Progetto Basi di Dati
CakeLab

Simone Magagna
matr.: 1009467

Manuel Sgaravato
matr.: 1004557

1. Abstract

Il progetto in esame consiste nella realizzazione di una base di dati per la gestione di un e-commerce di prodotti alimentari artigianali. Viene creata un'interfaccia web per l'interazione del cliente con la base di dati, offrendo la possibilità di effettuare ordini sulla selezione di prodotti presentati con spedizione a domicilio. Per ogni prodotto sarà possibile visualizzarne la descrizione visiva e testuale, permettendo al cliente di conoscere appieno l'alimento che sta per acquistare. Vi saranno poi utenti speciali della base di dati che chiameremo admin, ai quali sarà affidata la gestione degli ordini registrati dai clienti.

2. Analisi dei requisiti

Il progetto in esame vuole modellare una base di dati che gestisca la vendita di prodotti alimentari artigianali tramite un sito web. I clienti interagiscono con tale interfaccia ordinando i prodotti proposti, mentre alcuni admin interagiranno con la base di dati per gestirne gli ordini.
L'entità principale che si vuole modellare sono i **Prodotti**. Di questi interessano il nome, il peso specifico, il prezzo di vendita, le porzioni, la sua disponibilità e la categoria cui appartiene.
Ogni prodotto appartiene ad una determinata **Categoria** (Crostate di frutta, Gastronomia eccetera) e di queste interessa sapere il nome.
La categoria caratterizza un prodotto e semplifica le operazioni di ricerca dei clienti.
Ad ogni prodotto è associata una **Descrizione** testuale: possiamo vederla come la sua carta d'identità; vi elenca i pregi e la lista degli ingredienti.
Vi sono inoltre alcuni **Media** associati ad ogni prodotto (foto e video) che mostrano in maniera completa ed esaustiva il prodotto finale.
I **Clienti** interessati a comprare i prodotti proposti possono farlo attraverso un account a loro assegnato. Durante la creazione dell'account è chiesto loro di inserire alcuni dati personali: lo user, che dovrà differire da ogni altro all'interno della base di dati e che non dovrà necessariamente essere uguale al nome vero del cliente, la password con la quale accedere al proprio account, il nome e cognome reali dell'utente e l' e-mail unica per ogni cliente. Ai fini della spedizione del prodotto saranno necessarie ulteriori informazioni: l'indirizzo, il comune, la provincia, il CAP e un contatto telefonico per il corriere.
Ogni cliente ha la possibilità di effettuare un **Ordine** e ogni ordine può comprendere più prodotti presentati nel sito. E' importante conoscere la sua data di registrazione, il suo importo totale, lo user del cliente che lo ha effettuato, lo stato.
Un modo interessante di vedere gli ordini è di distinguere **Ordini in corso** da **Ordini chiusi**; questi ultimi cessano di richiedere uno sforzo logistico ma sono di grande interesse amministrativo.
Microscopicamente un ordine in corso è un stato di "registrato" appena il cliente lo inoltra, e solo quando diviene "pagato", cioè quando un cliente salda l'importo totale un admin può prenderlo in carico(passandolo "in lavorazione") e finalmente spedirlo(ordine "chiuso").
E' importante offrire un certo livello di trasparenza come servizio al cliente, quindi ogni fase della lavorazione deve essere tracciabile dal cliente.
Inoltre per ragioni amministrative ad ogni ordine "pagato" è associato un **Pagamento**, mentre agli ordini chiusi sono associati i dati relativi alla **Spedizione**.
Gli **Amministratori** si occuperanno della gestione degli ordini e dell'inserimento dei nuovi prodotti all'interno del database. Di questi si vogliono conoscere lo user, la password associata, il nome, il cognome e l'e-mail.
Verranno rese disponibili più opzioni di **Pagamento**(bonifico bancario, carta di credito, PayPal, postagiro) e la data in cui è stato effettuato.
In fine delle **Spedizioni** è utile conoscere il tracking (codice di spedizione), il vettore utilizzato e le data di spedizione del prodotto.

3. Progettazione concettuale

3.1 Lista delle classi

Tutti gli attributi delle classi sono NOT NULL poiché sono elementi indispensabili per la gestione della base di dati, con nessuna ridondanza. I soli attributi che possono assumere valore NULL sono: pesoProdotto nella classe Prodotti, pagamentoAssOrdine nella classe Ordini, spedizioneAssOrdine nella classe Ordini, adminAssOrdine nella classe Ordini.

- **Utenti:** modella un generico utente.
  Attributi:
  - userUtenti varchar(255)
  - passwordUtenti char(32)
  - nomeUtenti varchar(255),
  - cognomeUtenti varchar(255)
  - mailUtenti varchar(255)

  Sono definite le seguenti sottoclassi di Utenti (con vincolo partizionamento):

  1. **Clienti:** modella un utente cliente.
     Attributi:
     - indirizzoCliente varchar(255)
     - comuneCliente varchar(255)
     - provCliente char(2)
     - capCliente char(5)
     - cellCliente varchar(255)

  2. **Admin:** modella un utente amministratore. Gli attributi che interessano di un admin sono gli stessi che per un utente generico.

- **Ordini:** modella un generico ordine inoltrato da un cliente.
  Attributi:
  - annoOrdine int
  - dataOrdine timestamp
  - importoOrdine int
  - clienteOrdine int

  Sono definite le seguenti sottoclassi di Ordini (con vincolo partizionamento):

  1. **Ordini In Corso:** specifica un ordine non ancora chiuso da un pagamento e una spedizione.
     Attributi:
     - pagamentoAssOrdine int

  2. **Ordini Chiusi:** specifica un ordine completato da un pagamento da parte del cliente e da una spedizione.
     Attributi:
     - spedizioneAssOrdine int

- **Pagamenti:** modella il tipo di pagamento utilizzato dal cliente per saldare l'ordine.
  Attributi:
  - tipoPagamento int
  - dataPagamento timestamp

- **Spedizioni:** modella una spedizione a domicilio.
  Attributi:
  - trackingSpedizione varchar(255)
  - vettoreSpedizione varchar(255)
  - dataSpedizione timestamp

- **Prodotti:** modella un generico prodotto offerto.
  Attributi:
  - nomeProdotto varchar(255)
  - prezzoProdotto decimal(7,2)
  - pesoProdotto decimal(7,2)
  - porzioniProdotto int
  - categoriaProdotto int
  - disponibilitaProdotto int
  - descrizioneAssProdotto int

- **Categoria:** specifica le categorie a cui appartengono i prodotti:
  Attributi:
  - nomeCategoria varchar(255)

- **Media:** modella la classe dei media utilizzati per presentare ai clienti il prodotto.
  Attributi:
  - tipoMedia int
  - linkMedia varchar(255)
  - prodottoMedia int

- **DescrizioniProdotti:** contiene le varie descrizioni dei prodotti offerti.
  Attributi:
  - descrizione varchar(500)

3.2 Lista delle associazioni

Clienti-Ordini in Corso: inoltra.
Ad ogni ordine in corso è associato uno ed un solo cliente mentre un cliente può effettuare più ordini. La molteplicità dunque è: 1:N. L'associazione è parziale da Clienti verso Ordini in Corso, totale da Ordini in Corso verso Clienti poiché un cliente può essere registrato ma non effettuare ordini, ma un ordine deve essere stato inoltrato da un cliente.

Admin-Ordini: gestisce.
Un Admin può gestire più ordini mentre un singolo ordine può essere gestito da un solo amministratore. La molteplicità dunque è: 1:N. L'associazione è totale verso gli Admin e parziale negli Ordini poiché un admin può anche non gestire alcun ordine mantre un ordine deve essere gestito da un admin.

Ordini in Corso-Pagamenti: è saldato da.
Un ordine in corso è saldato da un solo tipo di pagamento mentre un pagamento può saldare più ordini in corso. La molteplicità è dunque: N:1. L'associazione è totale negli Ordini in Corso dato che un pagamento deve essere associato ad un ordine inoltrato, mentre è parziale verso Pagamenti perché un ordine in corso non necessariamente dovrà essere pagato.

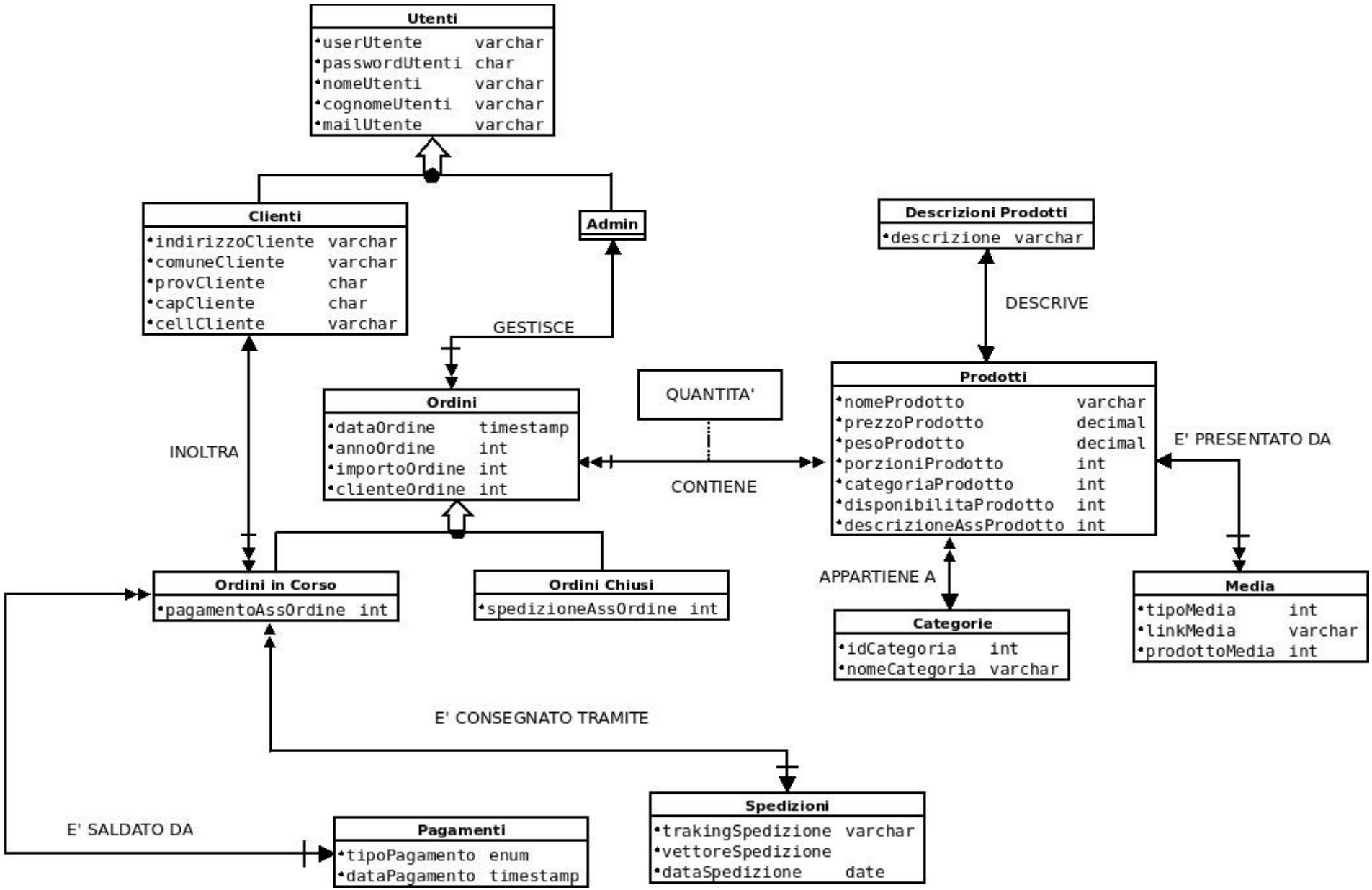
Ordini in Corso-Spedizioni: è consegnato tramite.
Un ordine in corso può essere spedito tramite un solo tipo di spedizione, mentre una singola spedizione può contenere più ordini. La molteplicità è dunque: N:1. L'associazione è parziale verso Spedizioni poiché un ordine può non venire più pagato da un cliente e perciò non verrà spedito, mentre è totale verso Ordini in Corso.

Ordini-Prodotti: contengono.
Un singolo ordine può contenere più prodotti e un prodotto può comparire in più ordini. La molteplicità è perciò: N:M. L'associazione è parziale verso Ordini poiché un prodotto può non comparire in nessun ordine, mentre è totale verso Prodotti.

Descrizioni Prodotti-Prodotti: descrive.
Una descrizione presenta un solo prodotto e un prodotto è presentato da una sola descrizione. La molteplicità risulta perciò essere 1:1. L'associazione è totale da entrambe le parti poiché ad un prodotto deve essere allegata una descrizione e una descrizione ha senso solo se accompagna un prodotto in vendita.

Prodotti-Categorie: appartiene a.
Ad una singola categoria possono appartenere più prodotti, mentre un prodotto può appartenere ad una sola categoria. La molteplicità dunque è: N:1. L'associazione risulta essere totale verso categorie e parziale verso prodotti, può esistere una categoria che per ragioni logistiche rimane senza prodotti in un lasso di tempo.

Prodotti-Media: è presentato da.
Un particolare media descrive solo un particolare prodotto, mentre ad un prodotto potranno essere associati più media. La molteplicità è perciò: 1:N. L'associazione è totale verso Prodotti poiché un media deve essere associato al prodotto che presenta, mentre un prodotto può non essere descritto da alcun media.


Utenti
•userUtente varchar
•passwordUtenti char
•nomeUtenti varchar
•cognomeUtenti varchar
•mailUtente varchar
Clienti
•indirizzoCliente varchar
•comuneCliente varchar
•provCliente char
•capCliente char
•cellCliente varchar
Admin
GESTISCE
Descrizioni Prodotti
•descrizione varchar
DESCRIVE
Ordini
•dataOrdine timestamp
•annoOrdine int
•importoOrdine int
•clienteOrdine int
QUANTITA'
CONTIENE
Prodotti
•nomeProdotto varchar
•prezzoProdotto decimal
•pesoProdotto decimal
•porzioniProdotto int
•categoriaProdotto int
•disponibilitaProdotto int
•descrizioneAssProdotto int
E' PRESENTATO DA
INOLTRA
Ordini in Corso
•pagamentoAssOrdine int
Ordini Chiusi
•spedizioneAssOrdine int
APPARTIENE A
Categorie
•idCategoria int
•nomeCategoria varchar
Media
•tipoMedia int
•linkMedia varchar
•prodottoMedia int
E' CONSEGNATO TRAMITE
Spedizioni
•trakingSpedizione varchar
•vettoreSpedizione
•dataSpedizione date
E' SALDATO DA
Pagamenti
•tipoPagamento enum
•dataPagamento timestamp

4. Progettazione logica

4.1 Gerarchie

La gerarchia della classe Utenti è stata tradotta tramite partizionamento orizzontale dato che la classe radice non ha delle associazioni proprie e le sottoclassi hanno numerosi attributi autonomi e associazioni proprie. Viene dunque eliminata la classe Utenti. La gerarchia completa viene quindi ridotta alle seguenti classi:

Clienti: essa eredita tutti i campi delle classi Utenti e Clienti, in più viene aggiunta una nuova chiave sintetica intera come chiave primaria:
- idCliente int <<PK>>

Admin: eredita tutti i campi di Utenti in più viene aggiunta una nuova chiave sintetica intera come chiave primaria:
- idAdmin int <<PK>>

La gerarchia della classe Ordini è stata tradotta tramite una relazione unica dato che le due sottoclassi differivano per pochissimi attributi. Vengono dunque eliminate le classi Ordini in Corso e Ordini Chiusi. La gerarchia completa viene quindi ridotta alla seguente classe:

Ordini: possiede tutti gli attributi che possedeva precedentemente nel modello ad oggetti più tutti gli attributi delle due sottoclassi:
- pagamentoAssOrdine int
- spedizioneassOrdine int

in più viene aggiunta una chiave sintetica come chiave primaria, un attributo di tipo enumerazione che indica lo stato dell'ordine(registrato, pagato, in lavorazione, chiuso) e due campi boleano che indicano rispettivamente se l'ordine è pagato/chiuso(cioè spedito) o meno:
- idOrdine int
- idInternoOrdine int
- statoOrdine enum
- isPagatoOrdine boolean
- isChiusoOrdine boolean

4.2 Chiavi sintetiche

Per agevolare una certa flessibilità nella stesura del codice e semplificare la gestione della base di dati sono state adottate le seguenti chiavi sintetiche: idAdmin nella classe Admin, idCliente nella classe Clienti, idCategoria nella classe Categorie, idDescrizione nella classe Descrizioni Prodotti, idProdotto nella classe Prodotti, idMedia nella classe Media, idPagamenti nella classe Pagamenti, idSpedizioni nella classe Spedizioni, idOrdine nella classe Ordini. Mentre abbiamo mantenuto la chiave 'classica' proveniente naturalmente dal modello relazionale nella tabella ORDINIPRODOTTI.

4.3 Associazioni

Clienti-Ordini: inoltra.
Ad ogni ordine in corso sarà associato uno ed uno solo cliente mentre un cliente può effettuare più ordini. La molteplicità dunque è: 1:N. L'associazione è parziale da Clienti verso Ordini in Corso, totale da Ordini in Corso verso Clienti poiché un cliente può essere registrato ma non effettuare ordini, ma un ordine deve essere stato inoltrato da un cliente.
Viene aggiunta una chiave esterna nella relazione Ordini verso Clienti:
- clienteOrdine int

Adimin-Ordini: gestisce.
Un Admin può gestire più ordini mentre un singolo ordine può essere gestito da un solo amministratore. La molteplicità dunque è: 1:N. L'associazione è totale verso gli Admin e parziale negli Ordini poiché un admin può anche non gestire alcun ordine mantre un ordine deve essere gestito da un admin.
Viene aggiunta una chiave esterna nella relazione Ordini verso Admin:
- adminAssOrdine int

Ordini in Corso-Pagamenti: è saldato da.
Un ordine in corso è saldato da un solo tipo di pagamento mentre un pagamento può saldare più ordini in corso. La molteplicità è dunque: N:1. L'associazione è totale negli Ordini in Corso dato che un pagamento deve essere associato ad un ordine inoltrato, mentre è parziale verso Pagamenti perché un ordine in corso non necessariamente dovrà essere pagato.
Viene aggiunta una chiave esterna nella relazione Ordini verso Pagamenti:
- pagamentoAssOrdine int

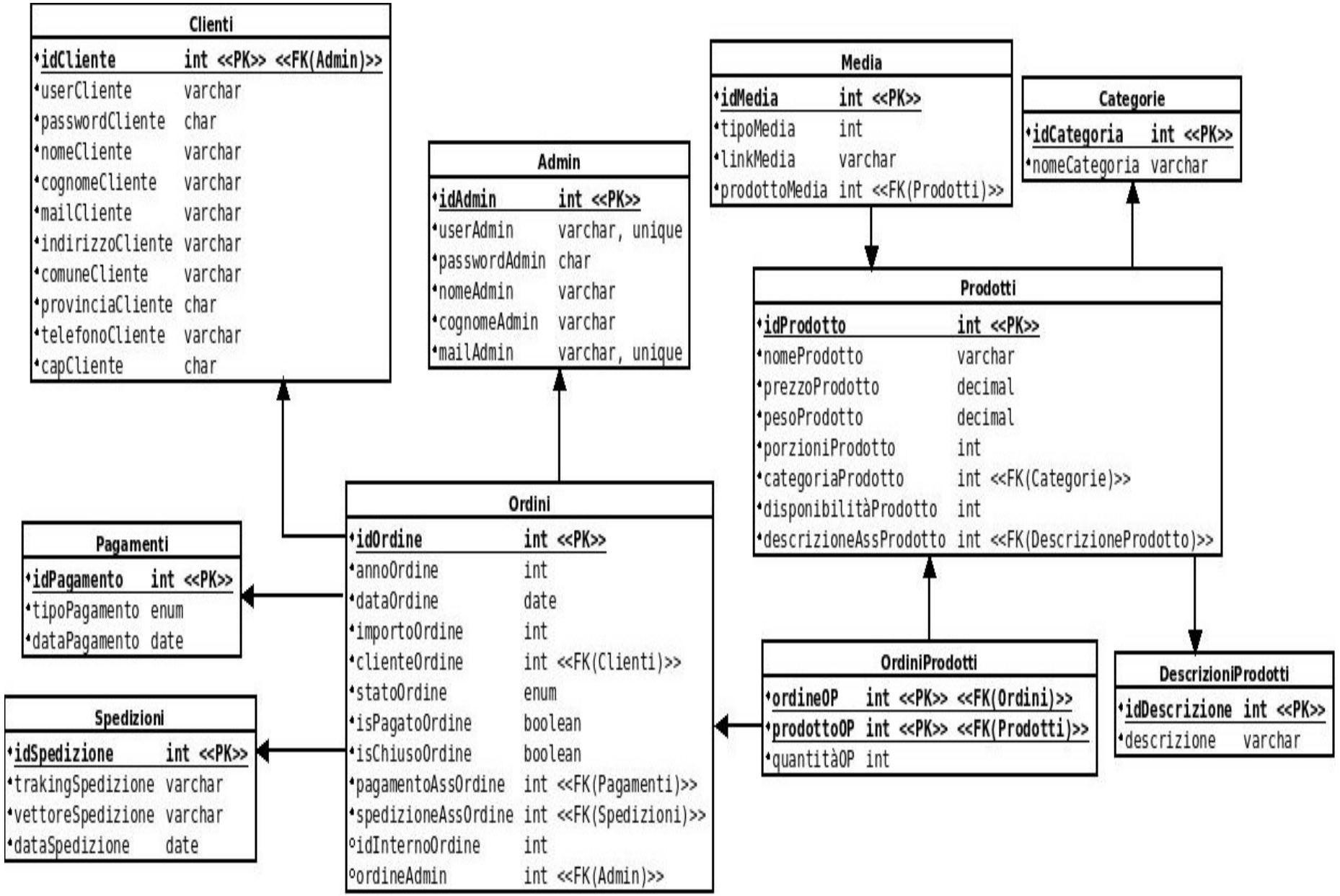
Ordini in Corso-Spedizioni: è consegnato tramite.
Un ordine in corso può essere spedito tramite un solo tipo di spedizione, mentre una singola spedizione può contenere più ordini. La molteplicità è dunque: N:1. L'associazione è parziale verso Spedizioni poiché un ordine può non venire più pagato da un cliente e perciò non verrà spedito, mentre è totale verso Ordini in Corso.
Viene aggiunta una chiave esterna nella relazione Ordini verso Spedizioni:
- spedizioneAssoOrdine int

Ordini-Prodotti: contengono.
Un singolo ordine può contenere più prodotti e un prodotto può comparire in più ordini. La molteplicità è perciò: N:M. L'associazione è parziale verso Ordini poiché un prodotto può non comparire in nessun ordine, mentre è totale verso Prodotti.
Data la molteplicità N:M viene creata una nuova relazione **OrdiniProdotti** con i seguenti attributi:
- ordineOP int <<PK>> <<FK(Ordini)>>
- prodottoOP int <<PK>> <<FK(Prodotti)>>
- quantitàOP int

Descrizioni Prodotti-Prodotti: descrive.
Una descrizione presenta un solo prodotto e un prodotto da una sola descrizione. La molteplicità risulta perciò essere. 1:1. . L'associazione risulta essere totale verso categorie e parziale verso prodotti, può esistere una categoria che per ragioni logistiche rimane senza prodotti in un lasso di tempo.
Viene aggiunta una chiave esterna nella relazione Prodotti verso Descrizione Prodotti:
- descrizioneAssOrdine int

Prodotti-Categorie: appartiene a.
Ad una singola categoria possono appartenere più prodotti, mentre un prodotto può appartenere ad una sola categoria. La molteplicità dunque è: N:1. L'associazione risulta essere totale da entrambe le parti dato che un prodotto deve appartenere ad una categoria e una categoria ha senso d'essere solo se ad essa appartiene almeno un prodotto.
Viene aggiunta una chiave esterna nella relazione Prodotti verso Descrizione Prodotti:
- categoriaProdotto int

Prodotti-Media: è presentato da.
Un particolare media descriverà solo un particolare prodotto, mentre ad un prodotto potranno essere associati più media. La molteplicità è perciò: 1:N. L'associazione è totale verso Prodotti poiché un media deve essere associato al prodotto che presenta, mentre un prodotto può non essere descritto da alcun media.
Viene aggiunta una chiave esterna nella relazione Media verso Prodotti:
- prodottoMedia int


Clienti
idCliente int <<PK>> <<FK(Admin)>>
userCliente varchar
passwordCliente char
nomeCliente varchar
cognomeCliente varchar
mailCliente varchar
indirizzoCliente varchar
comuneCliente varchar
provinciaCliente char
telefonoCliente varchar
capCliente char
Admin
idAdmin int <<PK>>
userAdmin varchar, unique
passwordAdmin char
nomeAdmin varchar
cognomeAdmin varchar
mailAdmin varchar, unique
Media
idMedia int <<PK>>
tipoMedia int
linkMedia varchar
prodottoMedia int <<FK(Prodotti)>>
Categorie
idCategoria int <<PK>>
nomeCategoria varchar
Prodotti
idProdotto int <<PK>>
nomeProdotto varchar
prezzoProdotto decimal
pesoProdotto decimal
porzioniProdotto int
categoriaProdotto int <<FK(Categorie)>>
disponibilitàProdotto int
descrizioneAssProdotto int <<FK(DescrizioneProdotto)>>
Ordini
idOrdine int <<PK>>
annoOrdine int
dataOrdine date
importoOrdine int
clienteOrdine int <<FK(Clienti)>>
statoOrdine enum
isPagatoOrdine boolean
isChiusoOrdine boolean
pagamentoAssOrdine int <<FK(Pagamenti)>>
spedizioneAssOrdine int <<FK(Spedizioni)>>
idInternoOrdine int
ordineAdmin int <<FK(Admin)>>
Pagamenti
idPagamento int <<PK>>
tipoPagamento enum
dataPagamento date
Spedizioni
idSpedizione int <<PK>>
trakingSpedizione varchar
vettoreSpedizione varchar
dataSpedizione date
OrdiniProdotti
ordineOP int <<PK>> <<FK(Ordini)>>
prodottoOP int <<PK>> <<FK(Prodotti)>>
quantitàOP int
DescrizioniProdotti
idDescrizione int <<PK>>
descrizione varchar

5. Implementazione della base di dati

Qui di seguito verrà riportato il codice relativo alla base di dati. Verranno definite le tabelle: ADMIN, CATEGORIE, CLIENTI, DESCRIZIONEPRODOTTI, ERRORI, MEDIA, ORDINI, ORDINIPRODOTTI, PAGAMENTI, PRDOTTI, SPEDIZIONI.
È stata implementata una tabella aggiuntiva ERRORI che contiene la descrizione degli errori rilevati dai trigger e dalle procedure.
La tabella è utile per innescare l'errore: quando un errore viene verificato lo si "genera" con un inserimento di record che è un doppione della chiave della riga corrispondente all'errore.

```
CREATE DATABASE IF NOT EXISTS `CakeLab`;

CREATE TABLE IF NOT EXISTS `CakeLab`.`ADMIN` (
  `idAdmin` INT NOT NULL AUTO_INCREMENT,
  `userAdmin` VARCHAR(255) NOT NULL ,
  `passwordAdmin` CHAR(32) NOT NULL ,
  `nomeAdmin` VARCHAR(255) NOT NULL ,
  `cognomeAdmin` VARCHAR(255) NOT NULL ,
  `mailAdmin` VARCHAR(255) NOT NULL ,
  PRIMARY KEY (`idAdmin`),
  unique (`userAdmin`),
  unique (`mailAdmin`) )
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `CakeLab`.`CLIENTI` (
  `idCliente` INT NOT NULL AUTO_INCREMENT,
  `userCliente` VARCHAR(255) NOT NULL ,
  `passwordCliente` CHAR(32) NOT NULL ,
  `nomeCliente` VARCHAR(255) NOT NULL ,
  `cognomeCliente` VARCHAR(255) NOT NULL ,
  `mailCliente` VARCHAR(255) NOT NULL ,
  `indirizzoCliente` VARCHAR(255) NOT NULL ,
  `comuneCliente` VARCHAR(255) NOT NULL ,
  `provCliente` CHAR(2) NOT NULL ,
  `capCliente` CHAR(5) NOT NULL ,
  `cellCliente` VARCHAR(255) NOT NULL ,
  PRIMARY KEY (`idCliente`) ,
  unique (`userCliente`),
  unique (`mailCliente`) )
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `CakeLab`.`CATEGORIE` (
  `idCategoria` INT NOT NULL AUTO_INCREMENT,
  `nomeCategoria` VARCHAR(255) NOT NULL ,
  unique (`nomeCategoria`),
  PRIMARY KEY (`idCategoria`) )
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `CakeLab`.`DESCRIZIONIPRODOTTI` (
  `idDescrizione` INT NOT NULL AUTO_INCREMENT,
  `descrizione` VARCHAR(500) NOT NULL ,
  PRIMARY KEY (`idDescrizione`) )
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `CakeLab`.`PRODOTTI` (
  `idProdotto` INT NOT NULL AUTO_INCREMENT,
  `nomeProdotto` VARCHAR(255) NOT NULL ,
  `prezzoProdotto` DECIMAL(7,2) NOT NULL ,
  `pesoProdotto` DECIMAL(7,2) NULL ,
  `porzioniProdotto` INT NOT NULL ,
  `categoriaProdotto` INT NOT NULL ,
  `disponibilitaProdotto` INT NOT NULL ,
  `descrizioneAssProdotto` INT NOT NULL ,
  PRIMARY KEY (`idProdotto`) ,
  unique (`nomeProdotto`),
  INDEX `fk_PRODOTTI_categoriaProdotto` (`categoriaProdotto` ASC) ,
  CONSTRAINT `fk_PRODOTTI_categoriaProdotto`
   FOREIGN KEY (`categoriaProdotto` )
   REFERENCES `CakeLab`.`CATEGORIE`(`idCategoria`)
   ON DELETE RESTRICT
   ON UPDATE CASCADE,
  INDEX `fk_PRODOTTI_descrizioneAssProdotto` (`descrizioneAssProdotto` ASC) ,
  CONSTRAINT `fk_PRODOTTI_descrizioneAssProdotto`
   FOREIGN KEY (`descrizioneAssProdotto`)
   REFERENCES `CakeLab`.`DESCRIZIONIPRODOTTI`(`idDescrizione`)
   ON DELETE RESTRICT
   ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `CakeLab`.`MEDIA` (
  `idMedia` INT NOT NULL AUTO_INCREMENT,
  `tipoMedia` INT NOT NULL ,
  `linkMedia` VARCHAR(255) NOT NULL ,
  `prodottoMedia` INT NOT NULL ,
  PRIMARY KEY (`idMedia`) ,
  INDEX `fk_MEDIA_prodottoMedia` (`prodottoMedia` ASC) ,
  CONSTRAINT `fk_MEDIA_prodottoMedia`
   FOREIGN KEY (`prodottoMedia` )
   REFERENCES `CakeLab`.`PRODOTTI`(`idProdotto`)
   ON DELETE CASCADE
   ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `CakeLab`.`PAGAMENTI` (
  `idPagamento` INT NOT NULL AUTO_INCREMENT,
  `tipoPagamento` ENUM('Bonifico Bancario', 'Carta di Credito', 'PayPal', 'Postagiro') NOT NULL ,
  `dataPagamento` TIMESTAMP NOT NULL DEFAULT NOW(),
  PRIMARY KEY (`idPagamento`) )
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `CakeLab`.`SPEDIZIONI` (
  `idSpedizione` INT NOT NULL AUTO_INCREMENT,
  `vettoreSpedizione` VARCHAR(255) NOT NULL ,
  `trackingSpedizione` VARCHAR(255) NOT NULL,
  `dataSpedizione` TIMESTAMP NOT NULL DEFAULT NOW(),
  PRIMARY KEY (`idSpedizione`) )
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `CakeLab`.`ORDINI` (
  `idOrdine` INT NOT NULL AUTO_INCREMENT,
  `idInternoOrdine` INT NOT NULL , -- nelle interfaccie web un ordine appare sempre identificato come num-anno. La chiave ufficiale però è idOrdine per abbassare la complessita'
  `annoOrdine` INT NOT NULL ,
  `dataOrdine` TIMESTAMP NOT NULL DEFAULT NOW() ,
  `importoOrdine` INT NOT NULL DEFAULT 0,
  `clienteOrdine` INT NOT NULL ,
  `statoOrdine` ENUM('Registrato', 'Pagato', 'In lavorazione', 'Chiuso') NOT NULL DEFAULT 'Registrato',
  `isPagatoOrdine` BOOLEAN NOT NULL DEFAULT FALSE,
  `isChiusoOrdine` BOOLEAN NOT NULL DEFAULT FALSE,
  `pagamentoAssOrdine` INT DEFAULT NULL ,
  `spedizioneAssOrdine` INT DEFAULT NULL ,
  `adminAssOrdine` INT DEFAULT NULL ,
  PRIMARY KEY (`idOrdine`) ,
  unique (`idInternoOrdine`,`annoOrdine`),
  INDEX `fk_ORDINI_clienteOrdine` (`clienteOrdine` ASC) ,
  CONSTRAINT `fk_ORDINI_clienteOrdine`
   FOREIGN KEY (`clienteOrdine`)
   REFERENCES `CakeLab`.`CLIENTI`(`idCliente`)
   ON DELETE RESTRICT
   ON UPDATE CASCADE,
  INDEX `fk_ORDINI_spedizioneAssOrdine` (`spedizioneAssOrdine` ASC) ,
  CONSTRAINT `fk_ORDINI_spedizioneAssOrdine`
   FOREIGN KEY (`spedizioneAssOrdine`)
   REFERENCES `CakeLab`.`SPEDIZIONI`(`idSpedizione`)
   ON DELETE SET NULL
   ON UPDATE CASCADE,
  INDEX `fk_ORDINI_adminAssOrdine` (`adminAssOrdine` ASC) ,
  CONSTRAINT `fk_ORDINI_adminAssOrdine`
   FOREIGN KEY (`adminAssOrdine`)
   REFERENCES `CakeLab`.`ADMIN`(`idAdmin`)
   ON DELETE RESTRICT
   ON UPDATE CASCADE,
  INDEX `fk_ORDINI_pagamentoAssOrdineI` (`pagamentoAssOrdine` ASC) ,
  CONSTRAINT `fk_ORDINI_pagamentoAssOrdineI`
   FOREIGN KEY (`pagamentoAssOrdine`)
   REFERENCES `CakeLab`.`PAGAMENTI`(`idPagamento`)
   ON DELETE SET NULL
   ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `CakeLab`.`ORDINIPRODOTTI` (
  `ordineOP` INT NOT NULL ,
  `prodottoOP` INT NOT NULL ,
  `quantitaOP` INT NOT NULL ,
  PRIMARY KEY (`ordineOP`,`prodottoOP`) ,
  CONSTRAINT `fk_ORDINIPRODOTTI_ordineOP`
   FOREIGN KEY (`ordineOP`)
   REFERENCES `CakeLab`.`ORDINI`(`idOrdine`)
   ON DELETE CASCADE
   ON UPDATE CASCADE,
  CONSTRAINT `fk_ORDINIPRODOTTI_prodottoOP`
   FOREIGN KEY (`prodottoOP`)
   REFERENCES `CakeLab`.`PRODOTTI`(`idProdotto`)
   ON DELETE RESTRICT
   ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `CakeLab`.`ERRORI` (
  `idErrore` INT NOT NULL AUTO_INCREMENT,
  `descErrore` VARCHAR(255) NOT NULL ,
  unique (`descErrore`),
  PRIMARY KEY (`idErrore`) )
ENGINE = InnoDB;
```

---

6. Triggers

Sono stati implementati alcuni triggers passivi e attivi per un mantenimento corretto della base di dati:

- **checkNewAdmin**: trigger passivo poiché verifica una condizione che, se non soddisfatta, determina il fallimento dell'inserimento.
  Il trigger assicura che l'inserimento di un nuovo admin non utilizzi uno username o un indirizzo di posta già registrato per un cliente.

- **checkNewCliente**: trigger passivo poiché verifica una condizione che, se non soddisfatta, determina il fallimento dell'inserimento.
  Il trigger assicura che l'inserimento di un nuovo cliente non utilizzi uno username o un indirizzo di posta già registrato per un admin.

- **updateImportoEQuantitaOrdine**: trigger attivo perchè modifica lo stato della base di dati.
  Aggiorna l'importo dell'ordine appena inserito, prodotto per prodotto, e mantiene il conto delle disponibilità.

- **updateQuantitaOnUpdateOrder**: trigger attivo perchè modifica lo stato della base di dati.
  Mantiene aggiornate le disponibilità dei prodotti quando vengono modificati ordini inoltrati.

- **checkUpdateProdotto:** trigger passivo perché non modifica lo stato della base di dati. Questo trigger assicura che a seguito di modifiche su prezzi e disponibilità di prodotti, non vi possano essere assegnazioni di valori negativi

Eccone il codice:

```
DROP TRIGGER IF EXISTS checkNewAdmin;
DELIMITER $
CREATE TRIGGER checkNewAdmin
BEFORE INSERT ON ADMIN FOR EACH ROW
BEGIN
DECLARE quanti integer;
SELECT count(*) INTO quanti
FROM CLIENTI
WHERE userCliente=NEW.userAdmin OR mailCliente=NEW.mailAdmin;
IF (quanti=1) THEN
        INSERT INTO ERRORI VALUES (2, "user o mail Admin riservata");
END IF;
END $
DELIMITER ;


DROP TRIGGER IF EXISTS checkNewCliente;
DELIMITER $
CREATE TRIGGER checkNewCliente
BEFORE INSERT ON CLIENTI FOR EACH ROW
BEGIN
DECLARE quanti integer;
SELECT count(*) INTO quanti
FROM ADMIN
WHERE userAdmin=NEW.userCliente OR mailAdmin=NEW.mailCliente;
 IF (quanti=1) THEN
INSERT INTO ERRORI VALUES (1, "user o mail Cliente riservata");
```

```
END IF;
END $
DELIMITER ;


DROP TRIGGER IF EXISTS updateImportoEQuantitaOrdine;
DELIMITER $$
CREATE TRIGGER updateImportoEQuantitaOrdine
AFTER INSERT ON ORDINIPRODOTTI FOR EACH ROW
BEGIN
DECLARE prezzo DECIMAL;
DECLARE conto DECIMAL;
CALL updQuantitaBody(NEW.prodottoOP, NEW.quantitaOP);
SELECT prezzoProdotto INTO prezzo FROM PRODOTTI WHERE IdProdotto = NEW.prodottoOP;
SET conto = prezzo * NEW.quantitaOP;
UPDATE ORDINI SET importoOrdine = importoOrdine + conto WHERE idOrdine = NEW.ordineOP;
END $$
DELIMITER ;


DROP TRIGGER IF EXISTS updateQuantitaOnUpdateOrder;
DELIMITER $
CREATE TRIGGER updateQuantitaOnUpdateOrder
AFTER UPDATE ON ORDINIPRODOTTI FOR EACH ROW
BEGIN
CALL updQuantitaBody(NEW.prodottoOP, NEW.quantitaOP);
END $
DELIMITER ;


CREATE TRIGGER checkUpdateProdotto ;
AFTER UPDATE ON PRODOTTI FOR EACH ROW
BEGIN
IF(NEW.prezzoProdotto<=0) THEN
INSERT INTO ERRORI(idErrore, descErrore)
VALUES (13, 'impossibile registrare un prezzo prodotto negativo o nullo');
END IF;
IF(NEW.disponibilitaProdotto<0) THEN
INSERT INTO ERRORI(idErrore, descErrore)
VALUES (5, 'impossibile registrare una disponibilita\' prodotto negativa');
END IF;
END $
DELIMITER ;
```

---

7. Funzioni e procedure

Le funzioni e le procedure utilizzate per i vari inserimenti e cancellazioni nella base di dati:

- **addAdim**: procedura che inserisce un nuovo cliente.
- **addCliente**: procedura che inserisce nella tabella CLIENTI un nuovo cliente aggiornandone tutti i campi.
- **addOrdine**: procedura che inserisce un nuovo ordine nella classe ORDINI.
- **addPagamento**: procedura che registra il pagamento per un dato ordine. Se l'ordine e' già stato pagato l'operazione fallisce.
- **addProdotto**: procedura che inserisce un nuovo prodotto in PRODOTTI e la sua relativa descrizione in DESCRIZIONEPRODOTTI.
- **addSpedizione**: procedura che registra la spedizione per un dato ordine. Se l'ordine non e' già stato pagato l'operazione fallisce.
- **delAdmin**: procedura che elimina un admin, se questo non è coinvolto nella tabella ORDINI.
- **delCliente**: procedura che elimina un cliente, se questo non è coinvolto nella tabella ORDINI.
- **delOrdine**: procedura che cancella un ordine dal database.
- **delProdotto**: procedura che elimina un prodotto, se questo non è coinvolto in altre tabelle.
- **tryLogin**: procedura che tenta un Login dato uno userUtente e una password già criptata.
- **updAdmin**: procedura che modifica un admin.
- **updCliente**: procedura che modifica un cliente.
- **updDispProdotto**: procedura che aggiorna la disponibilità di un prodotto nella tabella PRODOTTI.
- **updQuantitaBody**: procedura che viene richiamata dai triggers updateQuantitaOnUpdateOrder e updateImportoEQuantitaOrdine e aggiorna la disponibilità dei prodotti a seguito di un ordine.
- **superdelCliente**: procedura che elimina un cliente, e tutti gli ordini ad esso associati.
- **SPLIT_STRING**: funzione che scorre gli elementi di una stringa separati da un delimitatore.

Di seguito il codice:

```
DROP PROCEDURE IF EXISTS addAdmin;
DELIMITER $
CREATE PROCEDURE addAdmin
(IN newUserAdmin VARCHAR(255), IN newPasswordAdmin CHAR(32), IN newRipetiPasswordAdmin CHAR(32),
IN newNomeAdmin VARCHAR(255), IN newCognomeAdmin VARCHAR(255), IN newMailAdmin
VARCHAR(255), OUT esito BOOL, OUT tipoErrore VARCHAR(255))
BEGIN
DECLARE userRiservato BOOL;
DECLARE mailRiservata BOOL;
SET esito = TRUE;
SET @tipoErrore = "";
IF(newPasswordAdmin != newRipetiPasswordAdmin) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'I campi di verifica password non coincidono','<br />') INTO @tipoErrore;
END IF;
SELECT COUNT(*)>0 INTO userRiservato
FROM CLIENTI
WHERE userCliente=newUserAdmin;
IF(!(userRiservato)) THEN
SELECT COUNT(*)>0 INTO userRiservato
FROM ADMIN
WHERE userAdmin=newUserAdmin;
END IF;
```

```
IF(userRiservato) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'Username gia\' utilizzata','<br />') INTO @tipoErrore;
END IF;
SELECT COUNT(*)>0 INTO mailRiservata
FROM CLIENTI
WHERE mailCliente=newMailAdmin;
IF(!(mailRiservata)) THEN
SELECT COUNT(*)>0 INTO mailRiservata
FROM ADMIN
WHERE mailAdmin=newMailAdmin;
END IF;
IF(mailRiservata) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'Mail gia\' utilizzata','<br />') INTO @tipoErrore;
END IF;
IF(!esito) THEN
INSERT INTO ERRORI(idErrore, descErrore)
VALUES(11, 'Errore nell\'inserimento del nuovo utente');
END IF;
INSERT INTO ADMIN (userAdmin, passwordAdmin, nomeAdmin, cognomeAdmin, mailAdmin)
VALUES (newUserAdmin, newPasswordAdmin, newNomeAdmin, newCognomeAdmin, newMailAdmin);
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS addCliente;
DELIMITER $
CREATE PROCEDURE addCliente
(IN newUserCliente VARCHAR(255), IN newPasswordCliente CHAR(32), IN newRipetiPasswordCliente CHAR(32),
IN newNomeCliente VARCHAR(255), IN newCognomeCliente VARCHAR(255), IN newMailCliente
VARCHAR(255), IN newIndirizzoCliente VARCHAR(255), IN newComuneCliente VARCHAR(255), IN
newProvCliente CHAR(2), IN newCapCliente CHAR(5), IN newCellCliente VARCHAR(255), OUT esito BOOL,
OUT tipoErrore VARCHAR(255))
BEGIN
DECLARE userRiservato BOOL;
DECLARE mailRiservata BOOL;
SET esito = TRUE;
SET @tipoErrore = "";
IF(newPasswordCliente != newRipetiPasswordCliente) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'I campi di verifica password non coincidono','<br />') INTO @tipoErrore;
END IF;
SELECT COUNT(*)>0 INTO userRiservato
FROM CLIENTI
WHERE userCliente=newUserCliente;
IF(!(userRiservato)) THEN
SELECT COUNT(*)>0 INTO userRiservato
FROM ADMIN
WHERE userAdmin=newUserCliente;
END IF;
IF(userRiservato) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'Username gia\' utilizzata','<br />') INTO @tipoErrore;
END IF;
SELECT COUNT(*)>0 INTO mailRiservata
FROM CLIENTI
WHERE mailCliente=newMailCliente;
IF(!(mailRiservata)) THEN
SELECT COUNT(*)>0 INTO mailRiservata
```

```
FROM ADMIN
WHERE mailAdmin=newMailCliente;
END IF;
IF(mailRiservata) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'Mail gia\' utilizzata','<br />') INTO @tipoErrore;
END IF;
IF(!esito) THEN
INSERT INTO ERRORI(idErrore, descErrore)
VALUES(11, 'Errore nell\'inserimento del nuovo cliente');
END IF;
INSERT INTO CLIENTI (userCliente, passwordCliente, nomeCliente, cognomeCliente, mailCliente, indirizzoCliente,
comuneCliente, provCliente, capCliente, cellCliente)
VALUES (newUserCliente, newPasswordCliente, newNomeCliente, newCognomeCliente, newMailCliente,
newIndirizzoCliente, newComuneCliente, newProvCliente, newCapCliente, newCellCliente);
END $


DROP PROCEDURE IF EXISTS addOrdine;
DELIMITER $
CREATE PROCEDURE addOrdine
(IN listaProdotti VARCHAR(255), IN clienteOrdine INT, OUT esito BOOL, OUT codIntOrdineInserito
VARCHAR(255))
BEGIN
DECLARE idOrdine integer;
DECLARE idIntOrdine integer;
DECLARE anno integer;
DECLARE nProdChar varchar(255);
DECLARE nProd integer;
DECLARE idProdChar varchar(255);
DECLARE idProd integer;
DECLARE quantitaChar varchar(255);
DECLARE quantita integer;
DECLARE x integer;
DECLARE conta integer;
SELECT EXTRACT(YEAR FROM CURDATE()) into anno;
SELECT MAX(idInternoOrdine) INTO idIntOrdine FROM ORDINI WHERE annoOrdine = anno;
IF idIntOrdine IS NULL THEN
SET idIntOrdine = 0;
END IF;
SET idIntOrdine = idIntOrdine +1;
SET AUTOCOMMIT=0;
START TRANSACTION;
INSERT INTO ORDINI(idInternoOrdine, annoOrdine, clienteOrdine) VALUES (idIntOrdine, anno, clienteOrdine);
SELECT DISTINCT LAST_INSERT_ID() INTO idOrdine FROM ORDINI;
SELECT SPLIT_STRING(listaProdotti, ',', 1) INTO nProdChar; -- equivalente a: SET nProdChar =
SPLIT_STRING(listaProdotti, ',', 1);
SELECT CAST(nProdChar AS UNSIGNED INTEGER) INTO nProd;
SET x=2;
SET conta=0;
WHILE(conta<nProd) DO
SELECT SPLIT_STRING(listaProdotti, ',', x) INTO idProdChar;
SELECT SPLIT_STRING(listaProdotti, ',', x+1) INTO quantitaChar;
SELECT CAST(idProdChar AS UNSIGNED INTEGER) INTO idProd;
SELECT CAST(quantitaChar AS UNSIGNED INTEGER) INTO quantita;
INSERT INTO ORDINIPRODOTTI(ordineOP, prodottoOP, quantitaOP) VALUES (idOrdine, idProd, quantita);
SET x = x+2;
SET conta=conta+1;
END WHILE;
SELECT CONCAT(idIntOrdine, '-', anno) INTO codIntOrdineInserito;
```

```
COMMIT;
SET AUTOCOMMIT=1;
SET esito = TRUE;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS addPagamento;
DELIMITER $
CREATE PROCEDURE addPagamento
(IN idOrdinePagato INT, IN tipoPag ENUM('Bonifico Bancario', 'Carta di Credito', 'PayPal', 'Postagiro'), OUT esito
BOOL, OUT tipoErrore VARCHAR(255))
BEGIN
DECLARE ultimoID integer;
DECLARE isPO bool;
SET esito = TRUE;
SELECT isPagatoOrdine INTO isPO
FROM ORDINI
WHERE idOrdine=idOrdinePagato;
IF(isPO=TRUE) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'impossibile registrare un nuovo pagamento per un ordine gia\' saldato','<br />') INTO
@tipoErrore;
INSERT INTO ERRORI(idErrore, descErrore)
VALUES(6, 'impossibile registrare un nuovo pagamento per un ordine gia\' saldato');
END IF;
SET AUTOCOMMIT=0;
START TRANSACTION;
INSERT INTO PAGAMENTI(tipoPagamento)
VALUES(tipoPag);
SELECT last_insert_id() INTO ultimoID;
UPDATE ORDINI
SET statoOrdine='Pagato', isPagatoOrdine=TRUE, pagamentoAssOrdine=ultimoID
WHERE idOrdine=idOrdinePagato;
COMMIT;
SET AUTOCOMMIT=1;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS addProdotto;
DELIMITER $
CREATE PROCEDURE addProdotto
(IN nomeProdotto VARCHAR(255), IN prezzoProdotto DECIMAL(7,2), IN pesoProdotto DECIMAL(7,2), IN
porzioniProdotto INT, IN categoriaProdotto INT, IN disponibilitaProdotto INT, IN descrizione VARCHAR(500))
BEGIN
DECLARE codiceDesc INT;
SET AUTOCOMMIT=0;
START TRANSACTION;
INSERT INTO DESCRIZIONIPRODOTTI VALUES (NULL, descrizione);
SELECT DISTINCT LAST_INSERT_ID() INTO codiceDesc FROM DESCRIZIONIPRODOTTI;
INSERT INTO PRODOTTI (nomeProdotto, prezzoProdotto, pesoProdotto, porzioniProdotto, categoriaProdotto,
disponibilitaProdotto, descrizioneAssProdotto) VALUES (nomeProdotto, prezzoProdotto, pesoProdotto,
porzioniProdotto, categoriaProdotto, disponibilitaProdotto, codiceDesc);
COMMIT;
SET AUTOCOMMIT=1;
END $
DELIMITER ;
```

```
DROP PROCEDURE IF EXISTS addSpedizione;
DELIMITER $
CREATE PROCEDURE addSpedizione
(IN idOrdineSpedito INT, IN vettoreUsato VARCHAR(255), IN trackingSped VARCHAR(255), OUT esito BOOL,
OUT tipoErrore VARCHAR(255))
BEGIN
DECLARE ultimoID integer;
DECLARE isPO bool;
SET esito = TRUE;
SELECT isPagatoOrdine INTO isPO
FROM ORDINI
WHERE idOrdine=idOrdineSpedito;
IF(isPO=FALSE) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'impossibile spedire un ordine non ancora saldato','<br />') INTO @tipoErrore;
INSERT INTO ERRORI(idErrore, descErrore)
VALUES(7, 'impossibile spedire un ordine non ancora saldato');
END IF;
SET AUTOCOMMIT=0;
START TRANSACTION;
INSERT INTO SPEDIZIONI(vettoreSpedizione, trackingSpedizione)
VALUES(vettoreUsato, trackingSped);
SELECT last_insert_id() INTO ultimoID;
UPDATE ORDINI
SET statoOrdine='Chiuso', isChiusoOrdine=TRUE, spedizioneAssOrdine=ultimoID
WHERE idOrdine=idOrdineSpedito;
COMMIT;
SET AUTOCOMMIT=1;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS delAdmin;
DELIMITER $
CREATE PROCEDURE delAdmin
(IN idAdminDaCanc integer, OUT esito BOOL, OUT tipoErrore VARCHAR(255))
BEGIN
DECLARE quantiOrdini integer;
SET esito = TRUE;
SET @tipoErrore = "";
SELECT COUNT(*) INTO quantiOrdini
FROM ORDINI
WHERE adminAssOrdine = idAdminDaCanc;
IF (quantiOrdini>0) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'impossibile cancellare l\'admin perche\' ha in gestione almeno un ordine','<br />')
INTO @tipoErrore;
INSERT INTO ERRORI (idErrore, descErrore)
VALUES (12, 'impossibile cancellare l\'admin perche\' ha in gestione almeno un ordine');
END IF;
DELETE FROM ADMIN
WHERE idAdmin=idAdminDaCanc;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS delCliente;
DELIMITER $
CREATE PROCEDURE delCliente
(IN idClienteDaCanc integer, OUT esito BOOL, OUT tipoErrore VARCHAR(255))
```

```
BEGIN
DECLARE quantiOrdini integer;
SET esito = TRUE;
SET @tipoErrore = "";
SELECT COUNT(*) INTO quantiOrdini
FROM ORDINI
WHERE clienteOrdine = idClienteDaCanc;
IF (quantiOrdini>0) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'impossibile cancellare il cliente perche\' ha almeno un ordine associato','<br />')
INTO @tipoErrore;
INSERT INTO ERRORI (idErrore, descErrore)
VALUES (3, 'impossibile cancellare il cliente perche\' ha almeno un ordine associato');
END IF;
DELETE FROM CLIENTI
WHERE idCliente=idClienteDaCanc;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS delOrdine;
DELIMITER $
CREATE PROCEDURE delOrdine
(IN idOrdineDaCanc integer)
BEGIN
DECLARE statoOrdineDaCanc ENUM('Registrato', 'Pagato', 'In lavorazione', 'Chiuso');
DECLARE prodOP integer;
DECLARE quantOP integer;
DECLARE spedizioneAss integer;
DECLARE pagamentoAss integer;
DECLARE Done integer default 0;
DECLARE cursoreOP CURSOR FOR SELECT prodottoOP, quantitaOP
FROM ORDINIPRODOTTI
WHERE ordineOP=idOrdineDaCanc;
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET Done = 1;
SELECT statoOrdine INTO statoOrdineDaCanc
FROM ORDINI
WHERE idOrdine=idOrdineDaCanc;
SET AUTOCOMMIT=0;
START TRANSACTION;
IF(statoOrdineDaCanc='Registrato') THEN
OPEN cursoreOP;
REPEAT
        FETCH cursoreOP INTO prodOP, quantOP;
        IF NOT Done THEN
                UPDATE PRODOTTI
                SET disponibilitaProdotto=disponibilitaProdotto+quantOP
                WHERE idProdotto=prodOP;
        END IF;
UNTIL Done END REPEAT;
CLOSE cursoreOP;
END IF;
SELECT spedizioneAssOrdine INTO spedizioneAss
FROM ORDINI
WHERE idOrdine = idOrdineDaCanc;
DELETE FROM SPEDIZIONI
WHERE idSpedizione=spedizioneAss;
SELECT pagamentoAssOrdine INTO pagamentoAss
FROM ORDINI
```

```
WHERE idOrdine = idOrdineDaCanc;
DELETE FROM PAGAMENTI
WHERE idPagamento=spedizioneAss;
DELETE FROM ORDINIPRODOTTI
WHERE ordineOP=idOrdineDaCanc;
DELETE FROM ORDINI
WHERE idOrdine=idOrdineDaCanc;
COMMIT;
SET AUTOCOMMIT=1;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS delProdotto;
DELIMITER $
CREATE PROCEDURE delProdotto
(IN idProdottoDaCanc VARCHAR(255), OUT esito BOOL)
BEGIN
DECLARE quantiOrdini integer;
DECLARE idDesc integer;
SET AUTOCOMMIT=0;
START TRANSACTION;
SELECT COUNT(*) INTO quantiOrdini
FROM ORDINIPRODOTTI
WHERE prodottoOP = idProdottoDaCanc;
IF (quantiOrdini>0) THEN
INSERT INTO ERRORI (idErrore, descErrore)
VALUES (3, 'impossibile cancellare il prodotto perche\' ha almeno un ordine associato');
END IF;
SET FOREIGN_KEY_CHECKS=0;
DELETE FROM PRODOTTI WHERE idProdotto = idProdottoDaCanc;
SELECT descrizioneAssProdotto INTO idDesc FROM PRODOTTI WHERE idProdotto = idProdottoDaCanc;
DELETE FROM DESCRIZIONIPRODOTTI WHERE idDesc = idDesc;
DELETE FROM MEDIA WHERE prodottoMedia = idProdottoDaCanc;
SET FOREIGN_KEY_CHECKS=1;
COMMIT;
SET AUTOCOMMIT=1;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS tryLogin;
DELIMITER $
CREATE PROCEDURE tryLogin
(IN userUtente VARCHAR(255), IN passwordUtente CHAR(32), OUT esito BOOL, OUT gruppoUtente BOOL, OUT
nomeUtente VARCHAR(255), OUT cognomeUtente VARCHAR(255), OUT idUtente INT)
BEGIN
DECLARE utenteTrovato BOOL;
DECLARE adminTrovato BOOL;
SET utenteTrovato = FALSE;
SET adminTrovato = FALSE;
SELECT COUNT(*)>0 INTO utenteTrovato
FROM CLIENTI
WHERE userCliente=userUtente AND passwordCliente=passwordUtente;
IF(utenteTrovato) THEN
SET esito = TRUE;
SET gruppoUtente = FALSE;
SELECT nomeCliente, cognomeCliente, idCliente INTO nomeUtente, cognomeUtente, idUtente
FROM CLIENTI
WHERE userCliente=userUtente AND passwordCliente=passwordUtente;
```

```
ELSE
SELECT COUNT(*)>0 INTO adminTrovato
FROM ADMIN
WHERE userAdmin=userUtente AND passwordAdmin=passwordUtente;
IF(adminTrovato) THEN
SET esito = TRUE;
SET gruppoUtente = TRUE;
SELECT nomeAdmin, cognomeAdmin, idAdmin INTO nomeUtente, cognomeUtente, idUtente
FROM ADMIN
WHERE userAdmin=userUtente AND passwordAdmin=passwordUtente;
ELSE
SET esito = FALSE;
END IF;
END IF;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS updAdmin;
DELIMITER $
CREATE PROCEDURE updAdmin
(IN updIdAdmin INT, newUserAdmin VARCHAR(255), IN newNomeAdmin VARCHAR(255), IN
newCognomeAdmin VARCHAR(255), IN newMailAdmin VARCHAR(255), OUT esito BOOL, OUT tipoErrore
VARCHAR(255))
BEGIN
DECLARE userRiservato BOOL;
DEClARE mailRiservata BOOL;
SET esito = TRUE;
SET @tipoErrore = "";
SELECT COUNT(*)>0 INTO userRiservato
FROM CLIENTI
WHERE userCliente=newUserAdmin;
IF(!(userRiservato)) THEN
SELECT COUNT(*)>0 INTO userRiservato
FROM ADMIN
WHERE userAdmin=newUserAdmin
AND idAdmin <> updIdAdmin;
END IF;
IF(userRiservato) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'Username gia\' utilizzata','<br />') INTO @tipoErrore;
END IF;
SELECT COUNT(*)>0 INTO mailRiservata
FROM CLIENTI
WHERE mailCliente=newMailAdmin;
IF(!(mailRiservata)) THEN
SELECT COUNT(*)>0 INTO mailRiservata
FROM ADMIN
WHERE mailAdmin=newMailAdmin
AND idAdmin <> updIdAdmin;
END IF;
IF(mailRiservata) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'Mail gia\' utilizzata','<br />') INTO @tipoErrore;
END IF;
IF(!esito) THEN
INSERT INTO ERRORI(idErrore, descErrore)
VALUES(11, 'Errore nell\'inserimento del nuovo utente');
END IF;
UPDATE ADMIN SET userAdmin = newUserAdmin, nomeAdmin = newNomeAdmin, cognomeAdmin =
```

```
newCognomeAdmin, mailAdmin = newMailAdmin
WHERE idAdmin = updIdAdmin;
 END $
DELIMITER ;


DROP PROCEDURE IF EXISTS updCliente;
DELIMITER $
CREATE PROCEDURE updCliente
(IN updIdCliente INT,IN newUserCliente VARCHAR(255), IN newNomeCliente VARCHAR(255), IN
newCognomeCliente VARCHAR(255), IN newMailCliente VARCHAR(255), IN newIndirizzoCliente
VARCHAR(255), IN newComuneCliente VARCHAR(255), IN newProvCliente CHAR(2), IN newCapCliente
CHAR(5), IN newCellCliente VARCHAR(255), OUT esito BOOL, OUT tipoErrore VARCHAR(255))
BEGIN
DECLARE userRiservato BOOL;
DEClARE mailRiservata BOOL;
SET esito = TRUE;
SET @tipoErrore = "";
SELECT COUNT(*)>0 INTO userRiservato
FROM CLIENTI
WHERE userCliente=newUserCliente
AND idCliente <> updIdCliente;
IF(!(userRiservato)) THEN
SELECT COUNT(*)>0 INTO userRiservato
FROM ADMIN
WHERE userAdmin=newUserCliente;
END IF;
IF(userRiservato) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'Username gia\' utilizzata','<br />') INTO @tipoErrore;
END IF;
SELECT COUNT(*)>0 INTO mailRiservata
FROM CLIENTI
WHERE mailCliente=newMailCliente
AND idCliente <> updIdCliente;
IF(!(mailRiservata)) THEN
SELECT COUNT(*)>0 INTO mailRiservata
FROM ADMIN
WHERE mailAdmin=newMailCliente;
END IF;
IF(mailRiservata) THEN
SET esito = FALSE;
SELECT CONCAT(@tipoErrore,'Mail gia\' utilizzata','<br />') INTO @tipoErrore;
END IF;
IF(!esito) THEN
INSERT INTO ERRORI(idErrore, descErrore)
VALUES(11, 'Errore nell\'inserimento del nuovo cliente');
END IF;
UPDATE CLIENTI SET userCliente = newUserCliente, nomeCliente = newNomeCliente, cognomeCliente =
newCognomeCliente, mailCliente = newMailCliente, indirizzoCliente = newIndirizzoCliente, comuneCliente =
newComuneCliente, provCliente = newProvCliente, capCliente = newCapCliente, cellCliente = newCellCliente
WHERE idCliente = updIdCliente;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS updDispProdotto;
 DELIMITER $
CREATE PROCEDURE updDispProdotto
(IN idProdottoUpd int, IN nuovaDisp int)
```

```
BEGIN
IF(nuovaDisp<0) THEN
INSERT INTO ERRORI(idErrore, descErrore)
VALUES (5, 'impossibile registrare una disponibilita\' prodotto negativa');
END IF;
UPDATE PRODOTTI
SET disponibilitaProdotto=nuovaDisp
WHERE idProdotto=idProdottoUpd;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS updQuantitaBody;
DELIMITER $
CREATE PROCEDURE updQuantitaBody(prodOP integer, quantOP integer)
BEGIN
DECLARE vecchiaQuantita integer;
DECLARE nuovaQuantita integer;
SELECT disponibilitaProdotto INTO vecchiaQuantita
FROM PRODOTTI
WHERE idProdotto = prodOP;
SET nuovaQuantita = vecchiaQuantita-quantOP;
IF (nuovaQuantita<0) THEN
INSERT INTO ERRORI(idErrore, descErrore)
VALUES(4, 'impossibile registrare l\'ordine: la quantita\' richiesta eccede quella disponibile');
END IF;
UPDATE PRODOTTI
SET disponibilitaProdotto=nuovaQuantita
WHERE idProdotto=prodOP;
END $
DELIMITER ;


DROP PROCEDURE IF EXISTS superdelCliente;
DELIMITER $
CREATE PROCEDURE superdelCliente
(IN idClienteDaCanc integer)
BEGIN
DECLARE quantiOrdini integer;
DECLARE numOrdine integer;
DECLARE Done integer default 0;
DECLARE cursoreOrdini CURSOR FOR SELECT idOrdine
FROM ORDINI
WHERE clienteOrdine=idClienteDaCanc;
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET Done = 1;
SELECT COUNT(*) INTO quantiOrdini
FROM ORDINI
WHERE clienteOrdine=idClienteDaCanc;
SET AUTOCOMMIT=0;
START TRANSACTION;
IF (quantiOrdini>0) THEN
OPEN cursoreOrdini;
REPEAT
        FETCH cursoreOrdini INTO numOrdine;
        IF NOT Done THEN
                CALL delOrdine(numOrdine);
        END IF;
UNTIL Done END REPEAT;
CLOSE cursoreOrdini;
```

```
END IF;
DELETE FROM CLIENTI
WHERE idCliente=idClienteDaCanc;
COMMIT;
SET AUTOCOMMIT=1;
END $
DELIMITER ;


SET GLOBAL log_bin_trust_function_creators=1;
CREATE FUNCTION SPLIT_STRING(str VARCHAR(255), delimitatore VARCHAR(12), posizione INT)
RETURNS VARCHAR(255)
RETURN REPLACE(SUBSTRING(SUBSTRING_INDEX(str, delimitatore, posizione),
LENGTH(SUBSTRING_INDEX(str, delimitatore, posizione-1)) + 1), delimitatore, '');
```

---

8. Queries

Di seguito verrà riportato il codice di alcune query di interesse per attività amministrative con relativo output:

1. Per ogni prodotto l'Id, il nome e il numero di acquisti nell'anno 2014 in ordine di vendite:

```
SELECT prodottoOP, SUM(quantitaOP) AS numVendite
FROM ORDINIPRODOTTI INNER JOIN ORDINI ON idOrdine = ordineOP
WHERE annoOrdine = 2014
GROUP BY prodottoOP
ORDER BY numVendite DESC
```

prodottoOP	nomeProdotto	numVendite
3	Brioche alla crema	100
7	Crostata ai frutti di bosco	48
2	Treccia di ricotta	37
4	Torta paradiso	29
6	Sfogliata alle erbette	21
9	Cous cous con piovra	4
1	Crostata di crema cotta	4
5	Sfogliata di mele e radicchio	2
8	Trancio di pesce spada in crosta	2
10	Zuppa di pesce	1

2. Id del cliente che ha effettuato il maggior numero di ordini relativi al prodotto 4:

```
SELECT clienteOrdine FROM (
SELECT clienteOrdine, COUNT( * ) AS numOrdine
FROM ORDINI
INNER JOIN ORDINIPRODOTTI ON idOrdine = ordineOP
WHERE prodottoOP =4
GROUP BY clienteOrdine) t
HAVING max(numOrdine)
```

clienteOrdine
4

3. Id e nome del prodotto che a giugno 2014 è stato venduto più di 5 volte:

```
SELECT  idProdotto, nomeProdotto
FROM PRODOTTI NATURAL JOIN ORDINIPRODOTTI
WHERE idProdotto IN (
SELECT prodottoOP FROM ORDINIPRODOTTI INNER JOIN ORDINI ON idOrdine = ordineOP
WHERE statoOrdine = 'chiuso'
AND dataOrdine BETWEEN '2014-06-01' AND '2014-06-30' )
GROUP BY idProdotto
HAVING SUM(quantitaOP)>5;
```

idProdotto	nomeProdotto
3	Brioche alla crema

4. Nome, cognome e il totale speso per i clienti che hanno effettuato acquisti nel mese di giugno 2014. Per i clienti che non hanno effettuato acquisti ritorna 0 :

```
SELECT c.nomeCliente, c.cognomeCliente, t.totSpesa
FROM CLIENTI c INNER JOIN (
SELECT clienteOrdine, sum(importoOrdine) AS totSpesa
FROM ORDINI
WHERE dataOrdine BETWEEN '2014-06-01' AND '2014-06-30'
GROUP BY clienteOrdine) t
ON t.clienteOrdine = c.idCliente
UNION
SELECT c.nomeCliente, c.cognomeCliente, 0 as totSpesa
FROM CLIENTI c
WHERE c.idCliente NOT IN (
SELECT clienteOrdine
FROM ORDINI
WHERE dataOrdine between '2014-06-01' and '2014-06-30')
```

nomeCliente	cognomeCliente	totSpesa
Mario	Tubi	80
Guido	Ferrari	45
Fabio	Marra	320
Tony	Ryan	1935
Jeffrey	Bezos	1210
Carlo	Bianchi	0

5. Id, nome e categoria dei prodotti dei quali ne sono state vendute più di 10 copie quest'anno:

```
SELECT p.idProdotto, p.nomeProdotto, p.categoriaProdotto
FROM PRODOTTI p NATURAL JOIN ORDINIPRODOTTI op
WHERE p.idProdotto IN(
SELECT pr.idProdotto
FROM ORDINI o JOIN ORDINIPRODOTTI opr ON (o.idOrdine=opr.ordineOP) JOIN PRODOTTI pr ON (opr.prodottoOP=pr.idProdotto)
WHERE o.statoOrdine = 'chiuso' AND EXTRACT(YEAR FROM o.dataOrdine)=EXTRACT(YEAR FROM CURDATE()))
GROUP BY p.idProdotto
HAVING SUM(op.quantitaOP)>10;
```

idProdotto	nomeProdotto	categoriaProdotto
2	Treccia di ricotta	4
3	Brioche alla crema	1
4	Torta paradiso	4
6	Sfogliata alle erbette	8
7	Crostata ai frutti di bosco	5

6. Id e lo stato degli ordini che non contengono Crostate di crema cotta e i prodotti ordinati sono in numero minore ugale a due:

```
SELECT o.idOrdine, o.statoOrdine
FROM ORDINI o
WHERE NOT EXISTS(
SELECT *
FROM ORDINIPRODOTTI op JOIN PRODOTTI p ON(op.prodottoOP=p.idProdotto)
WHERE o.idOrdine=op.ordineOP AND(p.nomeProdotto = 'Crostate di creama cotta' OR op.quantitaOP>2));
```

idOrdine	statoOrdine
5	Registrato
6	Pagato
7	Registrato
8	Registrato

9. Interfaccia web

L'interfaccia web è composta dalle pagine: addAdmin, addCliente, getCarrello, getOrdini, getUtenti, index, scrClienti, updAdmin, updCliente e pagine contenenti le funzioni richiamate da queste: calcolaPaginazione, connessione, setup e validazione. L'interfaccia web inoltre fa utilizzo di un foglio di stile css, il file main.css. Per non produrre una realzione di lunghezza eccessiva di segiuto verrà riportato il codice del solo file index.php.

```
<?php
require_once "pack/setup.php" ;
require_once "pack/connessione.php" ;
require_once "pack/validazione.php" ;
require_once "pack/calcolaPaginazione.php" ;
$erroreLogin = false; // Variabili per segnalare errori nel form di login
$descErrore = "";
$isAvvisi = FALSE; // Variabili per segnalare tutti i possibili messaggi
$avvisi = "";

if(isset($_GET['logout'])){
	$isAvvisi = TRUE;
	$avvisi = "Grazie della visita. Arrivederci.";
}
// Se la pagina e' stata richiesta dal form dati, processo il login
if(isset($_POST['Login']))
{
	if($_POST['user']=="" || $_POST['pwd']=="")
	{
		$erroreLogin = true;
		$descErrore = "ERRORE: Campi Mancanti!";
	}else{
		$esitoLogin = false;
		$gruppoUtente = 0;
		$nomeUtente = "";
		$cognomeUtente = "";
		$idUtente = 0;

		$criptPsw = md5(sha1($_POST['pwd']));

		try{
			if (!($stmt = $mysqli->prepare("CALL tryLogin(?,?, @esito, @gruppoUtente,
@nomeUtente, @cognomeUtente, @idUtente);")))
				throw new Exception ('CALL fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

			if (!($stmt->bind_param("ss", $_POST['user'], $criptPsw)))
				throw new Exception ('bind_param fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

			if (!($stmt->execute()))
				throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
			if (!($stmt = $mysqli->prepare("SELECT @esito, @gruppoUtente, @nomeUtente,
@cognomeUtente, @idUtente;")))
				throw new Exception ('prepare fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

			if (!($stmt->execute()))
				throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

			if (!($stmt->bind_result($esitoU, $gruppoU, $nomeU, $cognomeU, $idU)))
				throw new Exception ('bind_result fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
```

```
                        if (!($stmt->fetch()))
                                throw new Exception ('fetch fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                        if (!($stmt->close()))
                                throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' .
$mysqli->error);

                }catch (Exception $excp) {
                echo $excp -> getMessage();
                }
                if($esitoU){
                        session_start();
                        $_SESSION['loggedIn'] = 'youAreLogged';
                        $_SESSION['id_U'] = $idU;
                        $_SESSION['nome_U'] = $nomeU;
                        $_SESSION['cognome_U'] = $cognomeU;
                        $_SESSION['gruppo_U'] =  $gruppoU;
                }else{
                        $erroreLogin = true;
                        $descErrore = "ERRORE: Utente sconosciuto!";
                }
        }
}else{  // a pagina e' stata richiesta da un Utente. Ho bisogno dei suoi dati di sessione eventualmente fosse gia' loggato
        session_start();
}
// Le query generiche per recuperare i prodotti quando non e' stata fatta una ricerca
$query = "SELECT idProdotto, nomeProdotto, prezzoProdotto, categoriaProdotto, disponibilitaProdotto,
descrizioneAssProdotto, porzioniProdotto FROM PRODOTTI LIMIT ?, ?;"; // le query base, se la pagina non e'
chiamata da una operazione di ricerca
$queryCount = "SELECT COUNT(*) FROM PRODOTTI;";
// Leggo le categorie per la composizione dell'input select
try{
        if (!($stmt = $mysqli->prepare("SELECT idCategoria, nomeCategoria FROM CATEGORIE;")))
                throw new Exception ('SELECT fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
        if (!($stmt->execute()))
                throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
        if (!($stmt->bind_result($idCategoria, $nomeCategoria)))
                throw new Exception ('bind_result fallita: (' . $mysqli->errno . ') ' . $mysqli->error)
        $selectCategorie = "<select name='cate'>";
        $selectCategorie .= "<option value='0'>Tutte le categorie</option>";
        while($stmt->fetch()){
                $selectCategorie .= "<option value='$idCategoria'>$nomeCategoria</option>";
        }
        $selectCategorie .= "<select name='cate'>";
        if (!($stmt->close()))
                throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' . $mysqli->error);
}catch (Exception $excp) {
   echo $excp -> getMessage();
}
// Processo una eventuale richiesta di update della quantita' di un prodotto
if(isset($_POST['invioUPDquantita'])){

        if(!validateID("prodottoqnt", "ID prodotto da aggiornare", false, $avvisi))
                $isAvvisi = true;

        if(!validateID("qnt", "nuova quantita' prodotto", false, $avvisi))
                $isAvvisi = true;

        if(!$isAvvisi){
```

```
                try{
                        if (!($stmt = $mysqli->prepare("CALL updDispProdotto(?,?);")))
                                throw new Exception ('CALL fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                        if (!($stmt->bind_param("ii", $_POST['prodottoqnt'], $_POST['qnt'])))
                                throw new Exception ('bind_param fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                        if (!($stmt->execute()))
                                throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                        if (!($stmt->close()))
                                throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' .
$mysqli->error);

                }catch (Exception $excp) {
                echo $excp -> getMessage();
                }

        }
        if(!($isAvvisi)){
                $avvisi = "Modifica del Prodotto riuscita!";
        }
        $isAvvisi = TRUE;  // settato in ogni caso per mostrare il messaggio di conferma operazione o gli eventuali
errori
}
// Processo una eventuale richiesta di update del prezzo di un prodotto
if(isset($_POST['invioUPDprezzo'])){

        if(!validateID("prodottoprz", "ID prodotto da aggiornare", false, $avvisi))
                $isAvvisi = true;
        if(!validateImporto("prz", "nuova quantita' prodotto", false, false, $avvisi))
                $isAvvisi = true;
        if(!$isAvvisi){
                try{
                        if (!($stmt = $mysqli->prepare("UPDATE PRODOTTI SET prezzoProdotto=? WHERE
idProdotto=?;")))
                                throw new Exception ('CALL fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                        if (!($stmt->bind_param("di", $_POST['prz'], $_POST['prodottoprz'])))
                                throw new Exception ('bind_param fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                        if (!($stmt->execute()))
                                throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                        if (!($stmt->close()))
                                throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' .
$mysqli->error);

                }catch (Exception $excp) {
                echo $excp -> getMessage();
                }
        }
        if(!($isAvvisi)){
                $avvisi = "Modifica del Prodotto riuscita!";
        }
        $isAvvisi = TRUE;  // settato in ogni caso per mostrare il messaggio di conferma operazione o gli eventuali
errori
}
// Processo una eventuale ricerca sui prodotti
if(isset($_POST['ricercaProd'])){
```

```
		if(!(validateID("cate", "categoria prodotto", false, $avvisi)))
			$isAvvisi = true;

		if(!(validateText("valRic","testo da ricercare", 3, true, $avvisi)))
			$isAvvisi = true;

		if(!$isAvvisi){
			if($_POST['cate']==0)
				$categ = "";
			else
				$categ = " AND categoriaProdotto = ".$_POST['cate'];
			if($_POST['valRic']=="")
				$where = " WHERE 0=0";
			else
				$where = " WHERE nomeProdotto LIKE '%".$_POST['valRic']."%'";
			// Nel caso sia stata fatta una ricerca conpongo le query personalizzate
			$query = "SELECT idProdotto, nomeProdotto, prezzoProdotto, categoriaProdotto,
disponibilitaProdotto, descrizioneAssProdotto, porzioniProdotto FROM PRODOTTI$where$categ LIMIT ?, ?;"; // le
query base, se la pagina non e' chiamata da una operazione di ricerca
			$queryCount = "SELECT COUNT(*) FROM PRODOTTI$where$categ;";

		}
}
 try{
		//Determino il numero totale di prodotti salvati nel programma
		if (!($stmt = $mysqli->prepare($queryCount)))
			throw new Exception ('SELECT COUNT(*) fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
		if (!($stmt->execute()))
			throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
		if (!($stmt->bind_result($nProdotti)))
			throw new Exception ('bind_result fallita: (' . $mysqli->errno . ') ' . $mysqli->erro
		if (!($stmt->fetch()))
			throw new Exception ('fetch fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
		if (!($stmt->close()))
			throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' . $mysqli->error);
}catch (Exception $excp) {
   echo $excp -> getMessage();
}
// Se non e' loggato un admin devo visualizzare anche le schede dei prodotti: visualizzo un numero di prodotti/pagina
ridotto
if(((isset($_SESSION['loggedIn']))&&($_SESSION['loggedIn'] ==
'youAreLogged')&&($_SESSION['gruppo_U']==0))||(!(isset($_SESSION['loggedIn'])))){
		$nMaxRighe = $nMaxRigheRidotto;
}
$page = calcolaPaginazione($nProdotti, $nMaxRighe, $nPages, $inizio, $quanti, $isAvvisi, $avvisi);

// Creo l'input select per la scelta della pagina
$selectBody = "";
for($i=1; $i<=$nPages+1;$i++){
		if($i == $page)
			$selectBody .= "<option value='$i' selected>$i</option>";
		else
			$selectBody .= "<option value='$i'>$i</option>";
}
try{
		// Estraggo le informazioni di ogni singolo prodotto che compone la pagina richiesta
		if (!($stmt = $mysqli->prepare($query)))
			throw new Exception ('SELECT fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

		if (!($stmt->bind_param("ii", $inizio, $quanti)))
```

```
                throw new Exception ('bind_param fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

        if (!($stmt->execute()))
                throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

        if (!($stmt->store_result())) // Serve per poter eseguire una query annidata nel fetch di questa senza dover aprire una seconda connessione sul DB
                throw new Exception ('store_result fallita: (' . $mysqli->errno . ') ' . $mysqli->error)
        if (!($stmt->bind_result($idProdotto, $nomeProdotto, $prezzoProdotto, $categoriaProdotto, $disponibilitaProdotto, $descrizioneAssProdotto, $porzioniProdotto)))
                throw new Exception ('bind_result fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
        $tabellaProdotti = "";
        if($nProdotti==0){ // Se la ricerca non ha prodotto risultati lo comunico
                $tabellaProdotti .= "Nessun Prodotto Trovato.";
        }else{ //la ricerca ha prodotto risultati
                $tabellaProdotti .= "<table class='righeSelect' id='getItems'>";
                $i = 0;
                while($stmt->fetch()){
                        // Sottraggo alla disponibilita' corrente eventuali prodotti gia' nel carrello
                        if(isset($_COOKIE['carrello'])){

                                $datiCarrello = explode(",",$_COOKIE['carrello']);
                                $nProdottiCarrello = $datiCarrello[0];

                                for($k=1;$k<=($nProdottiCarrello*2);$k=$k+2){
                                        if($datiCarrello[$k]==".$idProdotto){
                                                $disponibilitaProdotto = $disponibilitaProdotto-$datiCarrello[$k+1];
                                        }
                                }
                        }
                        if($disponibilitaProdotto > 4)
                                $disponibilita = "<img src='media/green.png' alt='in stock' />";
                        else if($disponibilitaProdotto > 0)
                                $disponibilita = "<img src='media/orange.png' alt='low stock' />";
                        else
                                $disponibilita = "<img src='media/red.png' alt='out of stock' />";

                        $disponibilita .= " Disponibilita': $disponibilitaProdotto pezzi";

                        // formatto l'importo con 2 decimali dopo la virgola
                        formattaImporto($prezzoProdotto);

                        $operazioni = "";
                        $controlliCarrello = "";
                        $schedaProdotto = "";
                        if((isset($_SESSION['loggedIn']))&&($_SESSION['loggedIn'] == 'youAreLogged')&&($_SESSION['gruppo_U']==1)){ // E' loggato un admin: invece dei pulsanti di gestione del carrello visualizzo quelli di amministrazione
                                $isAggiornaQnt = FALSE;
                                $isAggiornaPrz = FALSE;
                                $aggiornaQnt = "";
                if((isset($_GET['updQuantita']))&&($_GET['updQuantita']==$idProdotto)){
                                        $isAggiornaQnt = TRUE;
                                        $prod = $_GET['updQuantita'];
                                        $aggiornaQnt = "Inserisci Nuova Quantita' <form method='POST' action='index.php'><input type='text' name='qnt'><input type='hidden' name='prodottoqnt' value='$prod'><input type='submit' name='invioUPDquantita' value='Salva'> </form>";
                                }else
                                        $aggiornaQnt = "[<a href='index.php?updQuantita=$idProdotto'
```

```
class='operazione'><img src='media/edit.png' width='16px' height='16px'/>Aggiorna quantita'</a>]";

                                        $aggiornaPrz = "";
                                        if((isset($_GET['updPrezzo']))&&($_GET['updPrezzo']==$idProdotto)){
                                                $isAggiornaPrz = TRUE;
                                                $prod = $_GET['updPrezzo'];
                                                $aggiornaPrz = "Inserisci Nuovo Prezzo <form method='POST'
action='index.php'><input type='text' name='prz'><input type='hidden' name='prodottoprz' value='$prod'><input
type='submit' name='invioUPDprezzo' value='Salva'> </form>";
                                        }else
                                                $aggiornaPrz = "[<a href='index.php?updPrezzo=$idProdotto'
class='operazione'><img src='media/edit.png' width='16px' height='16px'/>Aggiorna Prezzo</a>]";

                                        if($isAggiornaQnt)
                                                $aggiornaPrz = "";

                                        if($isAggiornaPrz)
                                                $aggiornaQnt = "";
                                        $operazioni .= "$aggiornaQnt $aggiornaPrz";
                                }else{// E' loggato un Cliente oppure la pagina e' visitata da un ospite: visualizzo i pulsanti di
gestione del carrello
                                        if($disponibilitaProdotto>0){
                                        // creo la select per la selezione della quantita'
                                                $j=1;
                                                $quantita = "(Quantita'<select name='quantitaP' style='width:50px;'>";
                                                while($j<=$disponibilitaProdotto){
                                                        $quantita .= "<option value='$j'>$j</option>";
                                                        $j++;
                                                }
                                                $quantita .= "</select>)";
                                                $pulsante = "<input type='submit' name='aggProd' value='Aggiungi al
Carrello'>";
                                                $controlliCarrello = "<form method='POST' action='index.php'>$quantita
<input type='hidden' name='idProd' value='$idProdotto'> $pulsante</form>";
                                        }else{
                                                $controlliCarrello = "Prodotto non disponibile";
                                        }
                                        // Per ogni prodotto, leggo una foto ad esso associata per la creazione della sua
scheda
                                        if (!($stmt2 = $mysqli->prepare("SELECT linkMedia FROM MEDIA WHERE
prodottoMedia=? AND tipoMedia=1;")))
                                                throw new Exception ('SELECT fallita: (' . $mysqli->errno . ') ' . $mysqli-
>error);

                                        if (!($stmt2->bind_param("i", $idProdotto)))
                                                throw new Exception ('bind_param fallita: (' . $mysqli->errno . ') ' .
$mysqli->error);

                                        if (!($stmt2->execute()))
                                                throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli-
>error);

                                        if (!($stmt2->store_result())) // Serve per poter eseguire una query annidata nel fetch
di questa senza dover aprire una seconda connessione sul DB
                                                throw new Exception ('store_result fallita: (' . $mysqli->errno . ') ' .
$mysqli->error);

                                        if (!($stmt2->bind_result($linkProdotto)))
                                                throw new Exception ('bind_result fallita: (' . $mysqli->errno . ') ' . $mysqli-
>error);
```

```
                                        if (!($stmt2->fetch()))
                                                throw new Exception ('fetch fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                                        if (!($stmt2->close()))
                                                throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' . $mysqli->error);
                                        // Per ogni prodotto, leggo la categoria esso associata per la creazione della sua scheda
                                        if (!($stmt2 = $mysqli->prepare("SELECT nomeCategoria FROM CATEGORIE WHERE idCategoria=?")))
                                                throw new Exception ('SELECT fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                                        if (!($stmt2->bind_param("i", $categoriaProdotto)))
                                                throw new Exception ('bind_param fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                                        if (!($stmt2->execute()))
                                                throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                                        if (!($stmt2->store_result())) // Serve per poter eseguire una query annidata nel fetch di questa senza dover aprire una seconda connessione sul DB
                                                throw new Exception ('store_result fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                                        if (!($stmt2->bind_result($nomeCatProdotto)))
                                                throw new Exception ('bind_result fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                                        if (!($stmt2->fetch()))
                                                throw new Exception ('fetch fallita: (' . $mysqli->errno . ') ' . $mysqli->error);

                                        if (!($stmt2->close()))
                                                throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' . $mysqli->error);

                                        // Per ogni prodotto, leggo la descrizione esso associata per la creazione della sua scheda
                                        if (!($stmt2 = $mysqli->prepare("SELECT descrizione FROM DESCRIZIONIPRODOTTI WHERE idDescrizione=?;")))
                                                throw new Exception ('SELECT fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
                                        if (!($stmt2->bind_param("i", $descrizioneAssProdotto)))
                                                throw new Exception ('bind_param fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
                                        if (!($stmt2->execute()))
                                                throw new Exception ('execute fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
                                        if (!($stmt2->store_result())) // Serve per poter eseguire una query annidata nel fetch di questa senza dover aprire una seconda connessione sul DB
                                                throw new Exception ('store_result fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
                                        if (!($stmt2->bind_result($descrizioneProdotto)))
                                                throw new Exception ('bind_result fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
                                        if (!($stmt2->fetch()))
```

```
                                        throw new Exception ('fetch fallita: (' . $mysqli->errno . ') ' . $mysqli->error);
                                if (!($stmt2->close()))
                                        throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' . $mysqli->error);
                                // Rimuovo i ritorni a capo nella descrizione, non richiesti
                                $descrizioneProdotto = str_replace("\\n", "", $descrizioneProdotto);
                                $descrizioneProdotto = str_replace("-", "", $descrizioneProdotto);
                                if($i%2==1){
                                        $schedaProdotto = "<tr><td colspan='7' class='unpair'><div class='fotoProdotto'><img src='$linkProdotto' alt='foto prodotto $idProdotto' width='80px' height='50px' /></div><div class='datiProdotto'>$nomeCatProdotto<br/>($porzioniProdotto porzioni)</div><div class='descrizioneProdotto'>$descrizioneProdotto</div></td></tr>";
                                }
                                else{
                                        $schedaProdotto = "<tr><td colspan='7' class='pair'><div class='fotoProdotto'><img src='$linkProdotto' alt='foto prodotto $idProdotto' width='80px' height='50px' /></div><div class='datiProdotto'>$nomeCatProdotto<br/>($porzioniProdotto porzioni)</div><div class='descrizioneProdotto'>$descrizioneProdotto</div></td></tr>";
                                }
                        }
                        if($i%2==1){
                                $tabellaProdotti .= "<tr class='unpair' onMouseOver='setColor(this, 0, \"#cc6600\")' onMouseOut='setColor(this, 1, \"#282828\")'><td align='center' style='width:24px;'><input type='checkbox' name='$idProdotto' id='$idProdotto' onclick='select_row(this);' /></td><td style='width:268px;'>$nomeProdotto</td><td>$disponibilita</td><td align='center'>$prezzoProdotto</td><td align='center'> $operazioni$controlliCarrello</td></tr>
                                $schedaProdotto";
                        }else{
                                $tabellaProdotti .= "<tr class='pair' onMouseOver='setColor(this, 0, \"#cc6600\")' onMouseOut='setColor(this, 1, \"#282828\")'><td align='center' style='width:24px;'><input type='checkbox' name='$idProdotto' id='$idProdotto' onclick='select_row(this);' /></td><td style='width:268px;'>$nomeProdotto</td><td>$disponibilita</td><td align='center'>$prezzoProdotto</td><td align='center'> $operazioni$controlliCarrello</td></tr>
                                $schedaProdotto";
                        }//else-if
                        $i++;
                }//while
                $tabellaProdotti .= "<tr><td colspan='7'>Numero record visualizzati: $quanti/$nProdotti</td></tr>";
                $tabellaProdotti .= "<tr><td colspan='7'><form method='GET' ACTION='index.php'>Vai alla pagina: <select name='pagina'>$selectBody</select><input type='submit' value='Vai'></form></td></tr>";
                $tabellaProdotti .= "</table>";
        }//else-if

        if (!($stmt->close()))
                throw new Exception ('chiusura oggetto risultato fallito: (' . $mysqli->errno . ') ' . $mysqli->error);

}catch (Exception $excp) {
   echo $excp -> getMessage();
}
// Interazione col DB terminata. Chiudo la connessione
$mysqli->close();
// Se la pagina e' stata chiamata da una richiesta di aggiungere prodotti al carrello, la processo
if(isset($_POST['aggProd'])){
        if(!(validateID("quantitaP", "quantita' prodotto", false, $avvisi)))
                $isAvvisi = true;
        if(!(validateID("idProd", "ID prodotto", false, $avvisi)))
                $isAvvisi = true;
        if(!($isAvvisi)){
                if(isset($_COOKIE['carrello'])){
```

```
                                $datiCarrello = explode(",",$_COOKIE['carrello']);
                                $nProdottiCarrello = $datiCarrello[0];
                                // Controllo che nel carrello non ci siano gia' altri prodotti dello stesso tipo di quello appena inserito. In questo caso basta aumentare il campo quantita' corrispondente
                                $prodottoGiaInCarrello = FALSE;
                                for($k=1;$k<=($nProdottiCarrello*2);$k=$k+2){
                                        if($datiCarrello[$k]==$_POST['idProd']){
                                                $prodottoGiaInCarrello = TRUE;
                                                $datiCarrello[$k+1] += $_POST['quantitaP'];
                                        }
                                }
                                if(!($prodottoGiaInCarrello)){ // il prodotto non era gia' in carrello

                                        $datiCarrello[0]++; // aumento di 1 il campo che indica il numero di prodotti in carrello
                                        $nuovoCarrello = implode(",",$datiCarrello); // ricompongo il nuovo carrello copiando i dati del vecchio ma con primo campo incrementato di 1
                                        $nuovoCarrello .= ",".$_POST['idProd'].",".$_POST['quantitaP']; // aggiundo ul nuovo prosotto al carrello con la sua quantita'
                                }else{ // il prodotto era gia' in carrello: devo solo ricopiare perche' la quantita' corrispondente e' gia' stata incrementata
                                        $nuovoCarrello = implode(",",$datiCarrello); // ricompongo il nuovo carrello copiando i dati del vecchio ma con primo campo incrementato di 1
                                }
                                // salvo il cookie :)
                                setcookie ("carrello", $nuovoCarrello,time()+$tempoMaxCarrello); // il tempo massimo prima che il carrello scompaia dal computer dell'utente puo' essere impostato dal file di setup
                                header("Location: ".$_SERVER['PHP_SELF']."?addCarrello=true");
                        }else{
                                setcookie ("carrello", "1,".$_POST['idProd'].",".$_POST['quantitaP'],time()+$tempoMaxCarrello); // il tempo massimo prima che il carrello scompaia dal computer dell'utente puo' essere impostato dal file di setup
                                header("Location: ".$_SERVER['PHP_SELF']."?addCarrello=true");
                        }
                }
}
// Leggo i dati relativi al carrello
if(isset($_COOKIE['carrello'])){
        $datiCarrello = explode(",",$_COOKIE['carrello']);
        $nProdottiCarrello = $datiCarrello[0];
}else{
        $nProdottiCarrello = 0;
}
// Se la pagina e' stata richiesta dal link di Logout, eseguo il logout dell'utente connesso
if(isset($_GET['logout']) && $_GET['logout']=="true"){
        session_destroy();
        header("Location: ".$_SERVER['PHP_SELF']."?logout=messaggio");  // valido per tutte le pagine: posso fare copia incolla :)
}
?>
<html>
<head>
<meta charset="UTF-8">
<title>The CakeLab HomePage</title>
<link href="main.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="pack/js/setColor.js"></script>
<script type="text/javascript" src="pack/js/select_row.js"></script>
</head>
<body>
<div id="top">
```

```
        <div id="CakeLabLogo"><img src="media/logo.png" width="220px" height="86" alt="CakeLab Logo"
/></div>
        <div id="menuTop">
        <div id="home" class="pulsanteTop"><a href="index.php">Home</a></div>
      <?php
      if(!(isset($_SESSION['loggedIn']))){ // Se non e' loggato nessun utente permetto ad un eventuale cliente di
registrarsi
                  ?>
        <div id="nuovoUtente" class="pulsanteTop"><a href="addCliente.php">Sono Nuovo</a></div>
      <?php
                  }else if(($_SESSION['loggedIn'] == 'youAreLogged')&&($_SESSION['gruppo_U']==1)){ // se
invece e' loggato un admin gli permetto di creare un nuovo utente, ma con parole diverse :)
                  ?>
      <div id="nuovoUtente" class="pulsanteTop"><a href="addCliente.php">Nuovo Cliente</a></div>
      <?php
                  } // chiusura if-else nessun utente loggato/loggato un admin

                  if((isset($_SESSION['loggedIn']))&&($_SESSION['loggedIn'] ==
'youAreLogged')&&($_SESSION['gruppo_U']==1)){ // E' loggato un admin: solo un admin puo' aggiungere altri admin
                  ?>
      <div id="nuovoAdmin" class="pulsanteTop"><a href="addAdmin.php">Nuovo Admin</a></div>
      <?php
                  }
                  ?>
    </div>
        <div id="login">
        <?php
                  if((isset($_SESSION['loggedIn']))&&($_SESSION['loggedIn'] == 'youAreLogged')) // L'utente è gia'
loggato
                  {
                  ?>
      <div id="loggedBox">Benvenuto <span class="dato"><?php echo $_SESSION['nome_U'];?></span>(<a
class="pulsanteLoggedBox" href="index.php?logout=true">Logout</a>) <hr />
        <div id="iltuoCakeLab">
          <?php
                        if($_SESSION['gruppo_U']==1){ // L'utente loggato e' un admin
                        ?>
          <a class="pulsanteLoggedBox" href="getUtenti.php">Gestisci Utenti</a>
        <?php
                        }else{  // L'utente loggato e' un cliente
                        ?>
                  <a class="pulsanteLoggedBox" href="getOrdini.php">I tuoi ordini</a>
        <?php
        }// chiusura if-else utente loggato admin/cliente
          ?>
        </div>
          <div id="carrello">
        <?php
                        if($_SESSION['gruppo_U']==1){ // L'utente loggato e' un admin
                        ?>
                  <a class="pulsanteLoggedBox" href="getOrdini.php">Gestisci Ordini</a>
          <?php
                        }else{  // L'utente loggato e' un cliente
                        ?>
          <img width="16px" height="16px" src="media/carrello.gif" alt="un piccolo carrello" /><a
class="pulsanteLoggedBox" href="getCarrello.php">Carrello(<?php echo $nProdottiCarrello;?>)</a>
        <?php
        }// chiusura if-else utente loggato admin/cliente
          ?>
        </div>
```

```
	</div>
	<?php
				}else{  // L'utente non e' loggato
		?>
					<form method="POST" action="<?php echo $_SERVER['PHP_SELF']; /*L'invio del form sara' processato da questa stessa pagina*/?>">
					<div id="logTextFields">
						Username: <input type="text" name="user" size="15" /><br />
						Password:  <input type="password" name="pwd" size="15" /><br />
		<?php
							if($erroreLogin){
						?>
				<div class="errore"><?php echo $descErrore;?></div>
		<?php
							}// errore campi login mancanti
						?>
					</div>
					<div id="logPulsante">
						<input type="submit" name="Login" value="Login" />
					</div>
	<div id="carrello"><img width="16px" height="16px" src="media/carrello.gif" alt="un piccolo carrello" /><a class="pulsanteLoggedBox" href="getCarrello.php">Carrello(<?php echo $nProdottiCarrello;?>)</a></div>
					</form>
	<?php
				}// chiusura if-else utente loggato/non loggato
		?>
   </div>
</div>
<div id="center">
<?php   // Se la pagina e' stata richiamata a seguito di tentativi di visitare pagine riservate agli admin visualizzo il relativo messaggio d'errore
if((isset($_GET['errore']))&&($_GET['errore']=="noadmin")){
?>
<div class="risultato">
Non hai il permesso per accedere alla pagina richiesta.
</div>
<?php
}
?>
<?php   // Se la pagina e' stata richiamata a seguito dell'aggiunta di un nuovo prodotto al carrello, lo segnalo
if((isset($_GET['addCarrello']))&&($_GET['addCarrello']=="true")){
?>
<div class="risultato">
Prodotto correttamente aggiunto al carrello.
</div>
<?php
}
?>
<div id="barraRicerca">
<form method="POST" action="<?php echo $_SERVER['PHP_SELF']?>">
Ricerca
<?php
	echo $selectCategorie;
?>
<input class="mainRicerca" type="text" name="valRic" size="100" maxlength="255" >
<input type="submit" name="ricercaProd" value="Vai" >
</form>
</div>
```

```
<?php   // Se la pagina e' stata richiamata a seguito dell'invio del form di creazione utente visualizzo il messaggio con risultato
if($isAvvisi){
?>
<div class="risultato">
<?php echo $avvisi;?>
</div>
<?php
}
?>
<div class="visualizza">
<?php
            echo $tabellaProdotti;
?>
</div>
```
