UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Dipartimento di Ingegneria
Corso di Laurea Triennale in Informatica

Progettazione e sviluppo della base di dati SavingMoneyUnina

Docente:
Prof. Mara Sangiovanni

Autori:
Francesco Donnarumma
N86004658
Arturo Donnarumma
N86004837

Anno Accademico 2023/2024

Indice

Capitolo 1

Introduzione

Benvenuti nella documentazione dettagliata relativa alla struttura del database di SavingMoneyUnina. Questo documento fornisce una panoramica completa degli elementi chiave che costituiscono la base di dati, offrendo informazioni essenziali sulla progettazione e organizzazione necessarie per una gestione efficiente delle transazioni finanziarie.

Il database di SavingMoneyUnina è stato progettato per facilitare la registrazione, il recupero e l'analisi efficiente delle informazioni finanziarie personali e familiari. Attraverso una struttura intuitiva, consentiamo agli utenti di tracciare e gestire le transazioni provenienti da diverse fonti finanziarie.

La documentazione dettaglierà le tabelle principali, le relazioni chiave e gli schemi di collegamento tra i dati, fornendo una visione chiara sulla gestione automatica e manuale delle transazioni.

Questa guida è essenziale per coloro che necessitano di una visione approfondita sulla progettazione del database, utile sia nello sviluppo che nella manutenzione del sistema nell'ecosistema finanziario.

Capitolo 2

Progettazione Concettuale

2.1 Diagramma Delle Classi UML

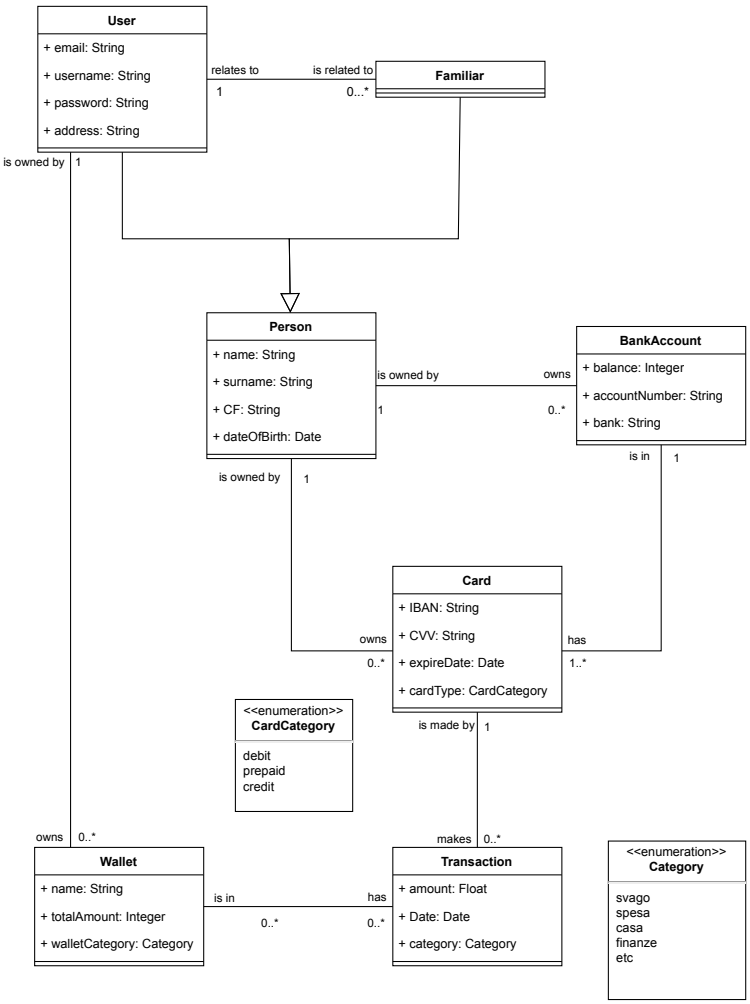

Figura 2.1: Diagramma UML

2.2 Diagramma ER (Entità Relazione)

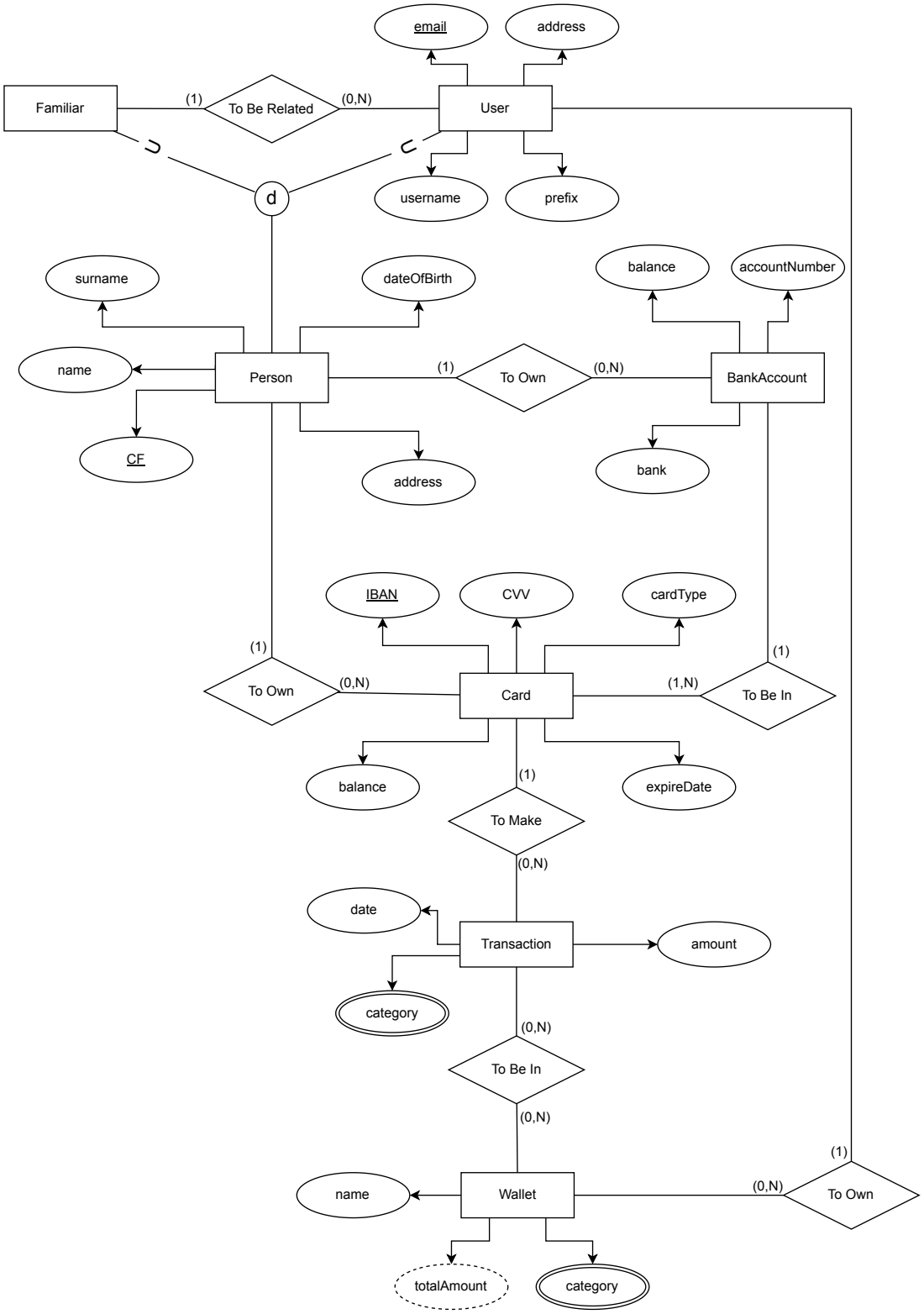

Figura 2.2: Diagramma ER

2.3 Ristrutturazione

2.3.1 Attributi multipli

Per quanto riguarda la gestione di attributi multipli, abbiamo deciso di gestire l'attributo *category* della tabella **Transaction**, originariamente definito come enumerazione, trasformandolo in una stringa, poiché non abbiamo bisogno di valori specifici, trattandosi di una categoria personalizzabile.

Anche per l'attributo *cardType* è stata applicata la stessa procedura. Il controllo dell'attributo verrà gestito tramite i vincoli approfonditi nel dizionario dei vincoli.

2.3.2 Generalizzazioni

Per la generalizzazione, essendo di tipologia totale e disgiunta, abbiamo optato per il metodo di eliminare la classe generale. Abbiamo trasferito tutti gli attributi di essa nelle classi specializzate, conservando le relative relazioni.

2.3.3 Analisi degli identificativi

Per la maggior parte delle classi, saranno utilizzati come identificativi attributi già presenti di natura nelle classi stesse, poiché risultano sufficienti e non richiedono l'uso di una chiave surrogata. Tuttavia, in alcune classi, sono presenti chiavi surrogate, identificate con il prefisso **ID_**.

2.3.4 Diagramma UML ristrutturato

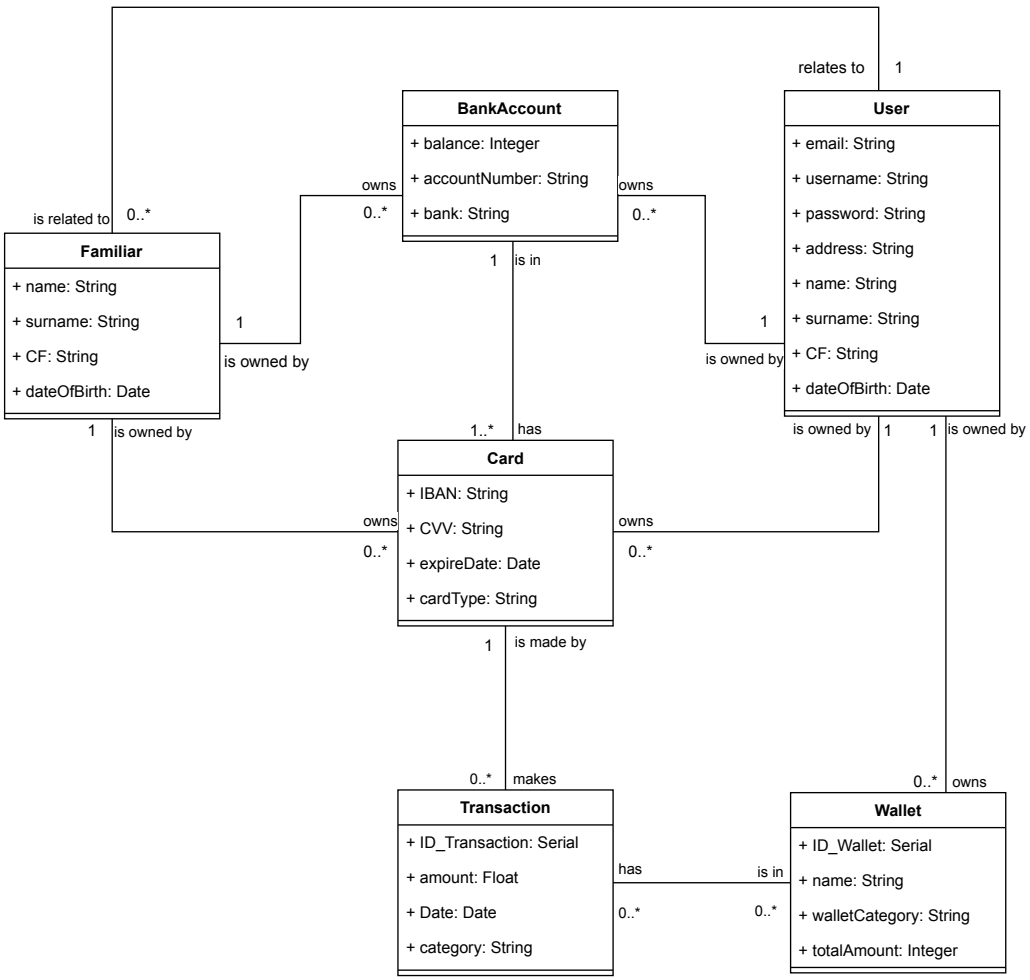

Figura 2.3: Diagramma UML Ristrutturato

2.4 Dizionari

2.4.1 Dizionario delle classi

Classe	Descrizione	Attributi
User	Classe utilizzata per identificare gli utenti registrati alla piattaforma.	**email** (*String*): chiave primaria, email con la quale l'utente si è registrato. **username** (*String*): nome che viene mostrato per riconoscere lo stesso. **password** (*String*): stringa atta alla convalidazione durante l'accesso all'account. **address** (*String*): indirizzo del domicilio. **name** (*String*): nome. **surname** (*String*): cognome. **CF** (*String*): codice fiscale. **dateOfBirth** (*Date*): data di nascita.
Familiar	Classe utilizzata per identificare i familiari degli utenti presenti nel database.	**name** (*String*): nome. **surname** (*String*): cognome. **CF** (*String*): codice fiscale, chiave primaria nel caso del familiare. **dateOfBirth** (*Date*): data di nascita.
BankAccount	Classe utilizzata per identificare i conti correnti appartenenti a utenti o familiari.	**balance** (*Integer*): indica il saldo disponibile sul conto corrente. **accountNumber** (*String*): chiave primaria, identificativa del conto corrente. **bank** (*String*): nome della banca alla quale è associato il conto corrente.
Card	Classe utilizzata per identificare le carte appartenenti a utenti o familiari.	**IBAN** (*String*): codice identificativo della carta. **CVV** (*String*): codice di sicurezza per le transazioni delle carte. **expireDate** (*Date*): data che indica la scadenza della carta. **cardType** (*String*): campo che indica la tipologia della carta.
Transaction	Classe utilizzata per tenere traccia di tutte le transazioni effettuate.	**ID_Transaction** (*Serial*): chiave surrogata, identificativo della singola transazione. **amount** (*Float*): indica l'ammontare della transazione. **date** (*Date*): data in cui è avvenuta la transazione. **category** (*String*): tipologia di transazione. Serve per l'associazione automatica ai portafogli.

Classe	Descrizione	Attributi
Wallet	Classe utilizzata per raggruppare transazioni.	**ID_Wallet** (*Serial*): chiave surrogata, identificativo del singolo protafoglio. **name** (*String*): nome del portafoglio. **walletCategory** (*String*): categoria del portafoglio. **totalAmount** (*Float*): indica la somma di tutte le transazioni relative al portafoglio.

2.4.2 Dizionario delle associazioni

Associazione	Descrizione	Classi coinvolte
To Be Related	Esprime la parentela tra gli utenti e i familiari	**Familiar** [1] (**relates to**): indica, per ogni familiare, con quale utente è imparentato. **User** [**0..***] (**is related to**): indica quali sono i familiari che sono imparentati con esso.
To Own	Esprime il possesso degli utenti sui conti correnti	**User** [**0..***] (**owns**): indica, per ogni utente, quali sono i conti correnti che possiede. **BankAccount** [1] (**is owned by**): indica l'utente che possiede il conto corrente in questione.
To Own	Esprime il possesso dei familiari sui conti correnti	**Familiar** [**0..***] (**owns**): indica, per ogni familiare, quali sono i conti correnti che possiede. **BankAccount** [1] (**is owned by**): indica il familiare che possiede il conto corrente in questione.
To Be In	Esprime la correlazione tra le carte e i conti correnti	**Card** [1] (**is in**): indica, per ogni carta, qual è il conto corrente a cui sono associate. **BankAccount** [**1..***] (**has**): indica quali sono le carte che sono associate al conto corrente in questione.
To Own	Esprime il possesso degli utenti sulle carte	**User** [**0..***] (**owns**): indica, per ogni utente, quali sono le carte che possiede. **Card** [1] (**is owned by**): indica l'utente che possiede la carta in questione.
To Own	Esprime il possesso dei familiari sulle carte	**Familiar** [**0..***] (**owns**): indica, per ogni utente, quali sono le carte che possiede. **Card** [1] (**is owned by**): indica l'utente che possiede la carta in questione.

Associazione	Descrizione	Classi coinvolte
To Make	Esprime il collegamento una la transazione e la carta con la quale è stata effettuata	**Card** [**0..***] (**makes**): indica, per ogni carta, tutte le transazioni effettuate. **Transaction** [**1**] (**is made by**): indica con quale carta è stata effettuata la transazione in questione.
To Be In	Esprime la correlazione tra le transazioni e i portafogli	**Wallet** [**0..***] (**has**): indica, per ogni portafoglio, quali sono le transazioni che lo compongono. **Transaction** [**0..***] (**is in**): indica qual è il portafoglio a cui fa riferimento la transazione in questione.

2.4.3 Dizionario dei vincoli

Vincolo	Tipo	Descrizione
unique_username	Intrarelazionale	Nella tabella User non ci possono essere t-uple diverse con lo stesso username.
unique_CF	Intrarelazionale	Nella tabella User non ci possono essere t-uple diverse con lo stesso CF.
check_BirthDate_User	Dominio	Nella tabella User la data deve essere necessariamente antecedente alla data odierna.
check_BirthDate_Familiar	Dominio	Nella tabella Familiar la data deve essere necessariamente antecedente alla data odierna.
ownership_check_BA	N-upla	Per ogni t-upla della tabella BankAccount, essa deve essere associata necessariamente o ad un Utente o ad un Familiare, ma non ad entrambi.
ownership_check_Card	N-upla	Per ogni t-upla della tabella Card, essa deve essere associata necessariamente o ad un Utente o ad un Familiare, ma non ad entrambi.
cardType_Check	Dominio	Nella tabella Card, per ogni t-upla, il campo cardCategory deve essere necessariamente “prepaid”, “debit” o “credit”.
check_Transaction_Date	Dominio	Nella tabella Transaction, per ogni t-upla, la data deve essere necessariamente antecedente o coincidente alla data odierna.

Vincolo	Tipo	Descrizione
check_Card_Owner	Interrelazionale	Quando viene inserita una carta, il proprietario di essa deve essere anche il proprietario del conto corrente al quale viene associata la carta, o al massimo un suo familiare, in entrambe le direzioni
check_Expire_Date	Interrelazionale	Quando viene inserita una transazione, la carta con la quale è stata effettuata la transazione deve risultare valida al momento della transazione stessa.

Capitolo 3

Progettazione Logica

3.1 Schema Logico

3.1.1 Traduzione delle classi e delle associazioni

User (email, username, password, address, name, surname, CF, dateOfBirth)

Familiar (name, surname, CF, dateOfBirth, familiarEmail)
Chiavi esterne: familiarEmail → User.email

BankAccount (balance, accountNumber, bank, ownerCF, ownerEmail)
Chiavi esterne: ownerCF → Familiar.CF
ownerEmail → User.email

Card (IBAN, CVV, expireDate, cardCategory, BA_Number, ownerCF, ownerEmail)
Chiavi esterne: BA_Number → BankAccount.accountNumber
ownerCF → Familiar.CF
ownerEmail → User.email

Wallet (ID_Wallet, name, WalletCategory, totalAmount, ownerEmail)
Chiavi esterne: ownerEmail → User.email

Transaction (ID_Transaction, amount, date, category, CardIBAN)
Chiavi esterne: cardIBAN → Card.IBAN

TransactionInWallet (ID_Transaction, ID_Wallet)
Chiavi esterne: ID_Transaction → Transaction.ID_Transaction
ID_Wallet → Wallet.ID_Wallet

3.1.2 Modalità di traduzione delle associazioni

Associazione	Implementazione
To Be Related	Relazione 0..* a 1, è stata migrata la chiave primaria di *User* (username) in *Familiar* (familiarUsername)
To Own	Tutte le relazioni di questo genere, tra *Familiar*, *User*, *BankAccount*, *Card* e *Wallet*, sono di tipologia 0..* a 1, di conseguenza sono state gestite tutte allo stesso modo. Ovvero migrando la chiave primaria dell'entità debole, nell'entità forte. Per controllare nel dettaglio le chiavi esterne in ognuna delle tabelle indicate vedere la tabella Traduzione delle classi.
To Be In (Card/BankAccount)	Relazione 1..* a 1, è stata migrata la chiave primaria di *BankAccount* (accountNumber) in *Card* (BA_Number)
To Make	Relazione 0..* a 1, è stata migrata la chiave primaria di *Card* (IBAN) in *Transaction* (CardIBAN)
To Be In (Transaction/Wallet)	Relazione 0..* a 0..*, è stata creata la tabella ponte *TransactionInWallet* che contiene le chiavi primarie di *Transaction* (ID_Transaction) e di *Wallet* (ID_Wallet)

Capitolo 4

Schema Fisico

4.1 Definizoni SQL delle tabelle

4.1.1 User

```
CREATE TABLE smu."user"[1] (
    email character varying(100) NOT NULL,
    username character varying(20) NOT NULL,
    password character varying(30) NOT NULL,
    address character varying(100),
    name character varying(20) NOT NULL,
    surname character varying(30) NOT NULL,
    cf character varying(16) NOT NULL,
    dateofbirth date NOT NULL,
    CONSTRAINT check_birthdate_user CHECK ((dateofbirth <
        CURRENT_DATE))
);
```

4.1.2 Familiar

```
CREATE TABLE smu.familiar (
    name character varying(20) NOT NULL,
    surname character varying(30) NOT NULL,
    cf character varying(16) NOT NULL,
    dateofbirth date NOT NULL,
    familiaremail character varying(100) NOT NULL,
    CONSTRAINT check_birthdate_familiar CHECK ((
        dateofbirth < CURRENT_DATE))
);
```

[1] vengono usate le virgolette perché altrimenti la parola *user* verrebbe identificata come keyword

4.1.3 BankAccount

```
CREATE TABLE smu.bankaccount (
    balance integer NOT NULL,
    accountnumber integer NOT NULL,
    bank character varying(40),
    ownercf character varying(16),
    owneremail character varying(100),
    CONSTRAINT ownership_check_ba CHECK (((ownercf IS
        NULL) <> (owneremail IS NULL)))
);
```

4.1.4 Card

```
CREATE TABLE smu.card (
    iban character varying(27) NOT NULL,
    cvv character varying(3) NOT NULL,
    expiredata date NOT NULL,
    cardtype character varying(11),
    ba_number integer NOT NULL,
    ownercf character varying(16),
    owneremail character varying(100),
    CONSTRAINT cardtype_check CHECK (((cardtype)::text =
        ANY ((ARRAY['prepaid'::character varying, 'debit'
        ::character varying, 'credit'::character varying
        ])::text[]))),
    CONSTRAINT ownership_check_card CHECK (((ownercf IS
        NULL) <> (owneremail IS NULL)))
);
```

4.1.5 Transaction

```
CREATE TABLE smu.transaction (
    id_transaction integer NOT NULL,
    amount double precision NOT NULL,
    date date NOT NULL,
    category character varying(35),
    cardiban character varying(27) NOT NULL,
    CONSTRAINT check_transaction_date CHECK ((date <=
        CURRENT_DATE))
);
```

4.1.6 Wallet

```
CREATE TABLE smu.wallet (
    id_wallet integer NOT NULL,
    name character varying(35) NOT NULL,
    walletcategory character varying(35) NOT NULL,
    totalamount double precision NOT NULL,
    owneremail character varying(100) NOT NULL
);
```

4.1.7 TransactionInWallet

```
CREATE TABLE smu.transactioninwallet (
    id_transaction integer NOT NULL,
    id_wallet integer NOT NULL
);
```

4.2 Definizoni SQL dei trigger

4.2.1 check_card_owner_trigger

```
CREATE TRIGGER check_card_owner_trigger
    BEFORE INSERT ON smu.card
    FOR EACH ROW
    EXECUTE FUNCTION smu.check_card_owner();

CREATE FUNCTION smu.check_card_owner() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
DECLARE
    BA_used smu.bankaccount%ROWTYPE;
    familiar_email smu.familiar.familiaremail%TYPE;
BEGIN

    familiar_email := NULL;

    -- Recupera le informazioni relative al conto
        corrente
    -- al quale si sta associando la carta
    SELECT *
    INTO BA_used
    FROM smu.bankaccount
    WHERE accountnumber = NEW.ba_number;

    -- Recupera l'email dell'account al quale e'
        associato
    -- il familiare, proprietario della carta
    IF NEW.ownercf IS NOT NULL THEN
        SELECT familiaremail
        INTO familiar_email
        FROM smu.familiar
        WHERE cf = NEW.ownercf;
    END IF;

    -- Se tutte le condizioni sono vere, viene inserita
        la carta, altrimenti viene restituita una
        exception
    IF (BA_used.owneremail = NEW.owneremail OR BA_used.
        ownercf = NEW.ownercf) OR
    (familiar_email = BA_used.owneremail) OR BA_used.
        ownercf IN (SELECT CF FROM smu.familiar WHERE
        familiaremail = NEW.owneremail)
```

```
    THEN
        -- L'IF e' stato strutturato in questo modo
            perche' non basta negare le condizioni per
            avere
        -- solo un IF THEN. Questo perche' nel caso in
            cui alcuni attributi sono NULL il sistema
        -- non e' in grado di fornire una valutazione
            sulla condizione, quindi gli AND da
            sostituire
        -- agli attuali OR sarebbero sempre falsi.
    ELSE
        RAISE EXCEPTION 'Il proprietario della carta deve
            essere anche il proprietario del conto
            corrente, o al massimo un suo familiare.';
    END IF;

    -- Nell'ultima porzione della condizione dell'IF
        viene controllato se il codice fiscale
    -- del proprietario del conto corrente e' presente
        nell'elenco dei familiari associati
    -- all'email del proprietario della carta

    RETURN NEW;
END;
$$;
```

4.2.2 connect_transaction_to_wallet_trigger

```
CREATE TRIGGER connect_transaction_to_wallet_trigger
    AFTER INSERT OR UPDATE ON smu.transaction
    FOR EACH ROW
    EXECUTE FUNCTION smu.connect_transaction_to_wallet();

CREATE FUNCTION smu.connect_transaction_to_wallet()
    RETURNS trigger
    LANGUAGE plpgsql
    AS $$
DECLARE
    wallet_row smu.wallet%ROWTYPE;
    card_row smu.card%ROWTYPE;
    old_card_row smu.card%ROWTYPE;
    account_email smu.user.email%TYPE;
    ba_row smu.bankaccount%ROWTYPE;
    old_ba_row smu.bankaccount%ROWTYPE;
BEGIN

    -- Seleziono la carta con la quale e' stata
        effettuata la transazione
    SELECT *
    INTO card_row
    FROM smu.card
    WHERE iban = NEW.cardiban;
```

```
-- Controlla se la carta e' scaduta o meno al momento
     della transazione
IF smu.expired_card(card_row.expiredata, NEW.date)
    THEN
    RAISE EXCEPTION 'La carta risultava scaduta al
        momento della transazione';
END IF;

-- Recupero il conto corrente al quale e' associato
    la carta
SELECT *
INTO ba_row
FROM smu.bankaccount
WHERE accountnumber = card_row.ba_number;

-- Controllo se la transazione puo' essere effettuata
     o meno
IF TG_OP = 'INSERT' AND ba_row.balance < NEW.amount
    THEN
    RAISE EXCEPTION 'Saldo sul conto corrente
        insufficiente';
ELSIF TG_OP = 'UPDATE' AND (ba_row.balance + OLD.
    amount) < NEW.amount THEN
    RAISE EXCEPTION 'Saldo sul conto corrente
        insufficiente al momento della transazione';
END IF;

-- Recupero l'email dell'account al quale saranno
    associati i portafogli
IF card_row.owneremail IS NOT NULL THEN
    -- Se la carta appartiene ad un utente, mi salvo
        l'email nella variabile account_email
    account_email := card_row.owneremail;
ELSE
    -- Altrimenti, appartiene sicuramente ad un
        familiare e vado a
    -- recuperare l'email dell'utente al quale e'
        associato
    SELECT familiaremail
    INTO account_email
    FROM smu.familiar
    WHERE cf = card_row.ownercf;
END IF;

IF TG_OP = 'INSERT' THEN
    -- Trova i wallet con la stessa categoria della
        transazione
    -- appena inserita che appartengono all'utente
        corretto
    FOR wallet_row IN
        SELECT *
```

```
            FROM smu.wallet
            WHERE walletcategory = NEW.category AND
                owneremail = account_email
        LOOP

            -- Collega la transazione al wallet trovato
            INSERT INTO smu.transactioninwallet (
                id_transaction, id_wallet)
            VALUES (NEW.id_transaction, wallet_row.
                id_wallet);

            -- Aggiorna il campo totalamount del wallet
            UPDATE smu.wallet
            SET totalamount = totalamount + NEW.amount
            WHERE id_wallet = wallet_row.id_wallet;

        END LOOP;

        -- Aggiorna il saldo del conto corrente
        UPDATE smu.bankaccount
        SET balance = balance - NEW.amount
        WHERE accountnumber = ba_row.accountnumber;
    END IF;

    IF TG_OP = 'UPDATE' THEN

        -- Se e' stata modificata la categoria, la
            transazione viene collegata ai nuovi
            portafogli
        -- altrimenti viene ricollegata agli stessi

        -- Scollego la transazione da tutti i portafogli
        DELETE FROM smu.transactioninwallet WHERE
            id_transaction = OLD.id_transaction;

        -- Seleziona tutti i portafogli a cui era
            collegata la transazione
        FOR wallet_row IN
            SELECT *
            FROM smu.wallet
            WHERE walletcategory = OLD.category AND
                owneremail = account_email
        LOOP

            -- Aggiorna il campo totalamount del wallet
            UPDATE smu.wallet
            SET totalamount = totalamount - OLD.amount
            WHERE id_wallet = wallet_row.id_wallet;

        END LOOP;

        -- Seleziona tutti i portafogli a cui deve essere
```

```
            collegata la transazione
        FOR wallet_row IN
            SELECT *
            FROM smu.wallet
            WHERE walletcategory = NEW.category AND
                owneremail = account_email
        LOOP
            -- Collega la transazione al wallet trovato
            INSERT INTO smu.transactioninwallet (
                id_transaction, id_wallet)
            VALUES (NEW.id_transaction, wallet_row.
                id_wallet);

            -- Aggiorna il campo totalamount del wallet
            UPDATE smu.wallet
            SET totalamount = totalamount + NEW.amount
            WHERE id_wallet = wallet_row.id_wallet;
        END LOOP;

        -- Seleziono la carta con la quale e' stata
            effettuata inizialmente la transazione
        SELECT *
        INTO old_card_row
        FROM smu.card
        WHERE iban = OLD.cardiban;

        -- Recupero il conto corrente al quale e'
            associato la vecchia carta
        SELECT *
        INTO old_ba_row
        FROM smu.bankaccount
        WHERE accountnumber = old_card_row.ba_number;

        -- Aggiorno il saldo del vecchio conto corrente
        UPDATE smu.bankaccount
        SET balance = balance + OLD.amount
        WHERE accountnumber = old_ba_row.accountnumber;

        -- Aggiorno il saldo del nuovo conto corrente
        UPDATE smu.bankaccount
        SET balance = balance - NEW.amount
        WHERE accountnumber = ba_row.accountnumber;

    END IF;

    RETURN NEW;
END;
$$;
```

4.2.3 update_wallet_category_trigger

```
CREATE TRIGGER update_wallet_category_trigger
    AFTER UPDATE OF walletcategory ON smu.wallet
    FOR EACH ROW
    EXECUTE FUNCTION smu.update_wallet_category();

CREATE FUNCTION smu.update_wallet_category() RETURNS
    trigger
    LANGUAGE plpgsql
    AS $$
DECLARE
    transaction_row smu.transaction%ROWTYPE;
BEGIN

    -- Vengono scollegate tutte le transazioni dal wallet
    DELETE FROM smu.transactioninwallet WHERE id_wallet =
         OLD.id_wallet;

    -- Reimposta a 0 la somma degli importi del
        portafoglio
    UPDATE smu.wallet
    SET totalamount = 0
    WHERE id_wallet = NEW.id_wallet;

    FOR transaction_row IN
        SELECT *
        FROM smu.transaction AS T
        WHERE T.category = NEW.walletcategory AND T.
            cardiban IN
        (SELECT iban FROM smu.card
        WHERE owneremail = NEW.owneremail UNION
        SELECT C.iban FROM smu.familiar AS F JOIN
        smu.card AS C ON F.cf = C.ownercf
        WHERE familiaremail = NEW.owneremail)
    LOOP
        -- Collega la transazione trovata
        INSERT INTO smu.transactioninwallet (
            id_transaction, id_wallet)
        VALUES (transaction_row.id_transaction, NEW.
            id_wallet);

        -- Aggiorna il campo totalamount del wallet
        UPDATE smu.wallet
        SET totalamount = totalamount + transaction_row.
            amount
        WHERE id_wallet = NEW.id_wallet;
    END LOOP;

    RETURN NEW;

END;
$$;
```

4.3 Definizoni SQL delle funzioni

4.3.1 expired_card

```
CREATE FUNCTION smu.expired_card(card_expire_date date,
    transaction_date date) RETURNS boolean
    LANGUAGE plpgsql
    AS $$
BEGIN
    -- Questa funzione controlla se la carta data in
        input era scaduta
    -- al momento della transazione data in input. La
        funzione viene usata
    -- all'interno della funzione "
        connect_transaction_to_wallet" la quale viene
    -- eseguita ad ogni inserimento di una nuova
        transazione.
    -- Questa funzione viene utilizzata per garantire il
        vincolo "check_expire_date".
    IF card_expire_date < transaction_date THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
$$;
```

Capitolo 5

Conclusione

In conclusione, la documentazione dettagliata sul database di SavingMoneyUnina fornisce una guida esaustiva e indispensabile per comprendere appieno la struttura e il funzionamento del sistema.

Grazie alla sua chiarezza e completezza, questa guida supporta gli utenti nello sfruttare appieno le potenzialità del database per registrare, gestire e analizzare le proprie transazioni finanziarie in modo efficiente e accurato.

Con una progettazione intuitiva e relazioni ben definite, il database si pone come uno strumento fondamentale per una gestione finanziaria personale e familiare efficace, sia nel presente che nel futuro.