**Relazione Progetto di Basi di Dati**

**Cash Flow Web**

**Giovanni Maria Savoca – 970094**

[giovannimaria.savoca@studio.unibo.it](mailto:giovannimaria.savoca@studio.unibo.it)

**Introduzione**

Il progetto "Cash Flow Web" è un'applicazione web progettata per aiutare gli utenti a gestire le proprie finanze personali in modo efficiente e organizzato. L'obiettivo principale del progetto è fornire una piattaforma user-friendly che consenta agli utenti di monitorare e controllare il proprio flusso di cassa, facilitando la registrazione delle transazioni, la gestione dei conti bancari e il rispetto dei budget di spesa.

Obiettivi del Progetto:

Gestione Completa delle Transazioni: Gli utenti possono registrare tutte le loro transazioni finanziarie, sia in entrata che in uscita, categorizzandole in modo da avere una visione chiara delle proprie spese e entrate.

Monitoraggio dei Conti: L'applicazione permette la creazione e la gestione di più conti bancari, consentendo agli utenti di suddividere le transazioni tra diversi conti per una gestione più precisa.

Impostazione di Budget: Gli utenti possono impostare budget massimi per diverse categorie di spesa, aiutandoli a controllare le loro finanze e prevenire spese eccessive.

Pianificazione dei Risparmi: Gli utenti possono creare obiettivi di risparmio e pianificare risparmi programmati, automatizzando il processo di allocazione dei risparmi giornalieri.

Gestione di Debiti e Crediti: L'applicazione supporta la registrazione di debiti e crediti, creando automaticamente le transazioni associate per una gestione accurata del credito e del debito.

Generazione di Report Finanziari: L'applicazione fornisce strumenti per la generazione di report finanziari dettagliati, permettendo agli utenti di analizzare le proprie finanze attraverso grafici e tabelle.

Tecnologie Utilizzate:

Il progetto è sviluppato utilizzando un stack tecnologico basato su AMP (Apache, MySQL, PHP) per il backend, con un sistema di log degli eventi implementato su MongoDB, un DBMS NoSQL. Il design del database relazionale segue le metodologie di progettazione apprese durante il corso, garantendo un'architettura robusta e scalabile.

**Specifiche sui Dati**

Elenco e descrizione dei dati gestiti dal sistema, come:

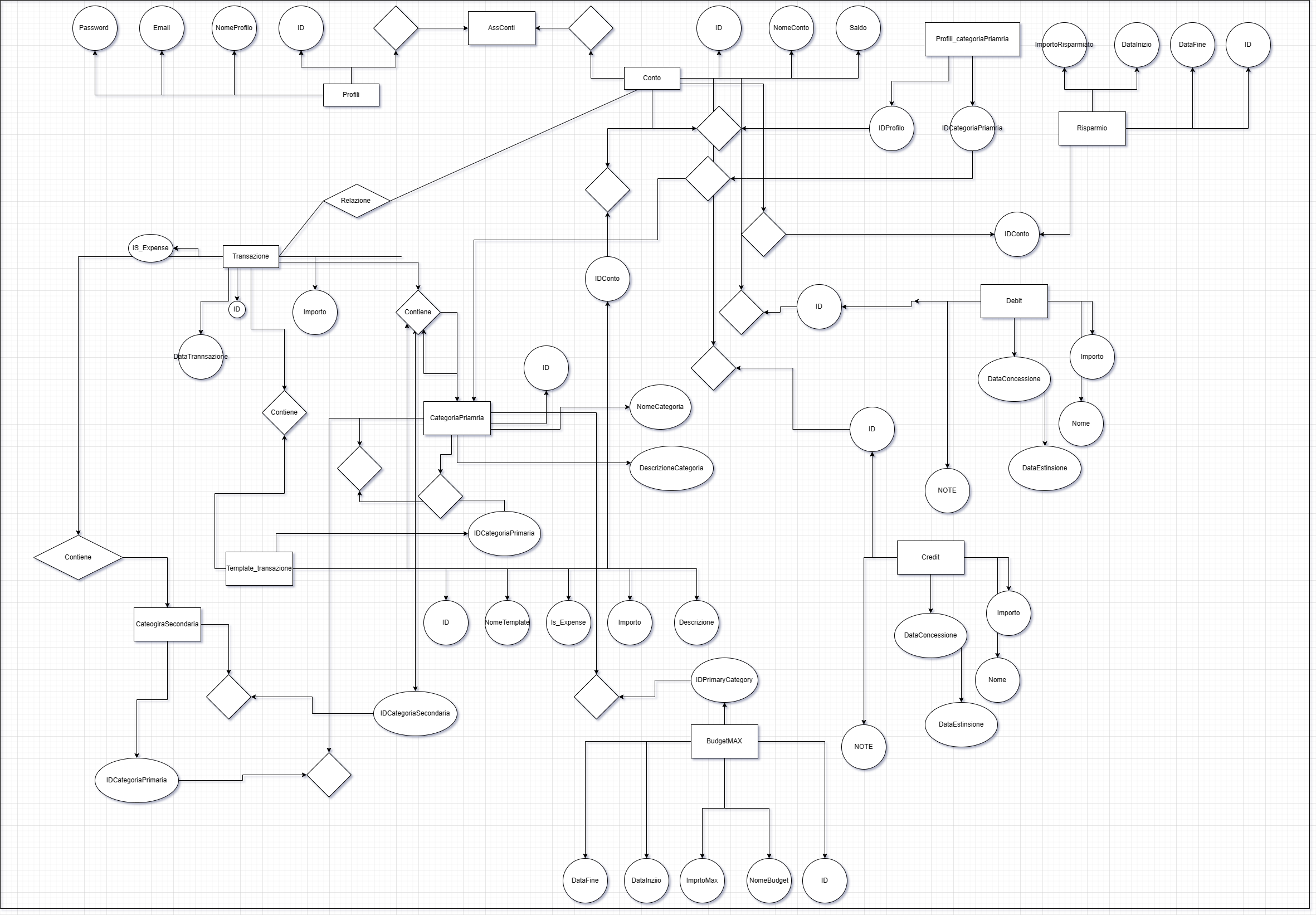
* Conti bancari
* Transazioni (entrate e uscite)
* Risparmi programmati
* Budget massimi per categorie di spesa
* Profili utente

Descrizione delle funzionalità chiave come:

* Allocare automaticamente i risparmi giornalieri
* Creare transazioni basate su template
* Gestire il credito e il debito con relative transazioni
* Monitorare e impedire superamenti di budget
* Tavola Media dei Volumi

**Progettazione Concettuale**

Diagramma E-R



**Dizionario delle Entità/Relazioni**

1. **profili**
   * **ID** (PK)
2. **conto**
   * **ID** (PK)
3. **assconti**
   * **IDProfilo** (FK) -> **profili.ID**
   * **IDConto** (FK) -> **conto.ID**
4. **categoriaprimaria**
   * **ID** (PK)
5. **categoriasecondaria**
   * **ID** (PK)
   * **IDCategoriaPrimaria** (FK) -> **categoriaprimaria.ID**
6. **budgetmax**
   * **ID** (PK)
   * **IDPrimaryCategory** (FK) -> **categoriaprimaria.ID**
7. **transazione**
   * **ID** (PK)
   * **IDConto** (FK) -> **conto.ID**
   * **IDCategoriaPrimaria** (FK) -> **categoriaprimaria.ID**
   * **IDCategoriaSecondaria** (FK) -> **categoriasecondaria.ID**
8. **credit**
   * **ID** (PK)
   * **IDConto** (FK) -> **conto.ID**
   * **IDCategoriaPrimaria** (FK) -> **categoriaprimaria.ID**
9. **debit**
   * **ID** (PK)
   * **IDConto** (FK) -> **conto.ID**
   * **IDCategoriaPrimaria** (FK) -> **categoriaprimaria.ID**
10. **profili\_categoriaprimaria**
    * **IDProfilo** (FK) -> **profili.ID**
    * **IDCategoriaPrimaria** (FK) -> **categoriaprimaria.ID**
11. **risparmi**
    * **ID** (PK)
    * **IDConto** (FK) -> **conto.ID**
    * **IDCategoriaPrimaria** (FK) -> **categoriaprimaria.ID**
12. **template\_transazioni**
    * **ID** (PK)
    * **IDConto** (FK) -> **conto.ID**
    * **IDCategoriaPrimaria** (FK) -> **categoriaprimaria.ID**
    * **IDCategoriaSecondaria** (FK) -> **categoriasecondaria.ID**

**Trigger, Stored Procedure ed Eventi**

**Trigger**

I trigger sono stati utilizzati per automatizzare le operazioni che devono essere eseguite in risposta a determinati eventi sulle tabelle del database.

1. **before\_budget\_insert\_check**: Questo trigger viene eseguito prima di un inserimento nella tabella budgetmax. Il suo scopo è verificare se la somma totale spesa per una categoria specificata nel periodo del nuovo budget supera l'importo massimo consentito. Se la somma spesa supera il budget, viene generato un errore.
2. **create\_transaction\_on\_credit\_insert:** Questo trigger viene eseguito dopo un inserimento nella tabella credit. Inserisce automaticamente una transazione nella tabella transazione per registrare il credito concesso.
3. **create\_transaction\_on\_debit\_insert:** Questo trigger viene eseguito dopo un inserimento nella tabella debit. Inserisce automaticamente una transazione nella tabella transazione per registrare il debito concesso.
4. **CheckBudgetBeforeTransaction:** Questo trigger viene eseguito prima di un inserimento nella tabella transazione. Verifica se la nuova transazione, sommata alle spese esistenti, supera il budget massimo per la categoria specificata. Se il budget viene superato, viene generato un errore.
5. **after\_transazione\_delete**: Questo trigger viene eseguito dopo una cancellazione nella tabella transazione. Aggiorna il saldo del conto associato aggiungendo l'importo della transazione cancellata se era una spesa, o sottraendolo se era un'entrata**.**
6. **after\_transazione\_insert:** Questo trigger viene eseguito dopo un inserimento nella tabella transazione. Aggiorna il saldo del conto associato sottraendo l'importo della nuova transazione se è una spesa, o aggiungendolo se è un'entrata.
7. **after\_transazione\_update:** Questo trigger viene eseguito dopo un aggiornamento nella tabella transazione. Aggiorna il saldo del conto associato annullando l'effetto della vecchia transazione e applicando l'effetto della nuova transazione.

**Stored Procedures**

Le procedure memorizzate sono state create per eseguire operazioni complesse che richiedono più passi.

1. **AllocateSavings:** Questa procedura distribuisce l'importo totale dei risparmi su base giornaliera tra le date di inizio e fine specificate, aggiornando il saldo del conto e inserendo una transazione per ogni giorno.
2. **AllocateSavingsDaily:** Questa procedura chiama la procedura AllocateSavings per ogni ID di risparmio presente nel sistema, automatizzando la distribuzione giornaliera dei risparmi.
3. **CreateTransactionFromTemplate:** Questa procedura crea una nuova transazione basata su un template specificato, inserendo tutti i dettagli della transazione (tipo di spesa, importo, conto, categoria) nella tabella transazione.
4. **create\_transaction\_on\_credit\_termination**: Crea una nuova transazione per la terminazione del credito ed elimina il credito dalla tabella credit.
5. **create\_transaction\_on\_debit\_termination**: Crea una nuova transazione per la terminazione del debito ed elimina il debito dalla tabella debit.
6. **GenerateFinancialReport**: Questa procedura genera un report finanziario basato su vari criteri di filtro forniti come input (data di inizio, data di fine, tipo di transazione, ID del conto, ID della categoria primaria e ID della categoria secondaria). Seleziona e restituisce le transazioni dalla tabella transazione che soddisfano i criteri specificati in formato CSV.

**Eventi**

Gli eventi sono stati programmati per eseguire automaticamente le procedure memorizzate a intervalli regolari.

1. **allocateSavingsEvent:** Questo evento viene eseguito ogni giorno e chiama la procedura AllocateSavingsDaily per distribuire i risparmi giornalieri.
2. **check\_debit\_credit\_expiry\_event**: Questo evento viene eseguito ogni giorno e controlla se ci sono crediti o debiti che scadono. Se trova scadenze, inserisce automaticamente le transazioni appropriate nella tabella transazione.

**Descrizione delle Funzionalità dell'Applicazione Web**

Cash Flow Web è un'applicazione sviluppata per gestire e monitorare le transazioni finanziarie di una persona fisica. Le principali funzionalità dell'applicazione sono descritte di seguito:

1. Registrazione e Autenticazione Utente: Gli utenti possono registrarsi al sistema inserendo un’e-mail e una password. Una volta registrati, possono accedere al sistema utilizzando le stesse credenziali. Le informazioni e le transazioni visualizzate sono specifiche per ogni utente, garantendo la privacy e la sicurezza dei dati personali.
2. Gestione delle Transazioni: L'applicazione consente agli utenti di tenere traccia di tutte le transazioni finanziarie in entrata e in uscita. Le transazioni possono essere categorizzate utilizzando conti e categorie specifiche. Ad esempio, le spese al supermercato possono essere registrate come una categoria, mentre l'entrata dello stipendio può essere un'altra. Le categorie sono divise in primarie e secondarie, con quest'ultime dipendenti dalle categorie primarie.
3. Creazione e Gestione dei Conti: Gli utenti possono creare conti diversi per suddividere le transazioni. Ad esempio, è possibile creare un conto per il conto corrente e uno per la carta di credito. Ogni transazione deve essere associata a uno specifico conto, permettendo agli utenti di monitorare separatamente il saldo e le transazioni per ogni conto.
4. Creazione e Gestione delle Categorie: Gli utenti possono creare categorie primarie e secondarie per organizzare meglio le loro transazioni. Durante la creazione di una transazione, l'inserimento della categoria secondaria è opzionale.
5. Template di Transazione: L'applicazione permette di creare template di transazioni, che includono informazioni come importo, tipo (entrata o uscita), conto, categorie primarie e secondarie, e una descrizione. Utilizzando questi template, gli utenti possono creare nuove transazioni in modo rapido ed efficiente.
6. Gestione di Debiti e Crediti: Gli utenti possono registrare debiti e crediti inserendo dettagli come importo, conto, categoria, data di inizio e data di fine. Al momento dell'inserimento, vengono create due transazioni: una alla data di inizio e una alla data di fine, tenendo conto se si tratta di un'entrata o di un'uscita.
7. Impostazione di Budget Massimi: È possibile impostare un budget massimo per una categoria specifica. Se una nuova transazione supera l'importo massimo del budget impostato per quella categoria, il sistema impedisce la creazione della transazione, aiutando gli utenti a mantenere il controllo delle loro spese.
8. Gestione dei Risparmi: Gli utenti possono creare obiettivi di risparmio, che dividono l'importo totale del risparmio per un determinato numero di giorni. Ogni giorno, viene creata automaticamente una transazione per l'importo del risparmio giornaliero.
9. Generazione di Report Finanziari: L'applicazione consente agli utenti di generare report finanziari personalizzati. Gli utenti possono selezionare intervalli di date, tipi di transazioni (entrate, uscite), conti specifici e categorie per visualizzare un riepilogo dettagliato delle loro finanze. I report possono includere grafici e tabelle che mostrano l'andamento delle spese e delle entrate nel tempo, aiutando gli utenti a identificare tendenze e a prendere decisioni finanziarie informate. Inoltre, i report possono essere esportati in formati PDF o Excel per un'analisi più approfondita.
10. Logout: Gli utenti possono effettuare il logout per terminare la sessione, garantendo la sicurezza dei dati personali.

Queste funzionalità permettono agli utenti di gestire in modo efficiente le proprie finanze, fornendo un sistema completo e user-friendly per il monitoraggio delle transazioni, la gestione dei risparmi e il controllo del budget.

**CODICE SQL:**

SET

  SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

SET

  time\_zone = "+00:00";

CREATE DATABASE IF NOT EXISTS `cashflowweb` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_general\_ci;

USE `cashflowweb`;

DELIMITER $ $ DROP PROCEDURE IF EXISTS `AllocateSavings` $ $ CREATE DEFINER = `root` @`localhost` PROCEDURE `AllocateSavings` (IN `SavingsID` INT) BEGIN DECLARE StartDate DATE;

DECLARE EndDate DATE;

DECLARE TotalAmount DECIMAL(10, 2);

DECLARE AccountID INT;

DECLARE Days INT;

DECLARE DailyAmount DECIMAL(10, 2);

DECLARE PrimaryCategoryID INT;

-- Seleziona i dettagli del risparmio

SELECT

  DataInizio,

  DataFine,

  ImportoRisparmiato,

  IDConto,

  IDCategoriaPrimaria INTO StartDate,

  EndDate,

  TotalAmount,

  AccountID,

  PrimaryCategoryID

FROM

  risparmi

WHERE

  ID = SavingsID;

-- Calcola il numero di giorni (+1 per includere sia la DataInizio che la DataFine)

SET

  Days = DATEDIFF(EndDate, StartDate) + 1;

-- Evita la divisione per zero

IF Days > 0 THEN

SET

  DailyAmount = TotalAmount / Days;

-- Controlla se il saldo del conto è sufficiente

IF (

  SELECT

    Saldo

  FROM

    conto

  WHERE

    ID = AccountID

) >= DailyAmount THEN -- Inserisce una transazione solo per il giorno corrente

IF CURDATE() BETWEEN StartDate

AND EndDate THEN -- Aggiorna il saldo del conto

UPDATE

  conto

SET

  Saldo = Saldo - DailyAmount

WHERE

  ID = AccountID;

-- Aggiungi una transazione per il risparmio giornaliero

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria,

    IDCategoriaSecondaria

  )

VALUES

  (

    1,

    DailyAmount,

    AccountID,

    CURDATE(),

    PrimaryCategoryID,

    NULL

  );

END IF;

END IF;

END IF;

END $ $ DROP PROCEDURE IF EXISTS `AllocateSavingsDaily` $ $ CREATE DEFINER = `root` @`localhost` PROCEDURE `AllocateSavingsDaily` () BEGIN DECLARE done INT DEFAULT FALSE;

DECLARE aSavingsID INT;

DECLARE cur CURSOR FOR

SELECT

  ID

FROM

  risparmi

WHERE

  DataFine >= CURDATE();

-- Seleziona solo i risparmi attivi

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET

  done = TRUE;

OPEN cur;

read\_loop: LOOP FETCH cur INTO aSavingsID;

IF done THEN LEAVE read\_loop;

END IF;

-- Chiama il procedimento per allocare i risparmi

CALL AllocateSavings(aSavingsID);

END LOOP;

CLOSE cur;

END $ $ DROP PROCEDURE IF EXISTS `CreateTransactionFromTemplate` $ $ CREATE DEFINER = `root` @`localhost` PROCEDURE `CreateTransactionFromTemplate` (IN `TemplateID` INT) BEGIN DECLARE ExpenseType TINYINT;

DECLARE Amount DECIMAL(10, 2);

DECLARE AccountID INT;

DECLARE PrimaryCategoryID INT;

DECLARE SecondaryCategoryID INT;

DECLARE Description VARCHAR(255);

SELECT

  Is\_Expense,

  Importo,

  IDConto,

  IDCategoriaPrimaria,

  IDCategoriaSecondaria,

  Descrizione INTO ExpenseType,

  Amount,

  AccountID,

  PrimaryCategoryID,

  SecondaryCategoryID,

  Description

FROM

  template\_transazioni

WHERE

  ID = TemplateID;

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDTemplate,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria,

    IDCategoriaSecondaria

  )

VALUES

  (

    ExpenseType,

    Amount,

    TemplateID,

    AccountID,

    CURDATE(),

    PrimaryCategoryID,

    SecondaryCategoryID

  );

END $ $ DROP PROCEDURE IF EXISTS `create\_transaction\_on\_credit\_termination` $ $ CREATE DEFINER = `root` @`localhost` PROCEDURE `create\_transaction\_on\_credit\_termination` (IN `creditID` INT) BEGIN DECLARE currentDate DATE;

DECLARE creditAmount DECIMAL(10, 2);

DECLARE accountID INT;

DECLARE primaryCategoryID INT;

SET

  currentDate = CURDATE();

-- Recupera i dettagli del credito

SELECT

  ImportoCredito,

  IDConto,

  IDCategoriaPrimaria INTO creditAmount,

  accountID,

  primaryCategoryID

FROM

  credit

WHERE

  ID = creditID;

-- Crea una nuova transazione per la terminazione del credito

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria

  )

VALUES

  (

    0,

    -- Credito (entrata)

    creditAmount,

    accountID,

    currentDate,

    primaryCategoryID

  );

-- Elimina il credito dopo aver creato la transazione

DELETE FROM

  credit

WHERE

  ID = creditID;

END $ $ DROP PROCEDURE IF EXISTS `create\_transaction\_on\_debit\_termination` $ $ CREATE DEFINER = `root` @`localhost` PROCEDURE `create\_transaction\_on\_debit\_termination` (IN `debitID` INT) BEGIN DECLARE currentDate DATE;

DECLARE debitAmount DECIMAL(10, 2);

DECLARE accountID INT;

DECLARE primaryCategoryID INT;

SET

  currentDate = CURDATE();

-- Recupera i dettagli del debito

SELECT

  ImportoDebito,

  IDConto,

  IDCategoriaPrimaria INTO debitAmount,

  accountID,

  primaryCategoryID

FROM

  debit

WHERE

  ID = debitID;

-- Crea una nuova transazione per la terminazione del debito

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria

  )

VALUES

  (

    1,

    -- Debito (uscita)

    debitAmount,

    accountID,

    currentDate,

    primaryCategoryID

  );

-- Elimina il debito dopo aver creato la transazione

DELETE FROM

  debit

WHERE

  ID = debitID;

END $ $ DELIMITER;

DROP TABLE IF EXISTS `assconti`;

CREATE TABLE `assconti` (

  `IDProfilo` int(11) DEFAULT NULL,

  `IDConto` int(11) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TABLE IF EXISTS `budgetmax`;

CREATE TABLE `budgetmax` (

  `ID` int(11) NOT NULL,

  `NomeBudget` varchar(255) DEFAULT NULL,

  `ImportoMax` decimal(10, 2) DEFAULT NULL,

  `DataInizio` date DEFAULT NULL,

  `DataFine` date DEFAULT NULL,

  `IDPrimaryCategory` int(11) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TRIGGER IF EXISTS `before\_budget\_insert\_check`;

DELIMITER $ $ CREATE TRIGGER `before\_budget\_insert\_check` BEFORE

INSERT

  ON `budgetmax` FOR EACH ROW BEGIN DECLARE TotalSpent DECIMAL(10, 2);

-- Calcola la somma totale spesa per la categoria specificata nel periodo del nuovo budget

SELECT

  SUM(t.Importo) INTO TotalSpent

FROM

  transazione t

WHERE

  t.IDCategoriaPrimaria = NEW.IDPrimaryCategory

  AND t.Is\_Expense = 1

  AND t.DataTransazione BETWEEN NEW.DataInizio

  AND NEW.DataFine;

-- Verifica se la somma spesa supera il budget massimo

IF TotalSpent IS NOT NULL

AND TotalSpent > NEW.ImportoMax THEN SIGNAL SQLSTATE '45000'

SET

  MESSAGE\_TEXT = 'Il budget inserito è già stato superato.';

END IF;

END $ $ DELIMITER;

DROP TABLE IF EXISTS `categoriaprimaria`;

CREATE TABLE `categoriaprimaria` (

  `ID` int(11) NOT NULL,

  `NomeCategoria` varchar(255) DEFAULT NULL,

  `DescrizioneCategoria` varchar(255) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TABLE IF EXISTS `categoriasecondaria`;

CREATE TABLE `categoriasecondaria` (

  `ID` int(11) NOT NULL,

  `IDCategoriaPrimaria` int(11) DEFAULT NULL,

  `NomeCategoria` varchar(255) DEFAULT NULL,

  `DescrizioneCategoria` varchar(255) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TABLE IF EXISTS `conto`;

CREATE TABLE `conto` (

  `ID` int(11) NOT NULL,

  `NomeConto` varchar(255) DEFAULT NULL,

  `Saldo` decimal(10, 2) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TABLE IF EXISTS `credit`;

CREATE TABLE `credit` (

  `ID` int(11) NOT NULL,

  `ImportoCredito` decimal(10, 2) DEFAULT NULL,

  `NomeImporto` varchar(255) DEFAULT NULL,

  `DataConcessione` date DEFAULT NULL,

  `DataEstinsione` date DEFAULT NULL,

  `Note` varchar(255) DEFAULT NULL,

  `IDConto` int(11) DEFAULT NULL,

  `IDCategoriaPrimaria` int(11) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TRIGGER IF EXISTS `create\_transaction\_on\_credit\_insert`;

DELIMITER $ $ CREATE TRIGGER `create\_transaction\_on\_credit\_insert`

AFTER

INSERT

  ON `credit` FOR EACH ROW BEGIN

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria

  )

VALUES

  (

    1,

    NEW.ImportoCredito,

    NEW.IDConto,

    NEW.DataConcessione,

    NEW.IDCategoriaPrimaria

  );

END $ $ DELIMITER;

DROP TABLE IF EXISTS `debit`;

CREATE TABLE `debit` (

  `ID` int(11) NOT NULL,

  `ImportoDebito` decimal(10, 2) DEFAULT NULL,

  `NomeImporto` varchar(255) DEFAULT NULL,

  `DataConcessione` date DEFAULT NULL,

  `DataEstinsione` date DEFAULT NULL,

  `Note` varchar(255) DEFAULT NULL,

  `IDConto` int(11) DEFAULT NULL,

  `IDCategoriaPrimaria` int(11) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TRIGGER IF EXISTS `create\_transaction\_on\_debit\_insert`;

DELIMITER $ $ CREATE TRIGGER `create\_transaction\_on\_debit\_insert`

AFTER

INSERT

  ON `debit` FOR EACH ROW BEGIN

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria

  )

VALUES

  (

    0,

    NEW.ImportoDebito,

    NEW.IDConto,

    NEW.DataConcessione,

    NEW.IDCategoriaPrimaria

  );

END $ $ DELIMITER;

DROP TABLE IF EXISTS `profili`;

CREATE TABLE `profili` (

  `ID` int(11) NOT NULL,

  `NomeProfilo` varchar(255) DEFAULT NULL,

  `Email` varchar(255) DEFAULT NULL,

  `Password` varchar(255) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TABLE IF EXISTS `profili\_categoriaprimaria`;

CREATE TABLE `profili\_categoriaprimaria` (

  `IDProfilo` int(11) NOT NULL,

  `IDCategoriaPrimaria` int(11) NOT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TABLE IF EXISTS `risparmi`;

CREATE TABLE `risparmi` (

  `ID` int(11) NOT NULL,

  `ImportoRisparmiato` decimal(10, 2) DEFAULT NULL,

  `DataInizio` date DEFAULT NULL,

  `DataFine` date DEFAULT NULL,

  `IDConto` int(11) DEFAULT NULL,

  `IDCategoriaPrimaria` int(11) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TABLE IF EXISTS `template\_transazioni`;

CREATE TABLE `template\_transazioni` (

  `ID` int(11) NOT NULL,

  `NomeTemplate` varchar(255) DEFAULT NULL,

  `Is\_Expense` tinyint(1) DEFAULT NULL,

  `Importo` decimal(10, 2) DEFAULT NULL,

  `IDConto` int(11) DEFAULT NULL,

  `IDCategoriaPrimaria` int(11) DEFAULT NULL,

  `IDCategoriaSecondaria` int(11) DEFAULT NULL,

  `Descrizione` varchar(255) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TABLE IF EXISTS `transazione`;

CREATE TABLE `transazione` (

  `ID` int(11) NOT NULL,

  `Is\_Expense` tinyint(1) DEFAULT NULL,

  `Importo` decimal(10, 2) DEFAULT NULL,

  `IDTemplate` int(11) DEFAULT NULL,

  `IDConto` int(11) DEFAULT NULL,

  `DataTransazione` date DEFAULT NULL,

  `IDCategoriaPrimaria` int(11) DEFAULT NULL,

  `IDCategoriaSecondaria` int(11) DEFAULT NULL

) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4\_general\_ci;

DROP TRIGGER IF EXISTS `CheckBudgetBeforeTransaction`;

DELIMITER $ $ CREATE TRIGGER `CheckBudgetBeforeTransaction` BEFORE

INSERT

  ON `transazione` FOR EACH ROW BEGIN DECLARE MaxAmount DECIMAL(10, 2);

DECLARE TotalSpent DECIMAL(10, 2);

SELECT

  ImportoMax INTO MaxAmount

FROM

  budgetmax

WHERE

  IDPrimaryCategory = NEW.IDCategoriaPrimaria

  AND CURDATE() BETWEEN DataInizio

  AND DataFine;

IF MaxAmount IS NOT NULL THEN

SELECT

  SUM(Importo) INTO TotalSpent

FROM

  transazione

WHERE

  IDCategoriaPrimaria = NEW.IDCategoriaPrimaria

  AND Is\_Expense = 1;

IF (TotalSpent + NEW.Importo > MaxAmount) THEN SIGNAL SQLSTATE '45000'

SET

  MESSAGE\_TEXT = 'Budget limit exceeded for this category.';

END IF;

END IF;

END $ $ DELIMITER;

DROP TRIGGER IF EXISTS `after\_transazione\_delete`;

DELIMITER $ $ CREATE TRIGGER `after\_transazione\_delete`

AFTER

  DELETE ON `transazione` FOR EACH ROW BEGIN IF OLD.Is\_Expense = 1 THEN

UPDATE

  conto

SET

  Saldo = Saldo + OLD.Importo

WHERE

  ID = OLD.IDConto;

ELSE

UPDATE

  conto

SET

  Saldo = Saldo - OLD.Importo

WHERE

  ID = OLD.IDConto;

END IF;

END $ $ DELIMITER;

DROP TRIGGER IF EXISTS `after\_transazione\_insert`;

DELIMITER $ $ CREATE TRIGGER `after\_transazione\_insert`

AFTER

INSERT

  ON `transazione` FOR EACH ROW BEGIN IF NEW.Is\_Expense = 1 THEN

UPDATE

  conto

SET

  Saldo = Saldo - NEW.Importo

WHERE

  ID = NEW.IDConto;

ELSE

UPDATE

  conto

SET

  Saldo = Saldo + NEW.Importo

WHERE

  ID = NEW.IDConto;

END IF;

END $ $ DELIMITER;

DROP TRIGGER IF EXISTS `after\_transazione\_update`;

DELIMITER $ $ CREATE TRIGGER `after\_transazione\_update`

AFTER

UPDATE

  ON `transazione` FOR EACH ROW BEGIN IF OLD.Is\_Expense = 1 THEN

UPDATE

  conto

SET

  Saldo = Saldo + OLD.Importo

WHERE

  ID = OLD.IDConto;

ELSE

UPDATE

  conto

SET

  Saldo = Saldo - OLD.Importo

WHERE

  ID = OLD.IDConto;

END IF;

IF NEW.Is\_Expense = 1 THEN

UPDATE

  conto

SET

  Saldo = Saldo - NEW.Importo

WHERE

  ID = NEW.IDConto;

ELSE

UPDATE

  conto

SET

  Saldo = Saldo + NEW.Importo

WHERE

  ID = NEW.IDConto;

END IF;

END $ $ DELIMITER;

ALTER TABLE

  `assconti`

ADD

  KEY `fk\_assconti\_profilo` (`IDProfilo`),

ADD

  KEY `fk\_assconti\_conto` (`IDConto`);

ALTER TABLE

  `budgetmax`

ADD

  PRIMARY KEY (`ID`),

ADD

  KEY `fk\_budgetmax\_primarycategory` (`IDPrimaryCategory`);

ALTER TABLE

  `categoriaprimaria`

ADD

  PRIMARY KEY (`ID`);

ALTER TABLE

  `categoriasecondaria`

ADD

  PRIMARY KEY (`ID`),

ADD

  KEY `categoriasecondaria\_primaria\_fk` (`IDCategoriaPrimaria`);

ALTER TABLE

  `conto`

ADD

  PRIMARY KEY (`ID`);

ALTER TABLE

  `credit`

ADD

  PRIMARY KEY (`ID`),

ADD

  KEY `credit\_conto\_fk` (`IDConto`),

ADD

  KEY `fk\_credit\_categoriaprimaria` (`IDCategoriaPrimaria`);

ALTER TABLE

  `debit`

ADD

  PRIMARY KEY (`ID`),

ADD

  KEY `debit\_conto\_fk` (`IDConto`),

ADD

  KEY `fk\_debit\_categoriaprimaria` (`IDCategoriaPrimaria`);

ALTER TABLE

  `profili`

ADD

  PRIMARY KEY (`ID`);

ALTER TABLE

  `profili\_categoriaprimaria`

ADD

  PRIMARY KEY (`IDProfilo`, `IDCategoriaPrimaria`),

ADD

  KEY `fk\_profili\_categoriaprimaria\_profilo` (`IDProfilo`),

ADD

  KEY `fk\_profili\_categoriaprimaria\_categoria` (`IDCategoriaPrimaria`);

ALTER TABLE

  `risparmi`

ADD

  PRIMARY KEY (`ID`),

ADD

  KEY `risparmi\_conto\_fk` (`IDConto`),

ADD

  KEY `fk\_risparmi\_categoriaprimaria` (`IDCategoriaPrimaria`);

ALTER TABLE

  `template\_transazioni`

ADD

  PRIMARY KEY (`ID`),

ADD

  KEY `template\_transazioni\_conto\_fk` (`IDConto`),

ADD

  KEY `template\_transazioni\_primaria\_fk` (`IDCategoriaPrimaria`),

ADD

  KEY `template\_transazioni\_secondaria\_fk` (`IDCategoriaSecondaria`);

ALTER TABLE

  `transazione`

ADD

  PRIMARY KEY (`ID`),

ADD

  KEY `transazione\_template\_fk` (`IDTemplate`),

ADD

  KEY `transazione\_conto\_fk` (`IDConto`),

ADD

  KEY `transazione\_primaria\_fk` (`IDCategoriaPrimaria`),

ADD

  KEY `transazione\_secondaria\_fk` (`IDCategoriaSecondaria`);

ALTER TABLE

  `budgetmax`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `categoriaprimaria`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `categoriasecondaria`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `conto`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `credit`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `debit`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `profili`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `risparmi`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `template\_transazioni`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `transazione`

MODIFY

  `ID` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE

  `assconti`

ADD

  CONSTRAINT `fk\_assconti\_conto` FOREIGN KEY (`IDConto`) REFERENCES `conto` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,

ADD

  CONSTRAINT `fk\_assconti\_profilo` FOREIGN KEY (`IDProfilo`) REFERENCES `profili` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE

  `budgetmax`

ADD

  CONSTRAINT `fk\_budgetmax\_primarycategory` FOREIGN KEY (`IDPrimaryCategory`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE;

ALTER TABLE

  `categoriasecondaria`

ADD

  CONSTRAINT `categoriasecondaria\_primaria\_fk` FOREIGN KEY (`IDCategoriaPrimaria`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE

  `credit`

ADD

  CONSTRAINT `credit\_conto\_fk` FOREIGN KEY (`IDConto`) REFERENCES `conto` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,

ADD

  CONSTRAINT `fk\_credit\_categoriaprimaria` FOREIGN KEY (`IDCategoriaPrimaria`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE;

ALTER TABLE

  `debit`

ADD

  CONSTRAINT `debit\_conto\_fk` FOREIGN KEY (`IDConto`) REFERENCES `conto` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,

ADD

  CONSTRAINT `fk\_debit\_categoriaprimaria` FOREIGN KEY (`IDCategoriaPrimaria`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE;

ALTER TABLE

  `profili\_categoriaprimaria`

ADD

  CONSTRAINT `fk\_profili\_categoriaprimaria\_categoria` FOREIGN KEY (`IDCategoriaPrimaria`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,

ADD

  CONSTRAINT `fk\_profili\_categoriaprimaria\_profilo` FOREIGN KEY (`IDProfilo`) REFERENCES `profili` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE

  `risparmi`

ADD

  CONSTRAINT `fk\_risparmi\_categoriaprimaria` FOREIGN KEY (`IDCategoriaPrimaria`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE,

ADD

  CONSTRAINT `risparmi\_conto\_fk` FOREIGN KEY (`IDConto`) REFERENCES `conto` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE

  `template\_transazioni`

ADD

  CONSTRAINT `fk\_template\_transazioni\_primaria` FOREIGN KEY (`IDCategoriaPrimaria`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE,

ADD

  CONSTRAINT `fk\_template\_transazioni\_primaria\_new` FOREIGN KEY (`IDCategoriaPrimaria`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE,

ADD

  CONSTRAINT `fk\_template\_transazioni\_secondaria` FOREIGN KEY (`IDCategoriaSecondaria`) REFERENCES `categoriasecondaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE,

ADD

  CONSTRAINT `fk\_template\_transazioni\_secondaria\_new` FOREIGN KEY (`IDCategoriaSecondaria`) REFERENCES `categoriasecondaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE,

ADD

  CONSTRAINT `template\_transazioni\_conto\_fk` FOREIGN KEY (`IDConto`) REFERENCES `conto` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE

  `transazione`

ADD

  CONSTRAINT `fk\_transazione\_primaria` FOREIGN KEY (`IDCategoriaPrimaria`) REFERENCES `categoriaprimaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE,

ADD

  CONSTRAINT `fk\_transazione\_secondaria` FOREIGN KEY (`IDCategoriaSecondaria`) REFERENCES `categoriasecondaria` (`ID`) ON DELETE

SET

  NULL ON UPDATE CASCADE,

ADD

  CONSTRAINT `transazione\_conto\_fk` FOREIGN KEY (`IDConto`) REFERENCES `conto` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE;

DELIMITER $ $ DROP EVENT IF EXISTS `allocateSavingsEvent` $ $ CREATE DEFINER = `root` @`localhost` EVENT `allocateSavingsEvent` ON SCHEDULE EVERY 1 DAY STARTS '2024-05-15 00:00:00' ON COMPLETION NOT PRESERVE ENABLE DO CALL AllocateSavingsDaily() $ $ DROP EVENT IF EXISTS `check\_debit\_credit\_expiry\_event` $ $ CREATE DEFINER = `root` @`localhost` EVENT `check\_debit\_credit\_expiry\_event` ON SCHEDULE EVERY 1 DAY STARTS '2024-05-08 00:00:00' ON COMPLETION NOT PRESERVE ENABLE DO BEGIN DECLARE done INT DEFAULT FALSE;

DECLARE debtCreditID INT;

DECLARE debtCreditType VARCHAR(10);

DECLARE cur CURSOR FOR

SELECT

  ID,

  'debit' AS type

FROM

  debit

WHERE

  DataEstinsione = CURDATE()

UNION

ALL

SELECT

  ID,

  'credit' AS type

FROM

  credit

WHERE

  DataEstinsione = CURDATE();

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET

  done = TRUE;

OPEN cur;

read\_loop: LOOP FETCH cur INTO debtCreditID,

debtCreditType;

IF done THEN LEAVE read\_loop;

END IF;

IF debtCreditType = 'debit' THEN

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria

  )

SELECT

  1,

  ImportoDebito,

  IDConto,

  DataEstinsione,

  IDCategoriaPrimaria

FROM

  debit

WHERE

  ID = debtCreditID;

ELSE

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria

  )

SELECT

  0,

  ImportoCredito,

  IDConto,

  DataEstinsione,

  IDCategoriaPrimaria

FROM

  credit

WHERE

  ID = debtCreditID;

END IF;

END LOOP;

CLOSE cur;

END $ $ DELIMITER;