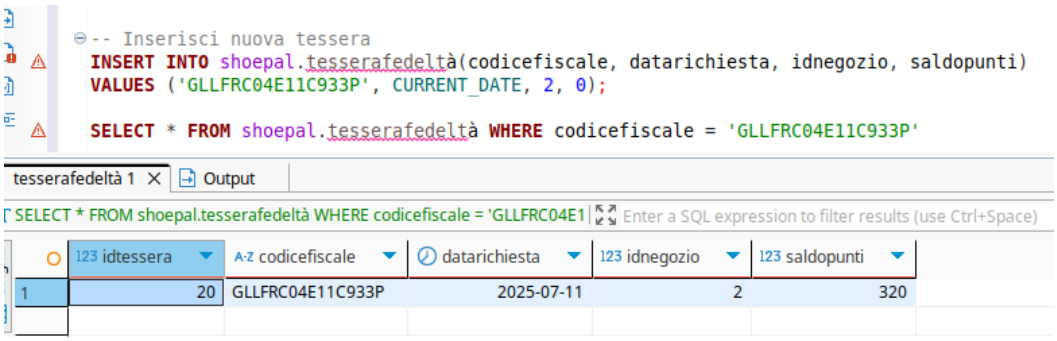


Verifico che le tessere ora si trovano in storicotessere:



- Trigger verifica tessere: trasferisce i punti quando creo una nuova tessera ad un utente con una tessera nell'archivio

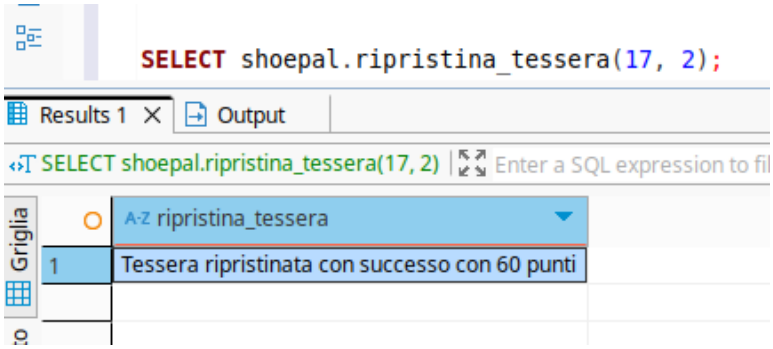
Prendiamo l'esempio precedente: vediamo che nell'archivio storico l'utente con codice fiscale 'GLLFRC04E11C933P' ha una tessera con 320 punti. Se quindi creo una nuova tessera all'utente, i punti saranno trasferiti lì sopra:



Ora l'utente ha una tessera nel negozio 2 sulla quale sono stati trasferiti i punti della precedente carta

- Ripristina Tessera: ripristina una tessera dell'archivio come una tessera attiva in un'altro negozio non chiuso

Sempre guardando l'esempio precedente vediamo che l'utente con codicefiscale 'NRCGLL90C41H501J' ha una tessera con ID 17 e 60 punti. Se ora chiamo la funzione ripristina_tessera:



```
SELECT * FROM shoepal.tesserafedeltà WHERE codicefiscale = 'NRCGLL90C41H501J';
```

esserafedeltà 1 | Output

SELECT * FROM shoepal.tesserafedeltà WHERE codicefiscale = 'NRCGLL90C | Enter a SQL expression to filter results (use

	123 idtessera	A-Z codicefiscale	datarichiesta	123 idnegozio	123 saldopunti
1	21	NRCGLL90C41H501J	2025-07-11	2	60

- Sposta prodotto tra negozi: sposto dei prodotti da un negozio ad un altro

Situazione iniziale:

```
-- Esempio: il negozio 1 ha 6 unità del prodotto 10 taglia '39', il negozio 2 invece ne ha 2
SELECT * FROM shoepal.disponibilità WHERE idnegozio = 1 AND idprodotto = 10 AND taglia = '39'
UNION
SELECT * FROM shoepal.disponibilità WHERE idnegozio = 2 AND idprodotto = 10 AND taglia = '39'
```

Results 1 | Output

SELECT * FROM shoepal.disponibilità WHERE idnegozio = 1 AND idprodotto | Enter a SQL expression to filter results (use Ctrl+Space)

	123 idnegozio	123 idprodotto	A-Z taglia	123 prezzo	123 quantità
1	1	10	39	119,99	6
2	2	10	39	121,99	2

Esecuzione:

```
-- Sposto 3 paia taglia 39 del prodotto 10 da negozio 1 a negozio 2
SELECT shoepal.sposta_prodotto_tra_negozi(1, 2, 10, '39', 3);
```

Results 1 | Output

SELECT shoepal.sposta_prodotto_tra_negozi(1, 2, 10, '39', 3) | Enter a SQL expression to filter results (use

	sposta_prodotto_tra_negozi
1	[v]

Risultato:

```
SELECT * FROM shoepal.disponibilità WHERE idnegozio = 1 AND idprodotto = 10 AND taglia = '39'
UNION
SELECT * FROM shoepal.disponibilità WHERE idnegozio = 2 AND idprodotto = 10 AND taglia = '39'
```

Results 1 | Output

SELECT * FROM shoepal.disponibilità WHERE idnegozio = 1 AND idprodotto = | Enter a SQL expression to filter results (use Ctrl+Space)

	123 idnegozio	123 idprodotto	A-Z taglia	123 prezzo	123 quantità
1	1	10	39	119,99	3
2	2	10	39	121,99	5

Inserimento errato:

```
-- Spostamento errato: provo a trasferire 10 paia taglia 39 del prodotto 10 da
-- negozio 1 a negozio 2 anche se ne possiedo solo 3
SELECT shoepal.sposta_prodotto_tra_negozi(1, 2, 10, '39', 10);
```

Results 1 | Output

SELECT shoepal.sposta_prodotto_tra_negozi(1, 2, 10, '39', 10) | Enter a SQL expression to filter results (use Ctrl+Space)

```
Errore SQL [23514]: ERROR: new row for relation "disponibilità" violates check constraint
"disponibilità_quantità_check"
  Dettaglio: Failing row contains (1, 10, 39, 119.99, -7).
  Dove: SQL statement "UPDATE shoepal.disponibilità
        SET quantità = v_nuova_quantita_origine
        WHERE idnegozio = p_idnegozio_origine
          AND idprodotto = p_idprodotto
          AND taglia = p_taglia"
PL/pgSQL function sposta_prodotto_tra_negozi(integer,integer,integer,character varying,integer) line 32 at
SQL statement
```