UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e Delle Scienze di Base

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea in Informatica

Insegnamento di Basi di Dati

Anno accademico 2023/2024

Progettazione e sviluppo di una Base di Dati relazionale per la gestione di un sistema di risparmio

Autori:
Miriam Gaetano
N86004605

Giulia Gargiulo
N86004670

Fortunata D'Urso
N86004687

Docente:
Prof. Mara Sangiovanni

Documentazione progetto

Fortunata D'Urso, Miriam Gaetano, Giulia Gargiulo

2023-2024

Indice

1 Descrizione del progetto

1.1 Analisi del problema

SavingMoneyUnina è un sistema che permette di tenere sotto controllo le finanze personali o familiari. All'interno della documentazione andremo ad analizzare le scelte da noi effettuate per una buona gestione del database.

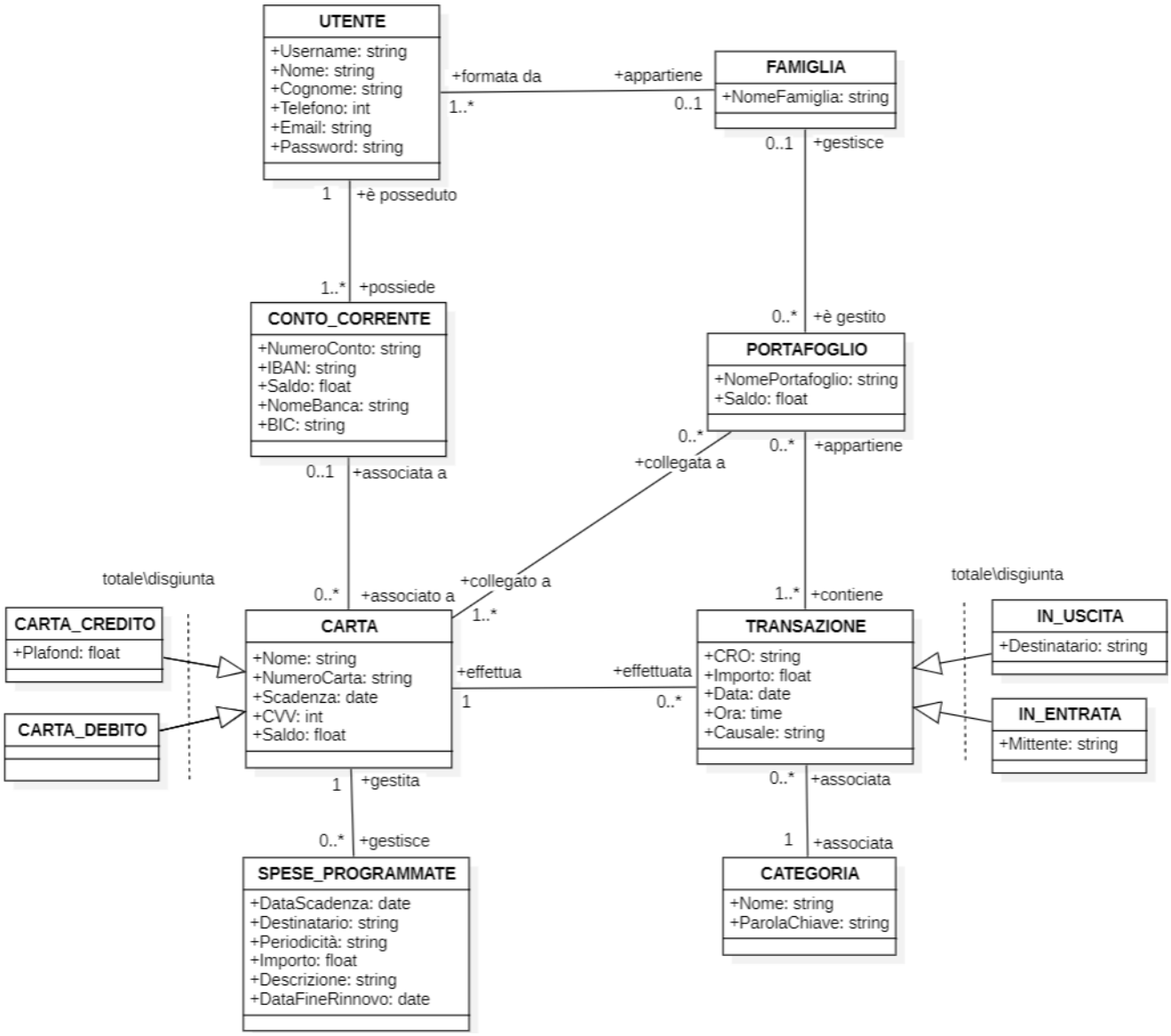
All'interno del sistema SavingMoneyUnina l'utente potrà collegare una o più carte di credito o/e debito con i rispettivi conti correnti e gestire le transazioni in entrata ed uscita le quali verranno categorizzate, automaticamente o manualmente, in diverse categorie prefornite o create dall'utente stesso.

L'utente inoltre potrà creare dei propri portafogli collegati ad una specifica carta, all'interno dei quali potrà decidere di inserire determinate transazioni. Sarà possibile visualizzare le spese effettuate di ogni portafoglio.
I portafogli potranno essere condivisi con il gruppo familiare in modo tale da rendere pubbliche le spese di quel determinato portafoglio anche per i restanti membri del gruppo familiare.

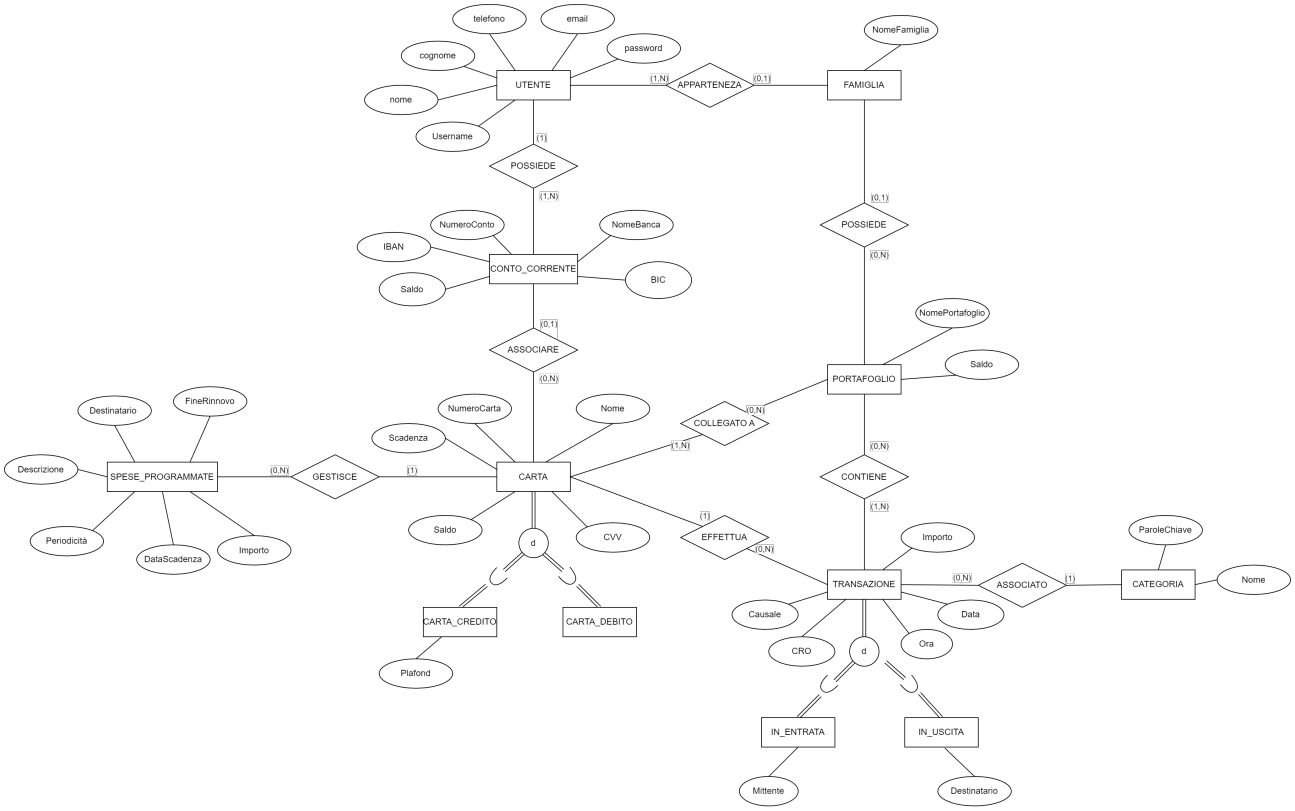
Infine si ha la possibilità di gestire delle spese ricorrenti, permettendo all'utente di impostare pagamenti ricorrenti o spese future, come le bollette mensili o le rate di un prestito. Le spese programmate verranno automaticamente contabilizzate nelle categorie di appartenenza, contribuendo a una migliore pianificazione finanziaria.

2 Progettazione concettuale

2.1 Diagramma delle classi UML



2.2 Diagramma ER



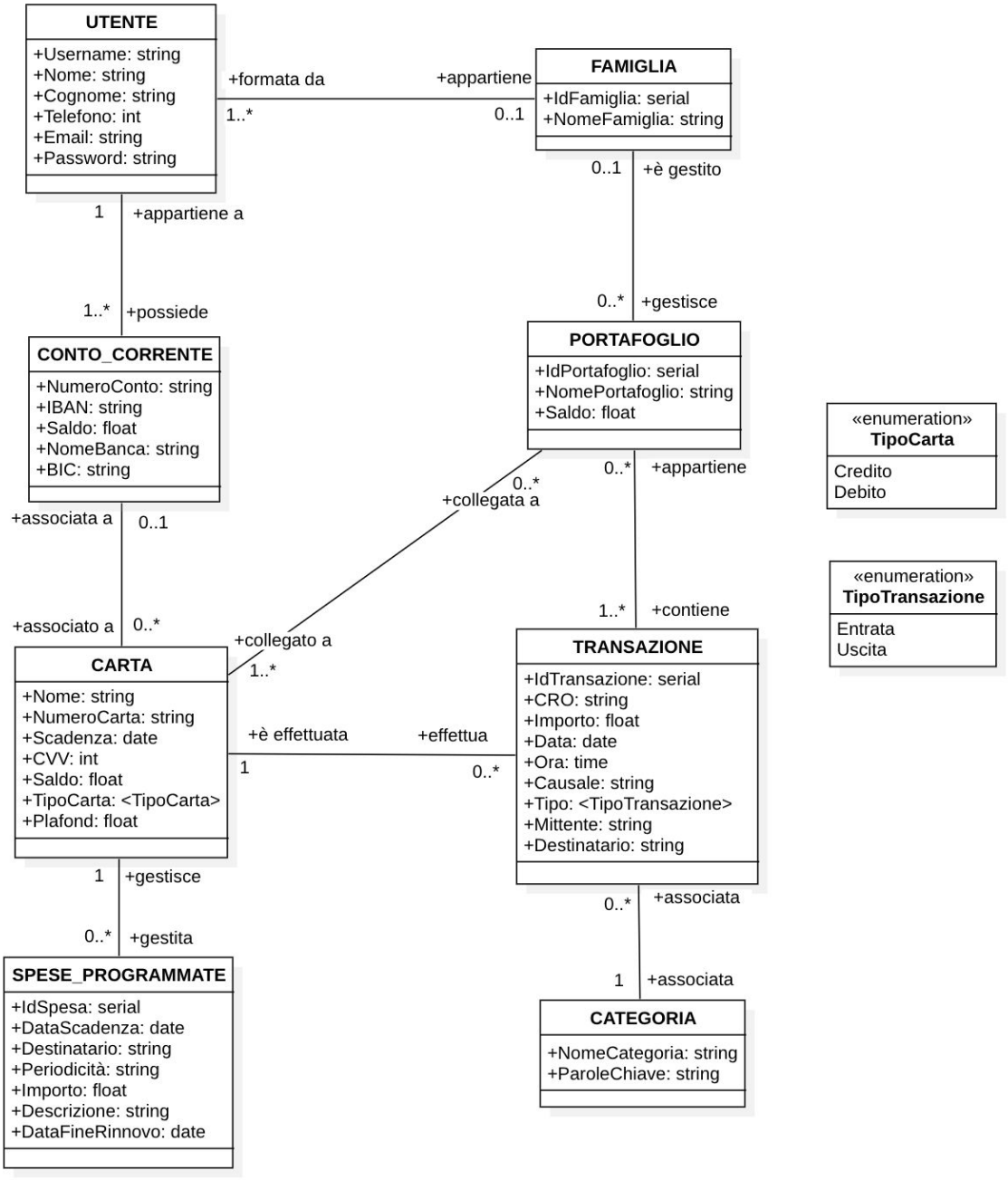
3 Ristrutturazione

3.1 Gestione delle relazioni totali/disgiunte

Durante la ristrutturazione del modello relazionale abbiamo deciso di gestire le relazioni totali/disgiunte portando le classi figlie all'interno della classe padre con l'aggiunta di un nuovo attributo di tipo enumerazione.
Per quanto riguarda le classi figlie "Carta di credito" e "Carta di debito" tutti i loro attributi sono stati spostati nella classe padre "Carta" e abbiamo aggiunto il nuovo attributo "TipoCarta". Quest'ultimo ci garantisce che ogni istanza di Carta sia chiaramente associata ad un solo tipo di carta, evitando ambiguità nelle relazioni e migliorando la coerenza del modello.
Il principale ruolo di "TipoCarta" è quello di identificare il tipo specifico di carta associato a ciascuna istanza della classe "Carta". Con le costanti definite nell'enumerazione ("Debito" o "Credito"), possiamo immediatamente capire se abbiamo a che fare con una carta di debito o una carta di credito.
Abbiamo attuato lo stesso ragionamento anche per quanto riguarda le sottoclassi "Transazione in entrata" e "Transazione in uscita" aggiungendo come nuovo attributo il "TipoTransazione".

Inoltre abbiamo deciso di aggiungere alle classi "Famiglia", "Transazione", "Portafoglio" e "SpeseProgrammate" gli Id in modo tale da rendere univoche le varie classi e riuscire a gestire meglio le relazioni tra le varie classi.

3.2 Diagramma delle classi UML ristrutturato



4 Dizionari

4.1 Dizionario delle classi

CLASSE	*ATTRIBUTI*	*DESCRIZIONE*
Famiglia	***IdFamiglia (serial):*** codice identificativo di un gruppo familiare. ***NomeFamiglia (string):*** nome di un gruppo famiglia.	Insieme di utenti
Utente	***Username (string):*** identificativo personale dell'utente. ***Nome (string):*** nome del titolare della/e carta/e. ***Cognome (string):*** cognome del titolare della/e carta/e. ***Telefono (string):*** recapito telefonico dell'utente. ***Email (string):*** email utente. ***Password (string):*** codice alfanumerico per accedere al proprio account.	Classe che conserva le informazioni relative ai singoli utenti.
ContoCorrente	***NumeroConto (serial):*** codice identificativo associato al conto. ***IBAN (string):*** si riferisce al numero di conto bancario espresso nel formato IBAN, è un identificativo standardizzato utilizzato per identificare univocamente un conto bancario a livello internazionale. ***Saldo (float):*** importo totale posseduto sul conto bancario. ***NomeBanca (string):*** nome dell'istituzione finanziaria presso la quale è aperto il conto. ***BIC (string):*** il Codice Identificativo Bancario è un codice alfanumerico che identifica in modo univoco una specifica istituzione finanziaria in una transazione finanziaria internazionale.	Classe che conserva le informazioni relative ai conti correnti.
Carta	***NumeroCarta (serial):*** codice identificativo di una carta. ***Nome (string):*** nome che l'utente da alla propria carta per distinguerla dalle altre di sua proprietà. ***Scadenza (date):*** data in cui scade la carta. ***Saldo (float):*** importo totale posseduto sulla carta. ***TipoCarta (TipoCarta):*** permette di distinguere la carta di credito da quella di debito e viceversa. ***Plafond (float):*** il plafond di una carta di credito si riferisce al limite massimo di spesa che il titolare della carta può effettuare in un mese. Questo limite è stabilito dalla banca o dall'istituto finanziario emittente. ***CVV (string):*** codice di sicurezza di 3 cifre utilizzato per verificare l'autenticità delle transazioni online.	Classe che conserva le informazioni relative alle carte di credito o debito.

Transazione	***IdTransazione (serial):*** codice identificativo di una transazione permette di distinguere le varie transazioni. ***CRO (string):*** Codice di Riferimento Operazione è un identificativo univoco assegnato a ciascuna transazione finanziaria in grado di tracciare una specifica operazione all'interno del sistema bancario. ***Importo (float):*** quantità di denaro che viene aggiunta o rimossa dal saldo attraverso un deposito, un pagamento ricevuto o un pagamento. ***Data (date):*** data in cui è stata effettuata la transazione. ***Ora (time):*** ora in cui è stata effettuata la transazione. ***Causale (string):*** motivo della transazione. ***Tipo (TipoTransazione):*** permette di distinguere le transazioni in entrata da quelle in uscita e viceversa. ***Mittente (string):*** è l'IBAN del possessore della carta che effettua una transazione. ***Destinatario (string):*** l'IBAN del destinatario è il numero di conto bancario dell'utente che riceve il denaro.	Classe che conserva le informazioni relative a ciascuna transazione in entrata o in uscita.
Categoria	***NomeCategoria (string):*** nome che identifica una categoria. ***ParolaChiave (string):*** serie di parole chiavi che rappresentano una determinata categoria	Classe che gestisce in quale modo le varie transazioni vengono suddivise in base a attributi comuni.
SpeseProgrammate	***IdSpesa (serial):*** codice identificativo della spesa programmata. ***Periodicità (string):*** intervallo di tempo entro il quale la spesa va ripetuta. ***DataScadenza (date):*** data entro cui deve essere effettuato il pagamento. ***Importo (float):*** importo della spesa programmata. ***Destinatario (string):*** destinatario della spesa programmata. ***FineRinnovo (date):*** data entro la quale termina la spesa programmata. ***Descrizione (string):*** indica il motivo della spesa programmata.	Classe che gestisce le informazioni riguardanti le spese programmate.

4.2 Dizionario delle associazioni

RELAZIONI	*ATTRIBUTI*	*DESCRIZIONE*
appartiene - formato da	***IdFamiglia [1,*] ruolo (formato da)***: indica il gruppo a cui appartengono uno o più utenti. ***Username [o,1] ruolo (appartiene)***: indica l'username dell'utente che può appartenere ad una famiglia.	Lega ogni utente ad un gruppo familiare se ne hanno uno.
possiede - appartiene a	***Username [1,*] ruolo (possiede)***: indica l'username dell'utente che possiede uno o più conti corrente. ***NumeroConto [1] ruolo (appartiene a)***: indica il numero del conto posseduto da un unico utente.	Lega ogni conto ad un utente.
associato a - associata a	***NumeroConto [0,*] ruolo (associato a)***: indica la/le carte associte ad un determinato conto corrente. ***NumeroCarta [0,1] ruolo (associata a)***: indica il conto corrente a ci è associata una carta.	Lega ogni carta ad un conto corrente.
è effettuata - effettua	***NumeroCarta [0,*] ruolo (effettua)***: indica la/le varie transazioni che la carta può effettuare. ***CRO [1] ruolo (è effettuata)***: indica la carta tramite cui è stata effettuata la transazione.	Lega ogni transazione ad una carta.
collegato a - collegata a	***NumeroCarta [0,*] ruolo (collegata a)***: indica la possibilità di una carta di poter essere collegata o meno a portafogli. ***IdPortafoglio [1,*] ruolo (collegato a)***: indica la/le carte collegate ad un determinato portafoglio.	Lega o meno una o più carte ad un portafoglio.
appartiene - contiene	***IdTransazione [0,*] ruolo (appartiene)***: indica la possibilità di una transazione di appartenere a nessuno o più portafogli. ***IdPortafoglio [1,*] ruolo (contiene)***: indica la/le transazioni contenute in quel portafoglio.	Lega ad ogni portafoglio una o più transazioni.
è gestito - gestisce	***IdFamiglia [0,*] ruolo (gestisce)***: indica la possibilità di un gruppo familiare di avere o meno più portafogli in comune. ***IdPortafoglio [0,1] ruolo (è gestito)***: indica la possibilità di un portafoglio di essere gestito da un intero gruppo familiare.	Lega determinati portafoglio con il resto del gruppo familiare.
associata - associata	***IdCategoria [0,*] ruolo (associata)***: indica la possibilità di una categoria di contenere nessuna o più transazioni. ***IdTransazione [1] ruolo (associata)***: indica la categoria associata ad una transazione.	Lega ogni transazione ad una categoria.

gestisce - gestita	***NumeroCarta [0,*] ruolo (gestisce)***: indica la possibilità di una carta di gestire nessuna o più spese programmate. ***IdSpesa [1] ruolo (gestita)***: indica la spesa programmata gestita da una determinata carta.	Lega ogni spesa programmata ad una carta.

4.3 Dizionario dei vincoli

VINCOLO	*TIPO*	*DESCRIZIONE*
CK_Telefono	Dominio	Il numero di telefono deve essere formato da un "+", il prefisso internazionale e 10 cifre da 0 a 9.

CK_Email	Dominio	L'e-mail deve avere la seguente forma:
		`[a-z, A-Z, 0-9, ".","_","\%","+","\_","-"]`
		`@ [a-z, A-Z, 0-9, ".","-"] . [a-z, A-Z]`
		(da 2 a 4 caratteri).

CK_Password	Dominio	La password deve contenere da 8 a 32 caratteri tra i caratteri seguenti: `a-z, A-Z, 0-9, !, ", #, %, &, (, ), *, +, ",", -, ., , :, ";", <, =, >, ?, @, [, ], , ^, _ , \{, \} , .`
CK_IBAN	Dominio	L'IBAN deve essere formato da: 2 caratteri tra A-Z, 2 cifre tra 0-9, un carattere tra A-Z, 5 cifre tra 0-9, 5 caratteri tra 0-9, A-Z .
CK_BIC	Dominio	Il BIC deve essere formato da: 6 caratteri tra A-Z, 2 caratteri tra 0-9 e A-Z, da 0 a 3 caratteri tra 0-9, A-Z.
CK_NumeroCarta	Dominio	Il numero della carta deve essere composto da 16 cifre tra 0-9.
CK_CVV	Dominio	Il CVV deve essere formato da 3 cifre tra 0-9 .
CK_TipoCarta	Dominio	Il tipo di Carta deve essere o "Credito" o "Debito".
CK_Periodicità	Dominio	La periodicità deve essere "7 giorni" o "15 giorni" o "1 mese" o "3 mesi" o "6 mesi" o "1 anno".
CK_CRO	Dominio	Il CRO di una transazione deve essere formato da: un numero variabile da 11 a 16 di cifre da 0-9.
CK_Tipo	Dominio	Il Tipo di una transazione deve essere "Entrata" o "Uscita".
CK_Data	Dominio	La data di una transazione deve essere precedente o uguale alla data attuale.

5 Progettazione Logica

5.1 Schema Logico

Famiglia (IdFamiglia, NomeFamiglia)

Utente (Username, Nome, Cognome, Telefono, Email, Password, IdFamiglia)

FK_Famiglia → IdFamiglia

ContoCorrente (NumeroConto, IBAN, Saldo, NomeBanca, BIC, Username)

FK_Utente → Username

Carta (NumeroCarta, Nome, CVV, Scadenza, Saldo, TipoCarta, Plafond, NumeroConto)

FK_Conto → NumeroConto

Categoria (NomeCategoria, ParoleChiavi)

Transazione (IdTransazione, CRO, Importo, Data, Ora, Causale, Tipo, Mittente, Destinatario, NumeroCarta, NomeCategoria)

FK_Carta → NumeroCarta
FK_Categoria → NomeCategoria

Portafoglio (IdPortafoglio, NomePortafoglio, Saldo, IdFamiglia)

FK_Famiglia → IdFamiglia

SpeseProgrammate (IdSpesa, Descrizione, Periodicita, DataScadenza, DataFineRinnovo, Importo, Destinatario, NumeroCarta)

FK_Carta → NumeroCarta

AssociazioneCartaPortafoglio (IdPortafoglio, NumeroCarta)

FK_Carta → NumeroCarta
FK_Portafoglio → IdPortafoglio

TransazioniInPortafogli (IdTransazione, IdPortafoglio)

FK_Transazione → IdTransazione
FK_Portafoglio → IdPortafoglio

6 Schema Fisico

6.1 Creazione Database

```
DROP SCHEMA IF EXISTS smu CASCADE;
CREATE SCHEMA smu;
```

6.2 Creazione Tabelle

6.2.1 Definizione della tabella FAMIGLIA

```
CREATE TABLE smu.FAMIGLIA(
    IdFamiglia      SERIAL,
    NomeGruppo      VARCHAR(32) NOT NULL,

    CONSTRAINT PK_famiglia PRIMARY KEY (IdFamiglia)
);
```

6.2.2 Definizione della tabella UTENTE

```
CREATE TABLE UTENTE(
    Username        VARCHAR(32),
    Nome            VARCHAR(32) NOT NULL,
    Cognome         VARCHAR(32) NOT NULL,
    Telefono        VARCHAR(13) NOT NULL,
    Email           VARCHAR(128)NOT NULL,
    Password        VARCHAR(32) NOT NULL,
    IdGruppo        INTEGER


    CONSTRAINT PK_Utente PRIMARY KEY (Username),
    CONSTRAINT FK_Famiglia FOREIGN KEY(IdGruppo) REFERENCES
            Famiglia(IdGruppo) ON DELETE CASCADE,
    CONSTRAINT UK_Utente UNIQUE (Email, Password),

    CONSTRAINT CK_Telefono CHECK (Telefono ~ '\+[0-9]{2}[0-9]{10}'),
    CONSTRAINT CK_Email CHECK (Email ~ '[a-zA-Z0-9._%+\-]@[a-zA-Z0-9.-]\d*.*[A-Za-z]{2,4}'),
    CONSTRAINT CK_Password CHECK (Password ~ '[a-zA-Z0-9! " # $ % & ( ) * + , - . / : ;
                                < = > ? @  \[ \] \ ^ _‘ \{ | \} ~ ]{8,32}')
);
```

6.2.3 Definizione della tabella CONTO CORRENTE

```
CREATE TABLE smu.CONTOCORRENTE(
    NumeroConto VARCHAR(12),
    IBAN        VARCHAR(27)  UNIQUE NOT NULL,
    Saldo       FLOAT        NOT NULL,
    NomeBanca   VARCHAR(128) NOT NULL,
    BIC         VARCHAR(11)  UNIQUE NOT NULL,
    Username    VARCHAR(32),


    CONSTRAINT PK_Conto PRIMARY KEY (NumeroConto),
    CONSTRAINT FK_Utente FOREIGN KEY(Username) REFERENCES
            smu.Utente(Username) ON DELETE CASCADE,
```

```
    CONSTRAINT CK_IBAN CHECK (IBAN ~ '[A-Z]{2}[0-9]{2}[A-Z]{1}[0-9]{5}[0-9]{5}[0-9A-Z]{5}'),
    CONSTRAINT CK_BIC CHECK (BIC ~ '[A-Z]{4}[A-Z]{2}[0-9A-Z]{2}[0-9A-Z]{0,3}')
);
```

6.2.4 Definizione della tabella CARTA

```
CREATE TABLE smu.CARTA(
    NumeroCarta VARCHAR(16),
    Nome        VARCHAR(32),
    CVV         VARCHAR(3) NOT NULL,
    Scadenza    DATE       NOT NULL,
    Saldo       FLOAT      NOT NULL,
    TipoCarta   VARCHAR(7) NOT NULL,
    Plafond     FLOAT,
    NumeroConto VARCHAR(16),


    CONSTRAINT PK_Carta PRIMARY KEY (NumeroCarta),
    CONSTRAINT FK_Conto FOREIGN KEY (NumeroConto) REFERENCES
            smu.ContoCorrente (NumeroConto),

    CONSTRAINT CK_NumeroCarta CHECK (NumeroCarta ~ '[0-9]{16}'),
    CONSTRAINT CK_CVV CHECK (CVV ~ '[0-9]{3}'),
    CONSTRAINT CK_TipoCarta CHECK (TipoCarta IN ('Credito', 'Debito'))
);
```

6.2.5 Definizione della tabella SPESE PROGRAMMATE

```
CREATE TABLE smu.SPESEPROGRAMMATE(
    IdSpesa         SERIAL,
    Descrizione     VARCHAR(64),
    Periodicita     VARCHAR(9) NOT NULL,
    DataScadenza    DATE       NOT NULL,
    DataFineRinnovo DATE,
    Importo         FLOAT      NOT NULL,
    Destinatario    VARCHAR(32),
    NumeroCarta     VARCHAR(16),


    CONSTRAINT PK_Spesa PRIMARY KEY (IdSpesa),
    CONSTRAINT FK_Carta FOREIGN KEY (NumeroCarta) REFERENCES
            smu.Carta (NumeroCarta) ON DELETE CASCADE,
    CONSTRAINT CK_Periodicita CHECK (Periodicita IN (
                                        '7 giorni', '15 giorni', '1 mese',
                                        '3 mesi', '6 mesi', '1 anno'))
);
```

6.2.6 Definizione della tabella CATEGORIA

```
CREATE TABLE CATEGORIA(
    NomeCategoria  VARCHAR(32),
    ParolaChiavi   VARCHAR(256)  NOT NULL,

    CONSTRAINT PK_Nome PRIMARY KEY (NomeCategoria)
);
```

6.2.7 Definizione della tabella TRANSAZIONE

```
CREATE TABLE smu.TRANSAZIONE(
    IdTransazione SERIAL,
    CRO           VARCHAR(16),
    Importo       FLOAT NOT NULL,
    Data          DATE  NOT NULL,
    Ora           TIME  NOT NULL,
    Causale       VARCHAR(128),
    Tipo          VARCHAR(7),
    Mittente      VARCHAR(32),
    Destinatario  VARCHAR(32),
    NumeroCarta   VARCHAR(16),
    NomeCategoria VARCHAR(32),


    CONSTRAINT PK_Transazione PRIMARY KEY (IdTransazione),
    CONSTRAINT FK_Carta FOREIGN KEY (NumeroCarta) REFERENCES
            smu.Carta (NumeroCarta) ON DELETE CASCADE,
    CONSTRAINT FK_Categoria FOREIGN KEY (NomeCategoria) REFERENCES
            smu.Categoria (NomeCategoria) ON DELETE CASCADE,

    CONSTRAINT CK_CRO CHECK (CRO ~ '[0-9]{11,16}'),
    CONSTRAINT CK_Tipo CHECK (Tipo IN ('Entrata', 'Uscita')),
    CONSTRAINT CK_Data CHECK (data <= CURRENT_DATE)
);
```

6.2.8 Definizione della tabella PORTAFOGLIO

```
CREATE TABLE PORTAFOGLIO(
    IdPortafoglio   SERIAL,
    NomePortafoglio VARCHAR(32) NOT NULL,
    Saldo           FLOAT       NOT NULL,
    IdFamiglia      INTEGER,

    CONSTRAINT PK_Portafoglio PRIMARY KEY (IdPortafoglio),
    CONSTRAINT FK_Famiglia FOREIGN KEY (IdFamiglia) REFERENCES
            smu.Famiglia (IdFamiglia) ON DELETE CASCADE

);
```

6.2.9 Definizione delle tabelle ponte

```
--tabella ponte tra Portafoglio e Carta
CREATE TABLE smu.AssociazioneCartaPortafoglio(
    IdPortafoglio INTEGER,
    NumeroCarta   VARCHAR(16),

    CONSTRAINT FK_Carta FOREIGN KEY (NumeroCarta) REFERENCES
            smu.Carta (NumeroCarta) ON DELETE CASCADE,
    CONSTRAINT FK_Portafoglio FOREIGN KEY (IdPortafoglio) REFERENCES
            smu.Portafoglio (IdPortafoglio) ON DELETE CASCADE
);


--tabella ponte tra Portafoglio e Transazione
CREATE TABLE smu.TransazioniInPortafogli(
    IdTransazione INTEGER,
    IdPortafoglio INTEGER,

    CONSTRAINT FK_Transazione FOREIGN KEY (IdTransazione) REFERENCES
            smu.Transazione (IdTransazione) ON DELETE CASCADE,
    CONSTRAINT FK_Portafoglio FOREIGN KEY (IdPortafoglio) REFERENCES
            smu.Portafoglio (IdPortafoglio) ON DELETE CASCADE
);
```

6.3 Definizione dei trigger

6.3.1 Trigger ModificaImporto

Trigger che imposta a negativo il valore di una transazione in uscita ed aggiorna il valore del saldo della carta e del conto a seguito di una transazione.

```
CREATE OR REPLACE FUNCTION smu.triggerTransazione() RETURNS TRIGGER AS
$$
BEGIN
    -- imposta l'importo delle transazioni in uscita a negativo.
    IF NEW.Tipo = 'Uscita' THEN
        UPDATE smu.Transazione
        SET Importo = -(NEW.Importo)
        WHERE IdTransazione = NEW.IdTransazione;
    END IF;

    --aggiorna il saldo della carta.
    UPDATE smu.Carta
    SET Saldo = Saldo + (SELECT T.Importo
                         FROM smu.Transazione AS T
                         WHERE T.IdTransazione = NEW.IdTransazione)
    WHERE NumeroCarta = NEW.NumeroCarta;


    --aggiorna il saldo del conto.
    UPDATE smu.ContoCorrente
    SET Saldo = Saldo + (SELECT T.Importo
                         FROM smu.Transazione AS T
                         WHERE T.IdTransazione = NEW.IdTransazione)
    WHERE NumeroConto = (SELECT Ca.NumeroConto
                         FROM smu.Carta AS Ca
                         WHERE Ca.NumeroCarta = NEW.NumeroCarta);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER ModificaImporto
    AFTER INSERT
    ON smu.Transazione
    FOR EACH ROW
EXECUTE FUNCTION smu.triggerTransazione();
```

6.3.2 Trigger tipoTransazione

Trigger che prima dell'inserimento di una transazione controlli che: se è di tipo 'Uscita' allora Mittente è NULL, se è di tipo 'Entrata' allora Destinatario è NULL.

```
    CREATE OR REPLACE FUNCTION smu.triggerTipoTransazione() RETURNS TRIGGER AS
    $$
    BEGIN
        IF NEW.Tipo = 'Uscita' AND NEW.Mittente IS NOT NULL THEN
            RAISE EXCEPTION 'ERRORE: Mittente non nullo in una transazione in uscita';
        END IF;

        IF NEW.Tipo = 'Entrata' AND NEW.Destinatario IS NOT NULL THEN
            RAISE EXCEPTION 'ERRORE: Destinatario non nullo in una transazione in entrata';
        END IF;

        RETURN NEW;

    END;
    $$LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ControlloTipoTransazione
    BEFORE INSERT ON smu.Transazione
    FOR EACH ROW EXECUTE FUNCTION smu.triggerTipoTransazione();
```

Insert per testare il trigger:

```
INSERT INTO smu.Transazione(CRO, Importo, Data, Ora, Causale, Tipo, Mittente,
                Destinatario, NumeroCarta, NomeCategoria)
VALUES(12345678910, 20.00, '2024-03-19', '09:45:00', 'Acquisto online',
        'Entrata', NULL,'E-commerce', '1234567890123456', NULL);

INSERT INTO smu.Transazione(CRO, Importo, Data, Ora, Causale, Tipo, Mittente,
                Destinatario, NumeroCarta, NomeCategoria)
VALUES(46173636910, 50.00, '2024-01-29', '13:35:00', 'Acquisto online',
        'Uscita', 'Amazon', NULL, '1234567890123456', NULL);
```

6.3.3 Trigger tipoCarta

Trigger che prima dell'inserimento di una carta controlli che: se il tipo è 'Debito' allora plafond è NULL, se il tipo è 'Credito' allora Plafond è NOT NULL.

```
CREATE OR REPLACE FUNCTION smu.triggerTipoCarta() RETURNS TRIGGER AS
$$
    BEGIN
        IF NEW.TipoCarta = 'Debito' AND NEW.Plafond IS NOT NULL THEN
            RAISE EXCEPTION 'ERRORE: Plafond non nullo in una carta di debito';
        END IF;

        IF NEW.TipoCarta = 'Credito' AND NEW.Plafond IS NULL THEN
            RAISE EXCEPTION 'ERRORE: Plafond nullo in una carta di credito';
        END IF;

        RETURN NEW;

    END;
$$LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ControlloTipoCarta
    BEFORE INSERT ON smu.Carta
    FOR EACH ROW EXECUTE FUNCTION smu.triggerTipoCarta();
```

Insert per testare il trigger:

```
INSERT INTO smu.CARTA(NumeroCarta, Nome, CVV, Scadenza, Saldo, TipoCarta, Plafond, NumeroConto)
VALUES('5355284922617884', 'Poste Pay Evolution', 100, '2025-12-31', 13.00, 'Credito', NULL, 1);

INSERT INTO smu.CARTA(NumeroCarta, Nome, CVV, Scadenza, Saldo, TipoCarta, Plafond, NumeroConto)
VALUES('5334628274884783', 'Carta prepagata', 345, '2024-08-31', 500.00, 'Debito', 1000.00, 1);
```

6.3.4 Trigger Categorizza transazione

Dopo l'inserimento di una nuova transazione il trigger si occupa di associare quest'ultima ad una categoria, andando a confrontare le parole chiavi di ogni categoria con la causale della transazione esaminata.

```
CREATE OR REPLACE FUNCTION smu.triggerCategorizzaTransazione() RETURNS TRIGGER AS
$$
DECLARE
    --variabile per iterare sul nome categoria
    nome_categoria smu.Categoria.NomeCategoria%TYPE;
    -- Array di testo per memorizzare le parole chiave di una categoria
    ArrayParoleChiavi TEXT[];
    -- Variabile per memorizzare temporaneamente ogni parola chiave
    parola TEXT;
    -- Variabile booleana per indicare se è stata trovata una corrispondenza
    matched BOOLEAN := FALSE;
BEGIN

    FOR nome_categoria IN
        SELECT NomeCategoria
        FROM smu.Categoria
    LOOP
        --la funzione string_to_array mi crea un'array di valori a partire da una stringa
            di parole separate da virgole
        --successivamente memorizzo tale array in ArrayParoleChiavi
        SELECT string_to_array(ParoleChiavi, ',') INTO ArrayParoleChiavi
            FROM smu.Categoria
            WHERE NomeCategoria = nome_categoria;

        --memorizzo nella variabile parola l'elemento dell'array ParoleChiave
        FOREACH parola IN ARRAY ArrayParoleChiavi
        LOOP
            -- Verifica se la causale contiene la parola chiave
            IF NEW.Causale ILIKE '%' || parola || '%' THEN
                NEW.NomeCategoria := nome_categoria;
                matched := TRUE;
                EXIT;  -- Esce dal loop delle parole chiave una volta trovata una corrispondenza
            END IF;
        END LOOP;

        -- Se è stata trovata una corrispondenza interrompe il ciclo delle categorie
        IF matched THEN
            EXIT;
        END IF;

    END LOOP;

    -- Se nessuna categoria è stata trovata, assegno la transazione alla categoria "Altro"
    IF NOT matched THEN
       NEW.NomeCategoria := 'Altro';
    END IF;

    --aggiorno il nomecategoria della transazione
    UPDATE smu.Transazione
    SET NomeCategoria = NEW.NomeCategoria
```

```
    WHERE IdTransazione = NEW.IdTransazione;

    RETURN NEW;  -- Restituisce la nuova riga con il campo NomeCategoria aggiornato
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER InserisciCategoriaInTransazione
    AFTER INSERT ON smu.Transazione
    FOR EACH ROW EXECUTE FUNCTION smu.triggerCategorizzaTransazione();
```

6.3.5 Trigger Inizializza Saldo Portafoglio

Trigger che, dopo l'inserimento di un nuovo portafoglio, inizializza il saldo del portafoglio a 0.

```
CREATE OR REPLACE FUNCTION smu.triggerInizializzaSaldoPortafoglio() RETURNS TRIGGER AS
$$
BEGIN
    UPDATE smu.Portafoglio
    SET Saldo = 0
    WHERE IdPortafoglio = NEW.IdPortafoglio;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER InizializzaSaldoPortafoglio
    AFTER INSERT
    ON smu.Portafoglio
    FOR EACH ROW
EXECUTE FUNCTION smu.triggerInizializzaSaldoPortafoglio();
```

6.3.6 Trigger Aggiorna Saldo Portafoglio

```
CREATE OR REPLACE FUNCTION smu.triggerAggiornaSaldoPortafoglio() RETURNS TRIGGER AS
$$
BEGIN
UPDATE smu.Portafoglio
    SET Saldo = Saldo + (SELECT T.Importo
                         FROM (smu.Transazione AS T JOIN smu.TransazioniInPortafogli
                         AS TP on T.IdTransazione = TP.IdTransazione)
                             JOIN smu.Portafoglio AS P on TP.IdPortafoglio = P.IdPortafoglio
                         WHERE T.IdTransazione = NEW.IdTransazione)
    WHERE IdPortafoglio = NEW.IdPortafoglio;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ModificaSaldoPortafoglio
    AFTER INSERT
    ON smu.TransazioniInPortafogli
    FOR EACH ROW
EXECUTE FUNCTION smu.triggerAggiornaSaldoPortafoglio();
```

6.4 Definizione delle funzioni

6.4.1 Procedura SpesaProgrammata

Procedura che gestisce le spese programmate: trasforma le spese programmate in transazioni in uscita quando la data di scadenza è uguale alla data attuale, ed aggiorna la data di scadenza in base alla periodicità per il prossimo rinnovo; se la data di fine rinnovo è uguale alla data attuale, allora elimina la spesa programmata.

```
CREATE OR REPLACE PROCEDURE smu.ProceduraSpesaProgrammata() AS
$$
    DECLARE
    destinatarioS smu.speseprogrammate.destinatario%TYPE;
    descrizioneS smu.speseprogrammate.descrizione%TYPE;
    numerocartaS smu.speseprogrammate.numerocarta%TYPE;
    intervalloS smu.speseprogrammate.periodicita%TYPE;
    IdSpesaS smu.speseprogrammate.idspesa%TYPE;
    FineRinnovo smu.SpeseProgrammate.dataFineRinnovo%TYPE;
    importoS FLOAT := 0;
    cursore REFCURSOR;

    BEGIN

        OPEN cursore FOR (SELECT S.Importo, S.Descrizione, s.Destinatario, S.NumeroCarta,
                          S.Periodicita, S.IdSpesa, S.DataFineRinnovo
                         FROM smu.SpeseProgrammate AS S
                         WHERE S.DataScadenza = CURRENT_DATE);
        LOOP
            FETCH cursore INTO importoS, descrizioneS, destinatarioS, numerocartaS,
                          intervalloS, IdSpesaS, FineRinnovo;
            EXIT WHEN NOT FOUND;

            -- Inserimento della transazione
            INSERT INTO smu.Transazione(Importo, Data, Ora, Causale, Tipo,
                                Mittente, Destinatario, NumeroCarta)
            VALUES (importoS, CURRENT_DATE, CURRENT_TIME, descrizioneS,
                                'Uscita', NULL, destinatarioS, numerocartaS);

            -- Se la data di fine rinnovo è uguale alla CURRENT_DATE,
                allora elimino la spesa programmata.
            IF FineRinnovo = CURRENT_DATE THEN
                DELETE FROM smu.SpeseProgrammate
                WHERE IdSpesa = IdSpesaS;
            END IF;

            --Aggiornamento della data di scadenza del prossimo pagamento programmato
            UPDATE smu.SpeseProgrammate
            SET DataScadenza = DataScadenza + (CASE
                           WHEN intervalloS = '7 giorni' THEN INTERVAL '7 days'
                           WHEN intervalloS = '15 giorni' THEN INTERVAL '15 days'
                           WHEN intervalloS = '1 mese' THEN INTERVAL '1 month'
                           WHEN intervalloS = '3 mesi' THEN INTERVAL '3 months'
                           WHEN intervalloS = '6 mesi' THEN INTERVAL '6 months'
                           WHEN intervalloS = '1 anno' THEN INTERVAL '1 year'
            END)
            WHERE IdSpesa = IdSpesaS;
```

```
        END LOOP;
        CLOSE cursore;
    END;
$$ LANGUAGE plpgsql;

CALL smu.SpesaProgrammata();
```