UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INFORMATICA

Progettazione ed implementazione di una base di dati per la gestione delle finanze personali

Docente
Prof. MARA SANGIOVANNI

Autori
GIANCARLO BRANDI
N86004541

SALVATORE CARLEO
N86004897

Anno accademico 2023/2024

Indice

1. Traccia

SavingMoneyUnina è un sistema che permette di tenere sotto controllo le finanze personali o familiari. Permette di collegare più carte di credito o debito, proprie o di un altro membro della famiglia gestendo le transazioni in entrata ed in uscita. Il sistema permette di suddividere le transazioni in gruppi (portafogli) appartenenti a diverse categorie (es. svago, spese mediche, stipendio, bollette ecc.). È possibile sincronizzare automaticamente le transazioni effettuate da una carta assegnandole ad uno specifico gruppo, oppure registrare una transazione manualmente. Si utilizzino le proprie conoscenze del dominio per definire dettagli non specificati nella traccia.

1.1 Output attesi dal committente

La documentazione, in formato PDF, deve riportare la motivazione delle scelte effettuate ad ogni passaggio, ed i seguenti diagrammi, schemi ed output:

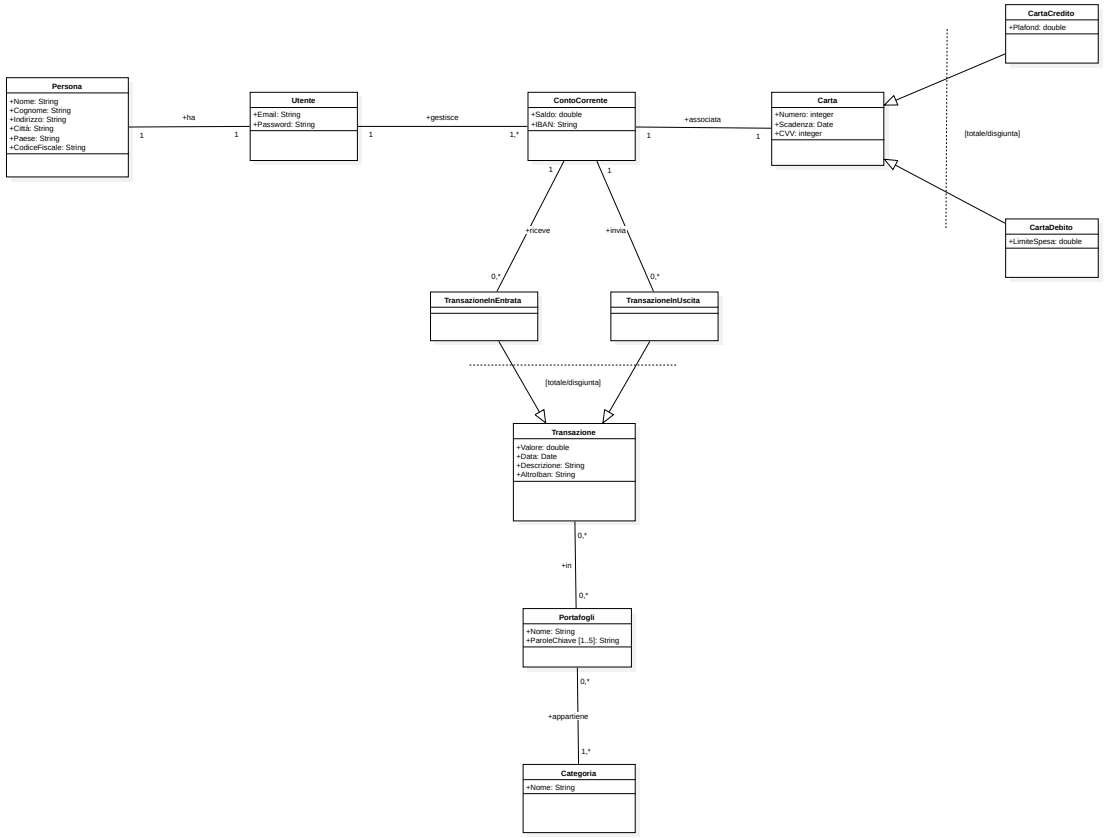
- Progettazione schema della base di dati:
  - Diagramma delle classi UML
  - Diagramma ER (Entità Relazione)
- Ristrutturazione dello schema secondo il modello relazionale:
  - Diagramma delle classi UML ristrutturato
  - Dizionario delle Classi, delle Associazioni e dei Vincoli
  - Schema Logico
- Schema fisico:
  - Struttura SQL delle tabelle
  - Trigger e procedure SQL

2. Progettazione

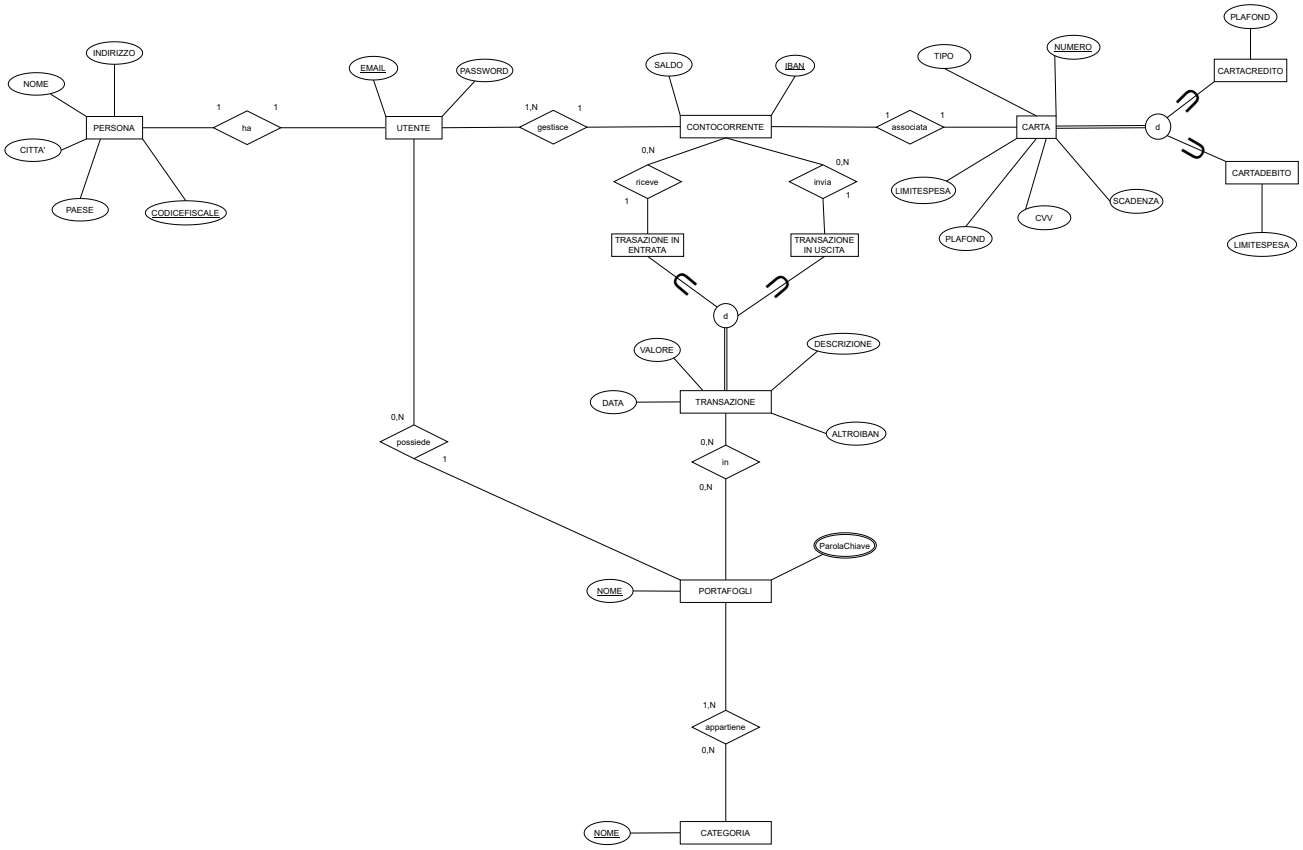
Nel presente capitolo vengono descritti gli elementi costituenti della base di dati e le relazioni che intercorrono tra di essi.

2.1 Schema concettuale

2.1.1 Diagramma UML



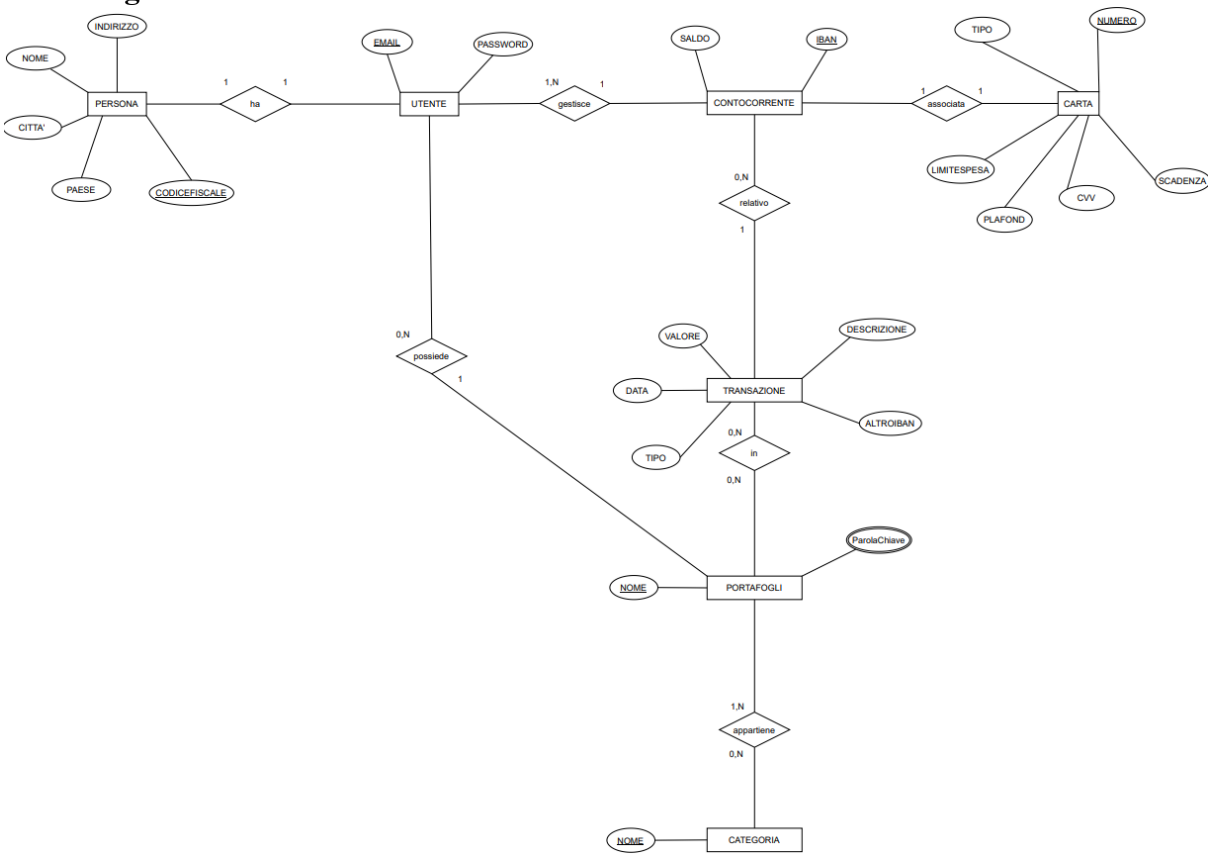
2.1.2 Diagramma Entità Relazione



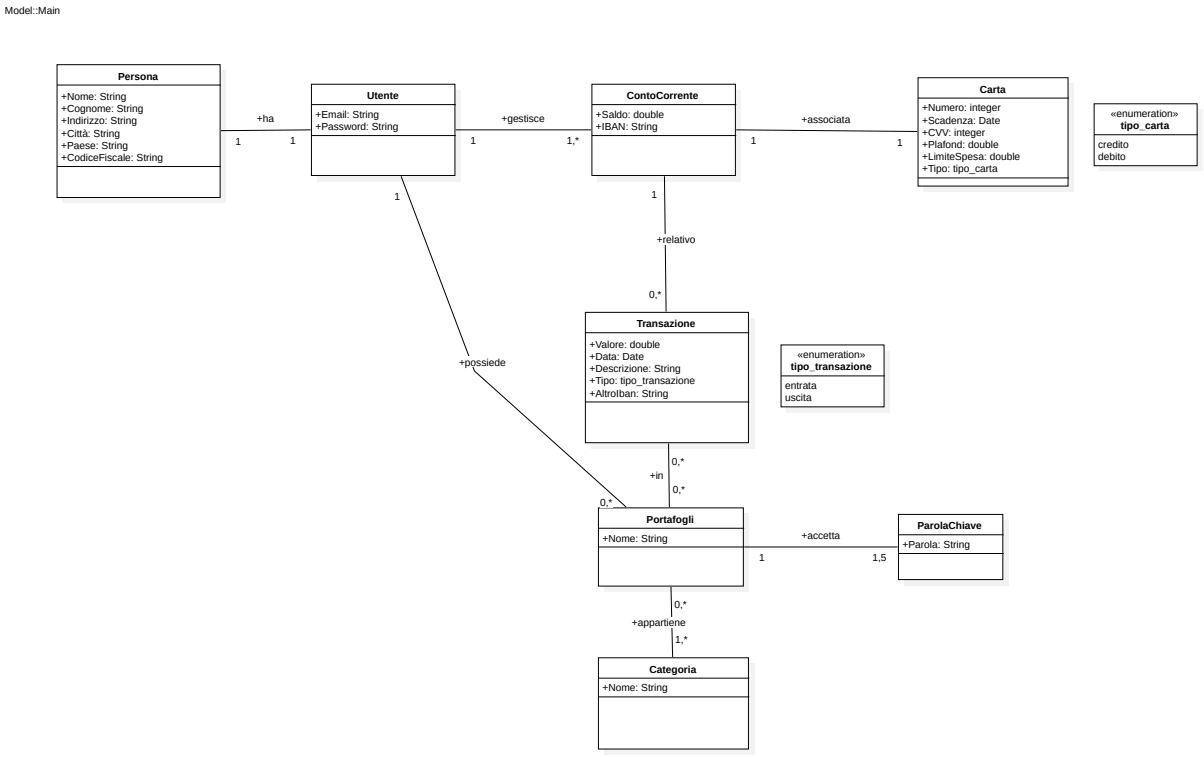
2.2 Ristrutturazione dello schema concettuale

Il passaggio intermedio tra la progettazione concettuale e quella logica è la ristrutturazione, che prevede alcune modifiche.

2.2.2 Diagramma Entità Relazione ristrutturato



2.2.3 Diagramma UML ristrutturato



2.2.3 Rimozione delle gerarchie
Le gerarchie delle classi TRANSIZIONE e CARTA sono state eliminate, e le classi specializzate sono state accorpate nella classe generale. Inoltre è stato introdotto l'attributo *tipo* che fa riferimento ad un'enumerazione, per entrambe le classi.

2.2.4 Rimozione degli attributi multipli
L'attributo multiplo *ParoleChiave* è stato rimosso dalla classe PORTAFOGLI ed è stata introdotta un'altra entità chiamata PAROLECHIAVI al cui interno è presente un attributo *Parola*. Tale entità è posta in associazione con Portafogli.

2.2.5 Rimozione delle associazioni ridondanti
Dopo aver rimosso le classi specializzate di TRANSAZIONE, è stata lasciata una sola associazione con CONTOCORRENTE per evitare ridondanze.

2.2.6 Scelta degli identificatori primari
È stato necessario introdurre delle chiavi surrogate per le classi TRANSAZIONE e PORTAFOGLI: *IdPortafogli* e *IdTransazione*.

2.2.7 Aggiunta di un'associazione tra Utente e Portafogli
Si è ritenuto necessario introdurre un'associazione tra le classi UTENTE e PORTAFOGLI per rendere più immediata la connessione tra un utente e i relativi portafogli gestiti, senza dover passare per CONTOCORRENTE, riducendo di fatto l'overhead al momento della progettazione fisica.

2.3 Dizionari
In questo capitolo è presente una descrizione esaustiva di ciascuna classe con i relativi attributi, associazioni e vincoli.

2.3.1 Dizionario delle classi

CLASSE	DESCRIZIONE	ATTRIBUTI
Persona	Tabella che si occupa di memorizzare i dati di una persona fisica.	`Nome` (totale): Identifica il nome della persona interessata. `Cognome` (totale): Indica il cognome della persona interessata. `Indirizzo` (totale): Specifica l'indirizzo della persona interessata.

		`Città` (totale): Definisce la città in cui vive la persona coinvolta. `Paese` (totale) : Indica il paese in cui vive la persona interessata. `Codice Fiscale` (totale): Chiave primaria della persona.
Utente	Tabella degli utenti registrati	`Email` (totale): Chiave primaria di utente. `Password` (totale): Indica la password registrata dall'utente.
ContoCorrente	Tabella dei conti correnti	`Saldo` (totale): Definisce il saldo dell'utente. `Iban` (totale): Chiave primaria del conto corrente.
Carta	Tabella delle carte associate ai conti correnti	`Numero` (totale): Chiave primaria della carta. `Scadenza` (totale): Indica la scadenza della carta. `CVV` (totale): Specifica il codice di sicurezza della carta. `Tipo` (totale): Tipologia di carta (credito/debito). `Plafond` (parziale): Limite massimo del credito concesso ad una carta di credito. `LimiteSpesa` (parziale): Limite di spesa massima per una carta di credito
Transazione	Tabella delle transazioni	`Valore` (totale): Identifica l'importo della transazione.

		`Data` (totale): Indica la data in cui è avvenuta la transazione. `Descrizione` (parziale): Descrizione della transazione. `AltroIban` (totale): Designa un iban della transazione che può essere del ricevente o dell'inviante. `Tipo` (totale): Stabilisce la transazione in entrata o in uscita.
Portafogli	Tabella contenitore delle transazioni	`Nome` (totale): Identifica il nome del portafoglio. `ParolaChiave` (totale): Indica le parole che attivano la funzione di sincronizzazione automatica.
Categoria	Tabella di categorie	`Nome` (totale): Chiave primaria di Categoria.

2.3.2 Dizionario delle associazioni

ASSOCIAZIONE	DESCRIZIONE	CLASSI COINVOLTE
Ha	Rappresenta l'associazione tra una persona fisica e il suo rispettivo utente registrato	**Persona [1..1]**: Indica la persona fisica **Utente [1..1]**: Rappresenta l'utente registrato
Gestisce	Ogni conto corrente è gestito da un utente registrato.	**Utente [1..1]**: Indica l'utente registrato **ContoCorrente [1..*]**: Indica i conti gestiti dall'utente registrato
Associata	Ad ogni conto corrente è associata una carta di pagamento.	**ContoCorrente [1..1]**: Indica il conto corrente. **Carta [1..1]**: Indica la singola carta associata.
Relativo	Ad ogni conto corrente sono associate più transazioni.	**ContoCorrente [1..1]**: Indica il conto corrente.

		Transazione [0..*]: Indica le transazioni che avvengono sul conto corrente.
In	In ogni portafogli sono presenti più transazioni.	**Transazione [0..*]**: Specifica l'insieme di transazioni. **Portafogli [0..*]**: Indica i diversi portafogli di cui fanno parte.
Accetta	Ogni portafoglio ha da una a cinque parole chiave associate per gestire la funzione di sincronizzazione automatica.	**Portafogli [1..1]**: Indica il portafogli considerato. **ParolaChiave[1..5]**: Indica le parole chiave associate a quel portafogli.
Appartiene	Ogni portafoglio ha una o più categorie associate.	**Portafogli [0..*]**: Indica il portafoglio considerato. **Categoria [1..*]**: Specifica le categorie associate al portafoglio.
Possiede	Ogni utente possiede più portafogli.	**Utente [1..1]**: Indica l'utente proprietario dei portafogli. **Portafogli [0..*]**: Indica i portafogli dell'utente.

2.3.3 Dizionario dei Vincoli

VINCOLO	TIPO	DESCRIZIONE
TIPO_CARTA_CHECK	Dominio	Il tipo di una carta deve essere esclusivamente uguale a "credito" o "debito". Se è di credito, dovrà avere l'attributo `LimiteSpesa` impostato a NULL. Se è di debito, dovrà avere l'attributo `Plafond` impostato a NULL.
TIPO_TRANSAZIONE_CHECK	Dominio	Il tipo di una transazione deve essere esclusivamente uguale a "entrata" o "uscita".

SCADENZA_CARTA_CHECK	Intrarelazionale	Una carta di pagamento, per poter operare, deve avere data di scadenza valida.
LIMITESPESA_PLAFOND_CHECK	Interelazionale	Una carta di credito non può superare il suo plafond di credito. Una carta di debito non può superare il suo limite di spesa.
PAROLECHIAVE_LIMITE_CHECK	Intrarelazionale	Un portafoglio non può avere più di 5 parole chiave associate.
NO_DUPLICATI_PORTAFOGLI	Intrarelazionale	Un portafogli non può contenere più volte la stessa transazione.

3. Progettazione logica

3.1 Schema logico

A tale livello di progettazione, le entità e le associazioni presenti nello schema concettuale ristrutturato vengono tradotte nello schema logico.

3.1.1 Traduzione delle classi

PERSONA

Nome	Cognome	Indirizzo	Città	Paese	CodiceFiscale

UTENTE

Email	Password

CONTOCORRENTE

Saldo	IBAN

CARTA

Numero	Scadenza	CVV	Tipo	Plafond	LimiteSpesa

TRANSAZIONE

IdTransazione	Valore	Data	Descrizione	AltroIban	Tipo

PORTAFOGLI

IdPortafogli	Nome

PAROLACHIAVE

Parola

CATEGORIA

Nome

3.1.2 Traduzione delle associazioni

3.1.2.1 Associazione 1:1 tra PERSONA e UTENTE

Trattandosi di un'associazione 1:1, scegliamo UTENTE come classe debole e vi migriamo la chiave primaria di PERSONA.

PERSONA

Nome	Cognome	Indirizzo	Città	Paese	CodiceFiscale

UTENTE

Email	Password	CodiceFiscale

3.1.2.2 Associazione 1:N tra UTENTE e CONTOCORRENTE

Trattandosi di un'associazione 1:N, bisogna migrare la chiave primaria della classe forte alla classe debole che è CONTOCORRENTE. Migriamo la chiave primaria di UTENTE in CONTOCORRENTE.

UTENTE

Email	Password	CodiceFiscale

CONTOCORRENTE

Saldo	IBAN	Email

3.1.2.3 Associazione 1:1 tra CONTOCORRENTE e CARTA

Trattandosi di un'associazione 1:1 scegliamo CONTOCORRENTE come classe debole e vi migriamo la chiave primaria di CARTA.

CARTA

Numero	Scadenza	CVV	Tipo	Plafond	LimiteSpesa

CONTOCORRENTE

Saldo	IBAN	Email	NumeroCarta

3.1.2.4 Associazione 1:N tra CONTOCORRENTE e TRANSAZIONE

Trattandosi di un'associazione 1:N, bisogna migrare la chiave primaria dalla classe forte alla classe debole che è TRANSAZIONE, migriamo la chiave primaria di CONTOCORRENTE in TRANSAZIONE.

CONTOCORRENTE

Saldo	IBAN	Email	NumeroCarta

TRANSAZIONE

IdTransazione	Valore	Data	Descrizione	AltroIban	Tipo	Iban

3.1.2.5 Associazione M:N tra TRANSAZIONE e PORTAFOGLI

Trattandosi di un'associazione M:N bisogna creare una tabella ponte migrando la chiave primaria di TRANSAZIONE in PORTAFOGLI.

TRANSAZIONE

IdTransazione	Valore	Data	Descrizione	AltroIban	Tipo	Iban

PORTAFOGLI

IdPortafogli	Nome

TRANSAZIONE_PORTAFOGLI

IdTransazione	IdPortafogli

3.1.2.6 Associazione 1:5 tra PORTAFOGLI e PAROLACHIAVE

Consideriamo l'associazione 1:5 come un'associazione 1:N, e migriamo la chiave primaria di PORTAFOGLI in PAROLACHIAVE.

PORTAFOGLI

IdPortafogli	Nome

PAROLACHIAVE

Parola	IdPortafogli

3.1.2.7 Associazione M:N tra PORTAFOGLI e CATEGORIA

Trattandosi di un'associazione M:N bisogna creare una tabella ponte migrando la chiave primaria di PORTAFOGLI e di CATEGORIA.

PORTAFOGLI

IdPortafogli	Nome

CATEGORIA

Nome

PORTAFOGLI_CATEGORIA

IdPortafogli	NomeCategoria

3.1.2.8 Associazione 1:N tra UTENTE e PORTAFOGLI

Trattandosi di un'associazione 1:N, bisogna migrare la chiave primaria dalla classe forte alla classe debole che è PORTAFOGLI, migriamo la chiave primaria di UTENTE in PORTAFOGLI.

UTENTE

Email	Password	CodiceFiscale

PORTAFOGLI

IdPortafogli	Nome	Email

3.2 Schema logico finale

Una volta tradotte tutte le associazioni, lo schema logico si presenta nel seguente modo.

PERSONA

Nome	Cognome	Indirizzo	Città	Paese	CodiceFiscale

UTENTE

Email	Password	CodiceFiscale

CONTOCORRENTE

Saldo	IBAN	Email	NumeroCarta

CARTA

Numero	Scadenza	CVV	Tipo	Plafond	LimiteSpesa

TRANSAZIONE

IdTransazione	Valore	Data	Descrizione	AltroIban	Tipo	Iban

PORTAFOGLI

IdPortafogli	Nome	Email

PAROLACHIAVE

Parola	IdPortafogli

CATEGORIA

Nome

TRANSAZIONE_PORTAFOGLI

IdTransazione	IdPortafogli

PORTAFOGLI_CATEGORIA

IdPortafogli	NomeCategoria

4. Schema fisico

Durante il capitolo, presenteremo il processo di traduzione dello schema logico a quello fisico, mettendo in evidenza elementi principali come tabelle, trigger, procedure e funzioni.

4.1 Tabelle

4.1.1 Carta

```
CREATE TABLE "SavingMoneyUnina".carta (
	numero varchar(16) NOT NULL,
	scadenza date NOT NULL,
	cvv varchar(3) NOT NULL,
	plafond float8 NOT NULL,
	limitespesa float8 NOT NULL,
	tipo varchar NOT NULL,
	CONSTRAINT numero_pk PRIMARY KEY (numero),
	CONSTRAINT scadenza_carta_check CHECK ((scadenza > now())),
	CONSTRAINT tipo_carta_check CHECK ((tipo = 'credito') AND
(limitespesa = 0)) OR ((tipo = 'debito') AND (plafond = 0))
);
```

4.1.2 Categoria

```
CREATE TABLE "SavingMoneyUnina".categoria (
	nome varchar NOT NULL,
	CONSTRAINT nome_pk PRIMARY KEY (nome)
);
```

4.1.3 ContoCorrente

```
CREATE TABLE "SavingMoneyUnina".contocorrente (
	saldo float8 NOT NULL,
	iban varchar(27) NOT NULL,
	email varchar NOT NULL,
	numerocarta varchar(16) NOT NULL,
	CONSTRAINT iban_pk PRIMARY KEY (iban),
	CONSTRAINT email_fk FOREIGN KEY (email) REFERENCES
"SavingMoneyUnina".utente(email) ON DELETE CASCADE,
	CONSTRAINT numerocarta_fk FOREIGN KEY (numerocarta) REFERENCES
"SavingMoneyUnina".carta(numero)
);
```

4.1.4 ParolaChiave

```
CREATE TABLE "SavingMoneyUnina".parolachiave (
	parola varchar NOT NULL,
	idportafogli int4 NOT NULL,
	CONSTRAINT idportafogli_fk FOREIGN KEY (idportafogli) REFERENCES
"SavingMoneyUnina".portafogli(idportafogli)
);
```

4.1.5 Persona

```
CREATE TABLE "SavingMoneyUnina".persona (
	nome varchar(50) NOT NULL,
	cognome varchar(50) NOT NULL,
	indirizzo varchar NULL,
	paese varchar NOT NULL,
	codicefiscale varchar NOT NULL,
	città varchar NOT NULL,
	CONSTRAINT codicefiscale_pk PRIMARY KEY (codicefiscale)
);
```

4.1.6 Portafogli

```
CREATE TABLE "SavingMoneyUnina".portafogli (
	nome varchar NOT NULL,
	idportafogli int4 NOT NULL GENERATED ALWAYS AS IDENTITY( INCREMENT BY
1 MINVALUE 1 MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE),
	CONSTRAINT portafogli_pk PRIMARY KEY (idportafogli)
);
```

4.1.7 Portafogli_Categoria

```
CREATE TABLE "SavingMoneyUnina".portafogli_categoria (
	idportafogli int4 NOT NULL,
	nomecategoria varchar NOT NULL,
	CONSTRAINT idportafogli_fk FOREIGN KEY (idportafogli) REFERENCES
"SavingMoneyUnina".portafogli(idportafogli),
	CONSTRAINT nomecategoria_fk FOREIGN KEY (nomecategoria) REFERENCES
"SavingMoneyUnina".categoria(nome)
);
```

4.1.8 Transazione

```
CREATE TABLE "SavingMoneyUnina".transazione (
	valore float8 NOT NULL,
	"data" date NOT NULL,
	descrizione varchar(200) NOT NULL,
	tipo varchar NOT NULL,
	altroiban varchar(27) NOT NULL,
	idtransazione int4 NOT NULL GENERATED ALWAYS AS IDENTITY( INCREMENT
BY 1 MINVALUE 1 MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE),
	iban varchar(27) NOT NULL,
	CONSTRAINT tipo_transazione_check CHECK ((tipo = 'entrata') OR (tipo
= 'uscita')),
	CONSTRAINT transazione_pk PRIMARY KEY (idtransazione),
	CONSTRAINT iban_fk FOREIGN KEY (iban) REFERENCES
"SavingMoneyUnina".contocorrente(iban) ON DELETE CASCADE
);
```

4.1.9 Transazione_Portafogli

```
CREATE TABLE "SavingMoneyUnina".transazione_portafogli (
	idtransazione int4 NOT NULL,
	idportafogli int4 NOT NULL,
	CONSTRAINT idportafogli_fk FOREIGN KEY (idportafogli) REFERENCES
"SavingMoneyUnina".portafogli(idportafogli),
	CONSTRAINT idtransazione_fk FOREIGN KEY (idtransazione) REFERENCES
"SavingMoneyUnina".transazione(idtransazione)
);
```

4.1.10 Utente

```
CREATE TABLE "SavingMoneyUnina".utente (
	email varchar NOT NULL,
	"password" varchar NOT NULL,
	codicefiscale varchar(16) NOT NULL,
	CONSTRAINT email_pk PRIMARY KEY (email),
	CONSTRAINT codicefiscale_fk FOREIGN KEY (codicefiscale) REFERENCES
"SavingMoneyUnina".persona(codicefiscale) ON DELETE CASCADE
);
```

4.2 Trigger

4.2.1 ParoleChiave_Limite_Check

```
CREATE OR REPLACE FUNCTION "SavingMoneyUnina".parolechiave_limite_check_f()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
declare
	conteggioid_v int4;
begin
	conteggioid_v := 0;

	SELECT COUNT(*)
	INTO conteggioid_v
	FROM "SavingMoneyUnina".parolachiave
	WHERE idportafogli=NEW.idportafogli;

	IF  conteggioid_v >= 5 THEN
		RAISE EXCEPTION 'Ci sono già 5 parole';
	END IF;

	return NEW;
END;

$function$
;
```

```
create trigger parolechiave_check before
insert
    on
    "SavingMoneyUnina".parolachiave for each row execute function
"SavingMoneyUnina".parolechiave_limite_check_f()
```

4.2.2 LimiteSpesa_Plafond_Check

```
CREATE OR REPLACE FUNCTION "SavingMoneyUnina".limitespesa_plafond_check_f()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
      DECLARE
      totale_speso float8;
      tipo_carta varchar;
      limite_spesa float8;
      plafond float8;

      numcarta varchar;
      saldoconto float8;
begin
      select numerocarta, saldo
      from "SavingMoneyUnina".contocorrente
      into numcarta, saldoconto
      where iban=new.iban;

      SELECT C.tipo, C.limitespesa, C.plafond
      FROM "SavingMoneyUnina".Carta as C
      INTO tipo_carta, limite_spesa, plafond
      WHERE numero=numcarta;

      if tipo_carta = 'debito' THEN
            SELECT SUM(valore) INTO totale_speso
            FROM "SavingMoneyUnina".Transazione
            WHERE tipo='uscita'
            GROUP BY ();

            if new.tipo = 'uscita' then
                  totale_speso := totale_speso + new.valore;
            end if;

            IF totale_speso >= limite_spesa THEN
                  RAISE EXCEPTION 'È stato superato il limite di spesa
della carta associata';
            END IF;
      end if;

      IF tipo_carta = 'credito' then
            if new.tipo = 'uscita' then
                  saldoconto := saldoconto - new.valore;
            end if;

            if new.tipo = 'entrata' then
                  saldoconto := saldoconto + new.valore;
            end if;

            if saldoconto < 0 and saldoconto < -plafond THEN
                  raise exception 'È stato superato il plafond';
            end if;
      end if;
      return new;
END;
$function$
;
```

```
create trigger limitespesa_plafond_check before
insert
    on
    "SavingMoneyUnina".transazione for each row execute function
"SavingMoneyUnina".limitespesa_plafond_check_f()
```

4.2.3 No_Duplicati_Portafogli

```
CREATE OR REPLACE FUNCTION "SavingMoneyUnina".no_duplicati_portafogli_f()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
      declare
            f record;
      BEGIN
            for f in (select * from
"SavingMoneyUnina".transazione_portafogli where
idportafogli=new.idportafogli and idtransazione=new.idtransazione) loop
                  raise exception 'Questa transazione è già presente nel
portafogli';
            end loop;
            --il loop parte solo se trova un occorrenza esistente, quindi
errore solo se trova duplicato
            return new;
      END;
$function$
;
```

```
create trigger no_duplicati_portafogli before
insert
    on
    "SavingMoneyUnina".transazione_portafogli for each row execute function
"SavingMoneyUnina".no_duplicati_portafogli_f()
```

4.2.4 Sincronizzazione_Automatica

```
CREATE OR REPLACE FUNCTION
"SavingMoneyUnina".sincronizzazione_automatica_f()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
      declare
            pc record;
      BEGIN
            for pc in (select * from parolachiave) loop
                  if new.descrizione ilike concat('% ',pc.parola,'%') or
new.descrizione ilike concat('%', pc.parola, ' %') or new.descrizione ilike
concat('% ', pc.parola, ' %') or lower(new.descrizione)=lower(pc.parola)
then
                        insert into
"SavingMoneyUnina".transazione_portafogli(idportafogli, idtransazione)
values(pc.idportafogli, new.idtransazione);
                  end if;
            end loop;
            return new;
      END;
$function$;
```

```
create trigger sincronizzazione_automatica after
insert
    on
    "SavingMoneyUnina".transazione for each row execute function
"SavingMoneyUnina".sincronizzazione_automatica_f()
```

4.3 Funzioni e procedure

4.3.1 Inserisci_ParoleChiave_Multiple

```
CREATE OR REPLACE PROCEDURE
"SavingMoneyUnina".inserisci_parolechiave_multiple(IN portafogli integer,
IN lista character varying)
 LANGUAGE plpgsql
AS $procedure$
      DECLARE
      parola varchar;
      contatore int4;
BEGIN
      contatore := 1;
      parola := SPLIT_PART(lista, ',', 1);

      WHILE parola <> '' loop
            INSERT INTO "SavingMoneyUnina".parolachiave(idportafogli,
parola) VALUES(portafogli, parola);
            contatore := contatore + 1;
            parola := SPLIT_PART(lista, ',', contatore);
      END LOOP;

      return;
      END;
$procedure$
;
```

4.3.2 Inserisci_Transazione

```
CREATE OR REPLACE PROCEDURE "SavingMoneyUnina".inserisci_transazione(IN
ibanconto character varying, IN tipo character varying, IN valore double
precision, IN data date, IN descrizione character varying, IN altroiban
character varying)
 LANGUAGE plpgsql
AS $procedure$
      declare
            saldo_conto float8;
            numcarta varchar;
            tipo_carta varchar;
            plafond_carta varchar;
      BEGIN
            select saldo,numcarta
            from "SavingMoneyUnina".contocorrente
            into saldo_conto,numcarta
            where iban = ibanconto;

            select c.tipo,c.plafond
            from "SavingMoneyUnina".carta c
            into tipo_carta,plafond_carta
            where numero = numcarta;

            if tipo = 'uscita' then
                  if tipo_carta = 'debito' and valore > saldo_conto then
                        raise exception 'Saldo insufficiente per completare
la transazione';
                  end if;
            end if;

            if valore < 0 then
                  raise exception 'Importo invalido';
            end if;

            if ibanconto = altroiban then
                  raise exception 'Non è possibile effettuare una
transazione sullo stesso conto';
            end if;

            insert into "SavingMoneyUnina".transazione(valore, data,
descrizione, tipo, altroiban, iban) VALUES(valore, data, descrizione, tipo,
altroiban, ibanconto);

            if tipo = 'entrata' then
                  update "SavingMoneyUnina".contocorrente
                  set saldo = saldo + valore
                  where iban=ibanconto;
            end if;
            if tipo = 'uscita' then
                  update "SavingMoneyUnina".contocorrente
                  set saldo = saldo - valore
                  where iban=ibanconto;
            end if;
      END;
$procedure$
;
```

4.3.3 Calcola_Saldo_Totale

```
CREATE OR REPLACE FUNCTION
"SavingMoneyUnina".calcola_saldo_totale(emailutente character varying)
 RETURNS double precision
 LANGUAGE plpgsql
AS $function$
      declare
            totale float8;
      BEGIN
            totale := 0;
            select SUM(saldo)
            from "SavingMoneyUnina".contocorrente
            into totale
            where email=emailutente;

            return totale;
      END;
$function$
;
```

4.3.4 Crea_Nuovo_Conto

```
CREATE OR REPLACE PROCEDURE "SavingMoneyUnina".crea_nuovo_conto(IN iban
character varying, IN saldo double precision, IN num_carta character
varying, IN scadenza date, IN cvv character varying, IN tipo_carta
character varying, IN limite double precision, IN email character varying)
 LANGUAGE plpgsql
AS $procedure$
      begin
            if tipo_carta = 'credito' then
                  insert into "SavingMoneyUnina".carta VALUES(num_carta,
scadenza, cvv, limite, 0, tipo_carta);
            end if;
            if tipo_carta = 'debito' then
                  insert into "SavingMoneyUnina".carta VALUES(num_carta,
scadenza, cvv, 0, limite, tipo_carta);
            end if;
            insert into "SavingMoneyUnina".contocorrente VALUES(saldo,
iban, email, num_carta);
      END;
$procedure$
;
```

4.3.5 Get_Parolechiave_String

```
CREATE OR REPLACE FUNCTION
"SavingMoneyUnina".get_parolechiave_string(portafogli integer)
 RETURNS character varying
 LANGUAGE plpgsql
AS $function$
      declare
            result varchar;
            parolatrovata record;
      BEGIN
            for parolatrovata in (select parola from
"SavingMoneyUnina".parolachiave where idportafogli=portafogli) loop
                  result := concat(result, parolatrovata.parola, ', ');
            end loop;
            result := rtrim(result, ', ');
            return result;
      END;
$function$
;
```

4.3.6 Report_Mensile

```
CREATE OR REPLACE FUNCTION "SavingMoneyUnina".report_mensile(mese integer,
anno integer, utente character varying)
 RETURNS character varying
 LANGUAGE plpgsql
AS $function$
      declare
            risultato varchar;

            conto record;

            entrata_max float8;
            entrata_avg float8;
            entrata_min float8;

            uscita_max float8;
            uscita_avg float8;
            uscita_min float8;

            saldo_iniziale float8;
            saldo_finale float8;

            tr record;
      BEGIN
            for conto in (select * from "SavingMoneyUnina".ContoCorrente
where email = utente) loop

                  entrata_max := 0;
                  entrata_avg := 0;
                  entrata_min := 0;
                  uscita_max := 0;
                  uscita_avg := 0;
                  uscita_min := 0;
                  saldo_iniziale := 0;
                  saldo_finale := 0;

                  select MAX(valore), AVG(valore), MIN(valore)
                  from "SavingMoneyUnina".transazione
                  into entrata_max, entrata_avg, entrata_min
                  where iban = conto.iban and tipo = 'entrata' and
extract(month from data) = mese and extract(year from data) = anno
                  group by();

                  select MAX(valore), AVG(valore), MIN(valore)
                  from "SavingMoneyUnina".transazione
                  into uscita_max, uscita_avg, uscita_min
                  where iban = conto.iban and tipo = 'uscita' and
extract(month from data) = mese and extract(year from data) = anno
                  group by();

                  for tr in (select * from "SavingMoneyUnina".transazione
where iban = conto.iban and data <= concat(anno,'-',mese,'-','1')::date)
loop
                        if tr.tipo = 'entrata' then
                              saldo_iniziale := saldo_iniziale + tr.valore;
                        else
                              saldo_iniziale := saldo_iniziale - tr.valore;
                        end if;
                  end loop;
```

```
                for tr in (select * from "SavingMoneyUnina".transazione
where iban = conto.iban and data <= (concat(anno,'-',mese,'-','1')::date +
interval '1 month' - interval '1 day')) loop --ultimo giorno del mese, gli
aggiungo 1 mese e gli tolgo 1 giorno
                      if tr.tipo = 'entrata' then
                            saldo_finale := saldo_finale + tr.valore;
                      else
                            saldo_finale := saldo_finale - tr.valore;
                      end if;
                end loop;

                if entrata_max is null then
                      entrata_max := 0;
                end if;
                if entrata_avg is null then
                      entrata_avg := 0;
                end if;
                if entrata_min is null then
                      entrata_min := 0;
                end if;
                if uscita_max is null then
                      uscita_max := 0;
                end if;
                if uscita_avg is null then
                      uscita_avg := 0;
                end if;
                if uscita_min is null then
                      uscita_min := 0;
                end if;

                risultato := concat(risultato, conto.iban, ';',
entrata_max, ';', entrata_avg, ';', entrata_min, ';', uscita_max, ';',
uscita_avg, ';', uscita_min, ';', saldo_iniziale, ';', saldo_finale, ',');
            end loop;
            risultato := rtrim(risultato, ',');
            return risultato;
      END;
$function$
;
```