**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 sito web che ti permette di gestire il proprio cash flow personale o attraverso il salvataggio di transazioni, conti, categorie.

**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.
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.

**Raccolta e Analisi dei Requisiti**

**Specifiche sui Dati**

Elenco e descrizione dei dati gestiti dal sistema, come:

* Conti
* 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

Tecnologie utilizzate:

Nel corso dello sviluppo del progetto "Cash Flow Web" sono state impiegate diverse tecnologie per garantire funzionalità complete e una gestione efficace dei dati:

**PHP**: Utilizzato per la creazione di file client e server, PHP ha permesso di gestire la logica di backend del sistema.

**XAMPP**: Impiegato come ambiente di sviluppo locale, XAMPP ha facilitato l'integrazione tra MySQL e PHP, offrendo un ambiente di test efficace prima del rilascio live.

**MongoDB**: Adottato per salvare tutti i log di inserimento ed eliminazione operazioni nel database, MongoDB fornisce una soluzione scalabile e performante per la gestione dei dati di log.

**JavaScript: AJAX e Chart.js:** Utilizzati per migliorare l'interfaccia utente, JavaScript, in combinazione con AJAX, ha permesso di aggiornare dinamicamente le categorie secondarie di transazioni senza necessità di ricaricare la pagina. Chart.js, invece, è una libreria JavaScript per la creazione di grafici interattivi, impiegata per visualizzare i dati in forma grafica.

**Progettazione Concettuale**

A diagram of a network

Description automatically generatedDiagramma E-R

**Dizionario delle Entità/Relazioni**

1. **profili**
   * ID (PK)
   * Email
   * Nome profile
   * Password
2. **conto**
   * ID (PK)
   * nomeConto
   * Saldo
3. **assconti**
   * IDProfilo (FK) -> profili.ID
   * IDConto (FK) -> conto.ID
4. **categoriaprimaria**
   * ID (PK)
   * nomeCategoria
   * DescrizioneCategoria
5. **categoriasecondaria**
   * ID (PK)
   * IDCategoriaPrimaria (FK) -> categoriaprimaria.ID
   * NomeCategoria
   * DescrizioneCategoria
6. **budgetmax**
   * ID (PK)
   * IDPrimaryCategory (FK) -> categoriaprimaria.ID
   * NomeBudget
   * DataInizio
   * DataFine
7. **transazione**
   * ID (PK)
   * IDConto (FK) -> conto.ID
   * IDCategoriaPrimaria (FK) -> categoriaprimaria.ID
   * IDCategoriaSecondaria (FK) -> categoriasecondaria.ID
   * Is\_expense
   * Import
   * dataTransazione
8. **credit**
   * ID (PK)
   * IDConto (FK) -> conto.ID
   * IDCategoriaPrimaria (FK) -> categoriaprimaria.ID
   * ImportoCredito
   * DataConcessione
   * DataEstinsione
9. **debit**
   * ID (PK)
   * IDConto (FK) -> conto.ID
   * IDCategoriaPrimaria (FK) -> categoriaprimaria.ID
   * ImportoDebito
   * DataConcessione
   * DataEstinsione
   * Note
10. **profili\_categoriaprimaria**
    * IDProfilo (FK) -> profili.ID
    * IDCategoriaPrimaria (FK) -> categoriaprimaria.ID
11. **risparmi**
    * ID (PK)
    * IDConto (FK) -> conto.ID
    * IDCategoriaPrimaria (FK) -> categoriaprimaria.ID
    * importoRisparmiato
    * DataInzio
    * DataFine
12. **template\_transazioni**
    * ID (PK)
    * IDConto (FK) -> conto.ID
    * IDCategoriaPrimaria (FK) -> categoriaprimaria.ID
    * IDCategoriaSecondaria (FK) -> categoriasecondaria.ID
    * NomeTemplate
    * Is\_expense
    * Import
    * descizione

**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.

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;

1. **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.

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;

1. **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.

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;

1. **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.

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;

1. **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**.**

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;

1. **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.

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;

1. **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.

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;

**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.

DELIMITER $$

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$$

1. **AllocateSavingsDaily:** Questa procedura chiama la procedura **AllocateSavings per ogni ID di risparmio** **attivo** presente nel sistema, automatizzando la distribuzione giornaliera dei risparmi.

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$$

1. **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.

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 $ $

1. **create\_transaction\_on\_credit\_termination**: Crea una nuova transazione per la terminazione del credito ed elimina il credito dalla tabella credit.

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 $ $

1. **create\_transaction\_on\_debit\_termination**: Crea una nuova transazione per la terminazione del debito ed elimina il debito dalla tabella debit.

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;

INSERT INTO

  transazione (

    Is\_Expense,

    Importo,

    IDConto,

    DataTransazione,

    IDCategoriaPrimaria

  )

VALUES

  (

    1,

    -- Debito (uscita)

    debitAmount,

    accountID,

    currentDate,

    primaryCategoryID

  );

DELETE FROM

  debit

WHERE

  ID = debitID;

END $ $

1. **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.

CREATE DEFINER = `root` @`localhost` PROCEDURE `GenerateFinancialReport` (

  IN `startDate` DATE,

  IN `endDate` DATE,

  IN `transactionType` TINYINT,

  IN `accountId` INT,

  IN `primaryCategoryId` INT,

  IN `secondaryCategoryId` INT

) BEGIN

SELECT

  t.ID,

  t.Is\_Expense,

  t.Importo,

  t.IDConto,

  t.DataTransazione,

  t.IDCategoriaPrimaria,

  t.IDCategoriaSecondaria

FROM

  transazione t

WHERE

  (

    startDate IS NULL

    OR t.DataTransazione >= startDate

  )

  AND (

    endDate IS NULL

    OR t.DataTransazione <= endDate

  )

  AND (

    transactionType = -1

    OR t.Is\_Expense = transactionType

  )

  AND (

    accountId IS NULL

    OR t.IDConto = accountId

  )

  AND (

    primaryCategoryId IS NULL

    OR t.IDCategoriaPrimaria = primaryCategoryId

  )

  AND (

    secondaryCategoryId IS NULL

    OR t.IDCategoriaSecondaria = secondaryCategoryId

  );

END $ $

1. **Delete Procedures**: Queste procedure sono utilizzate per eliminare record specifici dalle tabelle del database, assicurando che solo gli ID validi e corretti vengano rimossi per prevenire cancellazioni accidentali.

* **DeleteBudget**: Rimuove un record dalla tabella budgetmax utilizzando l'ID specificato, eliminando il budget corrispondente.
* **DeleteConto**: Cancella un conto bancario dalla tabella conto basandosi sull'ID fornito.
* **DeleteCredito**: Elimina un credito registrato nella tabella credit, specificando l'ID del credito da rimuovere.
* **DeleteDebito**: Rimuove un debito dalla tabella debit, utilizzando l'ID del debito per identificarlo.
* **DeletePrimaryCategory**: Cancella una categoria primaria dalla tabella categoriaprimaria attraverso l'ID della categoria.
* **DeleteSecondaryCategory**: Elimina una categoria secondaria dalla tabella categoriasecondaria, specificando l'ID della categoria secondaria.
* **DeleteRisparmio**: Rimuove un record di risparmio dalla tabella risparmi utilizzando l'ID del risparmio specificato.
* **DeleteTemplateTransaction**: Cancella un template di transazione dalla tabella template\_transazioni, identificato dall'ID del template.
* **DeleteTransaction**: Elimina una transazione dalla tabella transazione utilizzando l'ID della transazione.

1. **Insert Procedures**: Queste procedure inseriscono nuovi dati nelle tabelle, facilitando la creazione di nuovi record con parametri chiaramente definiti.

* **InsertBudget**: Inserisce un nuovo record nella tabella budgetmax, specificando nome, importo massimo, data di inizio e fine, e categoria primaria.
* **InsertConto**: Aggiunge un nuovo conto bancario alla tabella conto, includendo il nome del conto e il saldo iniziale, crea anche l’associazione con il profilo.
* **InsertCredit**: Registra un nuovo credito, specificando importo, nome, date di concessione ed estinzione, note, e le categorie associati.
* **InsertDebt**: Inserisce un nuovo debito con dettagli quali importo, nome, date di concessione ed estinzione, note, e associazioni di categorie e conti.
* **InsertPrimaryCategory**: Crea una nuova categoria primaria con nome e descrizione, crea anche l’associazione con il profilo.
* **InsertSecondaryCategory**: Aggiunge una categoria secondaria, associata a una categoria primaria esistente, specificando nome e descrizione.
* **InsertTransaction**: Inserisce una nuova transazione, definendo se è una spesa o entrata, l'importo, la data, e le categorie coinvolte.
* **InsertTransactionTemplate**: Crea un nuovo template di transazione che può essere utilizzato per generare transazioni future, includendo dettagli come tipo, importo, descrizione e categorie.

1. **Update Procedures**: Modificano i dati esistenti nelle tabelle per riflettere le nuove informazioni fornite, garantendo che le modifiche siano applicate correttamente ai record appropriati.

* **UpdateBudget**: Aggiorna i dettagli di un budget esistente, inclusi nome, importo massimo, data di inizio, data di fine e categoria primaria associata, basandosi sull'ID specificato.
* **UpdateConto**: Modifica il nome e il saldo di un conto bancario esistente, utilizzando l'ID del conto come riferimento.
* **UpdateCredito**: Aggiorna i dettagli di un credito esistente, compresi importo del credito, nome, date di concessione e estinzione, note aggiuntive, conto associato e categoria primaria, basati sull'ID del credito.
* **UpdateDebito**: Modifica i dettagli di un debito, come l'importo, il nome, le date di concessione ed estinzione, le note, il conto e la categoria primaria associati, identificato dall'ID del debito.
* **UpdatePrimaryCategory**: Aggiorna il nome e la descrizione di una categoria primaria, utilizzando l'ID della categoria come chiave di riferimento.
* **UpdateSecondaryCategory**: Modifica il nome e la descrizione di una categoria secondaria, oltre a aggiornare la categoria primaria associata, basandosi sull'ID della categoria secondaria.
* **UpdateRisparmio**: Aggiorna i dettagli di un obiettivo di risparmio, inclusi l'importo risparmiato, le date di inizio e fine e il conto associato, utilizzando l'ID del risparmio come riferimento.
* **UpdateTemplateTransaction**: Modifica un template di transazione esistente, aggiornando dettagli come il nome del template, se è una spesa o un'entrata, l'importo, il conto associato, le categorie primaria e secondaria e la descrizione, usando l'ID del template come chiave.
* **UpdateTransaction**: Aggiorna una transazione esistente modificando il tipo (spesa/entrata), l'importo, il conto associato, la data della transazione e le categorie (primaria e secondaria) correlate, identificata dall'ID della transazione.

1. **Get and Select Procedures**: Le seguenti stored procedures sono progettate per recuperare informazioni specifiche dal database, facilitando l'accesso e la visualizzazione dei dati in base a vari criteri di selezione:

* **GetAllBudget**: Recupera tutti i record dalla tabella budgetmax, fornendo una panoramica completa di tutti i budget disponibili nel sistema.
* **GetAllCategoriePrimarie**: Estrae tutte le categorie primarie dalla tabella categoriaprimaria, utili per la gestione e l'organizzazione delle transazioni e dei budget.
* **GetAllCategorieSecondarie**: Ottiene tutte le categorie secondarie dalla tabella categoriasecondaria, che sono sottocategorie delle categorie primarie e aiutano a dettagliare ulteriormente la classificazione delle spese.
* **GetAllConti**: Recupera tutti i conti bancari dalla tabella conto, essenziali per tracciare le diverse fonti e destinazioni finanziarie degli utenti.
* **GetAllCrediti**: Raccoglie tutti i dati sui crediti dalla tabella credit, mostrando i prestiti o i crediti attivi attribuiti agli account degli utenti.
* **GetAllDebiti**: Fornisce un elenco di tutti i debiti registrati nella tabella debit, rappresentando gli impegni finanziari degli utenti.
* **GetAllProfili**: Recupera i profili utente dalla tabella Profili, che contengono dettagli degli utenti registrati al sistema.
* **GetAllRisparmi**: Estrae tutti i dati relativi ai risparmi dalla tabella risparmi, fondamentali per gestire gli obiettivi di risparmio e la pianificazione finanziaria a lungo termine.
* **GetAllTransazioni**: Ottiene tutte le transazioni registrate nella tabella transazione, mostrando entrate e uscite per conto degli utenti.
* **GetAllTransazioniTemplate**: Fornisce un elenco di tutti i template di transazioni salvati nella tabella template\_transazioni, utilizzati per creare rapidamente nuove transazioni basate su schemi predefiniti.
* **selectAccountById**: Seleziona un conto specifico dalla tabella conto utilizzando un ID unico, permettendo di accedere rapidamente ai dettagli di un conto specifico.
* **selectBudgetFromID**: Recupera i dettagli di un budget specifico dalla tabella budgetmax basandosi sull'ID, utile per l'analisi e la gestione dei limiti di spesa.
* **selectCategoriaPrimariaById**: Ottiene i dettagli di una categoria primaria specifica dalla tabella categoriaprimaria, essenziale per la modifica o la visualizzazione delle informazioni di categoria.
* **selectCreditFromID**: Estrae i dettagli di un credito specifico dalla tabella credit, permettendo di visualizzare o modificare le condizioni di un credito esistente.
* **selectDebitFromID**: Recupera le informazioni relative a un debito dalla tabella debit usando un ID specifico, importante per la gestione e il controllo dei debiti.
* **selectSavingFromID**: Fornisce i dettagli di un risparmio individuale dalla tabella risparmi, utili per monitorare o aggiornare gli obiettivi di risparmio degli utenti.
* **selectSecondaryCategoryFromID**: Ottiene i dettagli di una categoria secondaria specifica dalla tabella categoriasecondaria, facilitando la gestione delle sottocategorie all'interno del sistema.
* **selectTransactionFromID**: Estrae i dettagli di una transazione specifica dalla tabella transazione, cruciale per la revisione o la correzione delle transazioni passate.

**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.

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() $ $

1. **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.

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 CALL create\_transaction\_on\_debit\_termination(debtCreditID);

ELSE CALL create\_transaction\_on\_credit\_termination(debtCreditID);

END IF;

END LOOP;

CLOSE cur;

END $ $ DELIMITER;

**COODICE SQL COMPLETO:**

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

SET time\_zone = "+00:00";

DELIMITER $$

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$$

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$$

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$$

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$$

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;

    INSERT INTO transazione (

        Is\_Expense,

        Importo,

        IDConto,

        DataTransazione,

        IDCategoriaPrimaria

    ) VALUES (

        1,  -- Debito (uscita)

        debitAmount,

        accountID,

        currentDate,

        primaryCategoryID

    );

    DELETE FROM debit WHERE ID = debitID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteBudget` (IN `p\_ID` INT)   BEGIN

    DELETE FROM budgetmax

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteConto` (IN `p\_ID` INT)   BEGIN

    DELETE FROM conto

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteCredito` (IN `p\_ID` INT)   BEGIN

    DELETE FROM credit

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteDebito` (IN `p\_ID` INT)   BEGIN

    DELETE FROM debit

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeletePrimaryCategory` (IN `p\_ID` INT)   BEGIN

    DELETE FROM categoriaprimaria

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteRisparmio` (IN `p\_ID` INT)   BEGIN

    DELETE FROM risparmi

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteSecondaryCategory` (IN `p\_ID` INT)   BEGIN

    DELETE FROM categoriasecondaria

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteTemplateTransaction` (IN `p\_ID` INT)   BEGIN

    DELETE FROM template\_transazioni

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteTransaction` (IN `p\_ID` INT)   BEGIN

    DELETE FROM transazione

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GenerateFinancialReport` (IN `startDate` DATE, IN `endDate` DATE, IN `transactionType` TINYINT, IN `accountId` INT, IN `primaryCategoryId` INT, IN `secondaryCategoryId` INT)   BEGIN

    SELECT

        t.ID,

        t.Is\_Expense,

        t.Importo,

        t.IDConto,

        t.DataTransazione,

        t.IDCategoriaPrimaria,

        t.IDCategoriaSecondaria

    FROM

        transazione t

    WHERE

        (startDate IS NULL OR t.DataTransazione >= startDate)

        AND (endDate IS NULL OR t.DataTransazione <= endDate)

        AND (transactionType = -1 OR t.Is\_Expense = transactionType)

        AND (accountId IS NULL OR t.IDConto = accountId)

        AND (primaryCategoryId IS NULL OR t.IDCategoriaPrimaria = primaryCategoryId)

        AND (secondaryCategoryId IS NULL OR t.IDCategoriaSecondaria = secondaryCategoryId);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllBudget` ()   BEGIN

    SELECT \* FROM `budgetmax`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllCategoriePrimarie` ()   BEGIN

    SELECT \* FROM `categoriaprimaria`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllCategorieSecondarie` ()   BEGIN

    SELECT \* FROM `categoriasecondaria`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllConti` ()   BEGIN

    SELECT \* FROM `conto`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllCrediti` ()   BEGIN

    SELECT \* FROM `credit`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllDebiti` ()   BEGIN

    SELECT \* FROM `debit`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllProfili` ()   BEGIN

    SELECT \* FROM `Profili`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllRisparmi` ()   BEGIN

    SELECT \* FROM `risparmi`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllTransazioni` ()   BEGIN

    SELECT \* FROM `transazione`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllTransazioniTemplate` ()   BEGIN

    SELECT \* FROM `template\_transazioni`;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetBudgetByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT budgetmax.\* FROM budgetmax

    JOIN categoriaprimaria ON budgetmax.IDPrimaryCategory = categoriaprimaria.ID

    JOIN profili\_categoriaprimaria ON categoriaprimaria.ID = profili\_categoriaprimaria.IDCategoriaPrimaria

    JOIN profili ON profili\_categoriaprimaria.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetCategoriaPrimariaByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT categoriaprimaria.\* FROM categoriaprimaria

    JOIN profili\_categoriaprimaria ON categoriaprimaria.ID = profili\_categoriaprimaria.IDCategoriaPrimaria

    JOIN profili ON profili\_categoriaprimaria.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetCategoriaSecondariaByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT categoriasecondaria.\* FROM categoriasecondaria

    JOIN categoriaprimaria ON categoriasecondaria.IDCategoriaPrimaria = categoriaprimaria.ID

    JOIN profili\_categoriaprimaria ON categoriaprimaria.ID = profili\_categoriaprimaria.IDCategoriaPrimaria

    JOIN profili ON profili\_categoriaprimaria.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetContoByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT conto.\* FROM conto

    JOIN assconti ON conto.ID = assconti.IDConto

    JOIN profili ON assconti.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetCreditiByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT credit.\* FROM credit

    JOIN conto ON credit.IDConto = conto.ID

    JOIN assconti ON conto.ID = assconti.IDConto

    JOIN profili ON assconti.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetDebitiByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT debit.\* FROM debit

    JOIN conto ON debit.IDConto = conto.ID

    JOIN assconti ON conto.ID = assconti.IDConto

    JOIN profili ON assconti.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetRisparmiByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT risparmi.\* FROM risparmi

    JOIN conto ON risparmi.IDConto = conto.ID

    JOIN assconti ON conto.ID = assconti.IDConto

    JOIN profili ON assconti.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetTransazioniByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT transazione.\* FROM transazione

    JOIN conto ON transazione.IDConto = conto.ID

    JOIN assconti ON conto.ID = assconti.IDConto

    JOIN profili ON assconti.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetTransazioniTemplateByEmail` (IN `email` VARCHAR(255))   BEGIN

    SELECT template\_transazioni.\* FROM template\_transazioni

    JOIN conto ON template\_transazioni.IDConto = conto.ID

    JOIN assconti ON conto.ID = assconti.IDConto

    JOIN profili ON assconti.IDProfilo = profili.ID

    WHERE profili.Email = email;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertBudget` (IN `\_NomeBudget` VARCHAR(255), IN `\_ImportoMax` DECIMAL(10,2), IN `\_DataInizio` DATE, IN `\_DataFine` DATE, IN `\_IDPrimaryCategory` INT)   BEGIN

    INSERT INTO budgetmax (NomeBudget, ImportoMax, DataInizio, DataFine, IDPrimaryCategory)

    VALUES (\_NomeBudget, \_ImportoMax, \_DataInizio, \_DataFine, \_IDPrimaryCategory);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertConto` (IN `\_NomeConto` VARCHAR(255), IN `\_Saldo` DECIMAL(10,2), IN `\_IDProfilo` INT)   BEGIN

    DECLARE new\_conto\_id INT;

    INSERT INTO `conto` (`NomeConto`, `Saldo`)

    VALUES (\_NomeConto, \_Saldo);

    SET new\_conto\_id = LAST\_INSERT\_ID();

    INSERT INTO `assconti` (`IDProfilo`, `IDConto`)

    VALUES (\_IDProfilo, new\_conto\_id);

    SELECT new\_conto\_id;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertCredit` (IN `\_ImportoCredito` DECIMAL(10,2), IN `\_NomeImporto` VARCHAR(255), IN `\_DataConcessione` DATE, IN `\_DataEstinsione` DATE, IN `\_Note` TEXT, IN `\_IDConto` INT, IN `\_IDCategoriaPrimaria` INT)   BEGIN

    INSERT INTO credit (ImportoCredito, NomeImporto, DataConcessione, DataEstinsione, Note, IDConto, IDCategoriaPrimaria)

    VALUES (\_ImportoCredito, \_NomeImporto, \_DataConcessione, \_DataEstinsione, \_Note, \_IDConto, \_IDCategoriaPrimaria);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertDebt` (IN `\_ImportoDebito` DECIMAL(10,2), IN `\_NomeImporto` VARCHAR(255), IN `\_DataConcessione` DATE, IN `\_DataEstinsione` DATE, IN `\_Note` TEXT, IN `\_IDConto` INT, IN `\_IDCategoriaPrimaria` INT)   BEGIN

    INSERT INTO debit (ImportoDebito, NomeImporto, DataConcessione, DataEstinsione, Note, IDConto, IDCategoriaPrimaria) VALUES (\_ImportoDebito, \_NomeImporto, \_DataConcessione, \_DataEstinsione, \_Note, \_IDConto, \_IDCategoriaPrimaria);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `insertPrimaryCategory` (IN `\_NomeCategoria` VARCHAR(255), IN `\_DescrizioneCategoria` TEXT, IN `\_IDProfilo` INT)   BEGIN

    DECLARE new\_category\_id INT;

    INSERT INTO `categoriaprimaria` (`NomeCategoria`, `DescrizioneCategoria`)

    VALUES (\_NomeCategoria, \_DescrizioneCategoria);

    SET new\_category\_id = LAST\_INSERT\_ID();

    INSERT INTO `profili\_categoriaprimaria` (`IDProfilo`, `IDCategoriaPrimaria`)

    VALUES (\_IDProfilo, new\_category\_id);

    SELECT new\_category\_id;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertProfile` (IN `\_NomeProfilo` VARCHAR(255), IN `\_Email` VARCHAR(255), IN `\_Password` VARCHAR(255))   BEGIN

    INSERT INTO Profili (NomeProfilo, Email, Password)

    VALUES (\_NomeProfilo, \_Email, \_Password);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertSavings` (IN `\_ImportoRisparmiato` DECIMAL(10,2), IN `\_DataInizio` DATE, IN `\_DataFine` DATE, IN `\_IDConto` INT, IN `\_IDCategoriaPrimaria` INT)   BEGIN

    INSERT INTO risparmi (ImportoRisparmiato, DataInizio, DataFine, IDConto, IDCategoriaPrimaria) VALUES (\_ImportoRisparmiato, \_DataInizio, \_DataFine, \_IDConto, \_IDCategoriaPrimaria);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertSecondaryCategory` (IN `\_IDCategoriaPrimaria` INT, IN `\_NomeCategoria` VARCHAR(255), IN `\_DescrizioneCategoria` TEXT)   BEGIN

    INSERT INTO `categoriasecondaria` (IDCategoriaPrimaria, NomeCategoria, DescrizioneCategoria) VALUES (\_IDCategoriaPrimaria, \_NomeCategoria, \_DescrizioneCategoria);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertTransaction` (IN `\_Is\_Expense` BOOLEAN, IN `\_Importo` DECIMAL(10,2), IN `\_IDConto` INT, IN `\_DataTransazione` DATE, IN `\_IDCategoriaPrimaria` INT, IN `\_IDCategoriaSecondaria` INT)   BEGIN

    INSERT INTO transazione (Is\_Expense, Importo, IDConto, DataTransazione, IDCategoriaPrimaria, IDCategoriaSecondaria) VALUES (\_Is\_Expense, \_Importo, \_IDConto, \_DataTransazione, \_IDCategoriaPrimaria, \_IDCategoriaSecondaria);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertTransactionTemplate` (IN `\_NomeTemplate` VARCHAR(255), IN `\_Is\_Expense` BOOLEAN, IN `\_Importo` DECIMAL(10,2), IN `\_IDConto` INT, IN `\_IDCategoriaPrimaria` INT, IN `\_IDCategoriaSecondaria` INT, IN `\_Descrizione` TEXT)   BEGIN

    INSERT INTO template\_transazioni (NomeTemplate, Is\_Expense, Importo, IDConto, IDCategoriaPrimaria, IDCategoriaSecondaria, Descrizione)

    VALUES (\_NomeTemplate, \_Is\_Expense, \_Importo, \_IDConto, \_IDCategoriaPrimaria, \_IDCategoriaSecondaria, \_Descrizione);

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectAccountById` (IN `accountID` INT)   BEGIN

    SELECT \* FROM conto WHERE ID = accountID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectBudgetFromID` (IN `budgetID` INT)   BEGIN

    SELECT \* FROM budgetmax WHERE ID = budgetID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectCategoriaPrimariaById` (IN `primaryID` INT)   BEGIN

    SELECT \* FROM categoriaprimaria WHERE ID = primaryID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectCreditFromID` (IN `creditID` INT)   BEGIN

    SELECT \* FROM credit WHERE ID = creditID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectDebitFromID` (IN `debitID` INT)   BEGIN

    SELECT \* FROM debit WHERE ID = debitID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectIdContoFromNome` (IN `accountName` VARCHAR(255))   BEGIN

    SELECT ID FROM conto WHERE NomeConto = accountName;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectIDProfileByEmail` (IN `userEmail` VARCHAR(255))   BEGIN

    SELECT ID FROM Profili WHERE Email = userEmail;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectSavingFromID` (IN `savingID` INT)   BEGIN

    SELECT \* FROM risparmi WHERE ID = savingID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectSecondaryCategoryFromID` (IN `secondaryID` INT)   BEGIN

    SELECT \* FROM categoriasecondaria WHERE ID = secondaryID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectSecondaryFromPrimary` (IN `primaryID` INT)   BEGIN

    SELECT \* FROM categoriasecondaria WHERE IDCategoriaPrimaria = primaryID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectTemplateTransactionFromID` (IN `templateID` INT)   BEGIN

    SELECT \* FROM template\_transazioni WHERE ID = templateID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectTransactionFromID` (IN `transactionID` INT)   BEGIN

    SELECT \* FROM transazione WHERE ID = transactionID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `selectUserByEmail` (IN `userEmail` VARCHAR(255))   BEGIN

    SELECT \* FROM Profili WHERE Email = userEmail;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateBudget` (IN `p\_NomeBudget` VARCHAR(255), IN `p\_ImportoMax` DECIMAL(10,2), IN `p\_DataInizio` DATE, IN `p\_DataFine` DATE, IN `p\_IDPrimaryCategory` INT, IN `p\_ID` INT)   BEGIN

    UPDATE budgetmax

    SET NomeBudget = p\_NomeBudget,

        ImportoMax = p\_ImportoMax,

        DataInizio = p\_DataInizio,

        DataFine = p\_DataFine,

        IDPrimaryCategory = p\_IDPrimaryCategory

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateConto` (IN `p\_NomeConto` VARCHAR(255), IN `p\_Saldo` DECIMAL(10,2), IN `p\_ID` INT)   BEGIN

    UPDATE conto

    SET NomeConto = p\_NomeConto,

        Saldo = p\_Saldo

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateCredito` (IN `p\_ImportoCredito` DECIMAL(10,2), IN `p\_NomeImporto` VARCHAR(255), IN `p\_DataConcessione` DATE, IN `p\_DataEstinsione` DATE, IN `p\_Note` TEXT, IN `p\_IDConto` INT, IN `p\_IDCategoriaPrimaria` INT, IN `p\_ID` INT)   BEGIN

    UPDATE credit

    SET ImportoCredito = p\_ImportoCredito,

        NomeImporto = p\_NomeImporto,

        DataConcessione = p\_DataConcessione,

        DataEstinsione = p\_DataEstinsione,

        Note = p\_Note,

        IDConto = p\_IDConto,

        IDCategoriaPrimaria = p\_IDCategoriaPrimaria

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateDebito` (IN `p\_ImportoDebito` DECIMAL(10,2), IN `p\_NomeImporto` VARCHAR(255), IN `p\_DataConcessione` DATE, IN `p\_DataEstinsione` DATE, IN `p\_Note` TEXT, IN `p\_IDConto` INT, IN `p\_IDCategoriaPrimaria` INT, IN `p\_ID` INT)   BEGIN

    UPDATE debit

    SET ImportoDebito = p\_ImportoDebito,

        NomeImporto = p\_NomeImporto,

        DataConcessione = p\_DataConcessione,

        DataEstinsione = p\_DataEstinsione,

        Note = p\_Note,

        IDConto = p\_IDConto,

        IDCategoriaPrimaria = p\_IDCategoriaPrimaria

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdatePrimaryCategory` (IN `p\_NomeCategoria` VARCHAR(255), IN `p\_DescrizioneCategoria` TEXT, IN `p\_ID` INT)   BEGIN

    UPDATE categoriaprimaria

    SET NomeCategoria = p\_NomeCategoria,

        DescrizioneCategoria = p\_DescrizioneCategoria

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateRisparmio` (IN `p\_ImportoRisparmiato` DECIMAL(10,2), IN `p\_DataInizio` DATE, IN `p\_DataFine` DATE, IN `p\_IDConto` INT, IN `p\_ID` INT)   BEGIN

    UPDATE risparmi

    SET ImportoRisparmiato = p\_ImportoRisparmiato,

        DataInizio = p\_DataInizio,

        DataFine = p\_DataFine,

        IDConto = p\_IDConto

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateSecondaryCategory` (IN `p\_NomeCategoria` VARCHAR(255), IN `p\_DescrizioneCategoria` TEXT, IN `p\_IDCategoriaPrimaria` INT, IN `p\_ID` INT)   BEGIN

    UPDATE categoriasecondaria

    SET NomeCategoria = p\_NomeCategoria,

        DescrizioneCategoria = p\_DescrizioneCategoria,

        IDCategoriaPrimaria = p\_IDCategoriaPrimaria

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateTemplateTransaction` (IN `p\_NomeTemplate` VARCHAR(255), IN `p\_Is\_Expense` BOOLEAN, IN `p\_Importo` DECIMAL(10,2), IN `p\_IDConto` INT, IN `p\_IDCategoriaPrimaria` INT, IN `p\_IDCategoriaSecondaria` INT, IN `p\_Descrizione` TEXT, IN `p\_ID` INT)   BEGIN

    UPDATE template\_transazioni

    SET NomeTemplate = p\_NomeTemplate,

        Is\_Expense = p\_Is\_Expense,

        Importo = p\_Importo,

        IDConto = p\_IDConto,

        IDCategoriaPrimaria = p\_IDCategoriaPrimaria,

        IDCategoriaSecondaria = p\_IDCategoriaSecondaria,

        Descrizione = p\_Descrizione

    WHERE ID = p\_ID;

END$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateTransaction` (IN `p\_Is\_Expense` BOOLEAN, IN `p\_Importo` DECIMAL(10,2), IN `p\_IDConto` INT, IN `p\_DataTransazione` DATE, IN `p\_IDCategoriaPrimaria` INT, IN `p\_IDCategoriaSecondaria` INT, IN `p\_ID` INT)   BEGIN

    UPDATE transazione

    SET Is\_Expense = p\_Is\_Expense,

        Importo = p\_Importo,

        IDConto = p\_IDConto,

        DataTransazione = p\_DataTransazione,

        IDCategoriaPrimaria = p\_IDCategoriaPrimaria,

        IDCategoriaSecondaria = p\_IDCategoriaSecondaria

    WHERE ID = p\_ID;

END$$

DELIMITER ;

CREATE TABLE assconti (

  IDProfilo int(11) DEFAULT NULL,

  IDConto int(11) DEFAULT NULL

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

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;

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 ;

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;

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;

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;

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;

DELIMITER $$

CREATE TRIGGER `check\_credit\_expiry` AFTER UPDATE ON `credit` FOR EACH ROW BEGIN

    IF NEW.DataEstinsione = CURDATE() THEN

        CALL create\_transaction\_on\_credit\_termination(NEW.ID);

    END IF;

END

$$

DELIMITER ;

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 ;

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;

DELIMITER $$

CREATE TRIGGER `check\_debit\_expiry` AFTER UPDATE ON `debit` FOR EACH ROW BEGIN

    IF NEW.DataEstinsione = CURDATE() THEN

        CALL create\_transaction\_on\_debit\_termination(NEW.ID);

    END IF;

END

$$

DELIMITER ;

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 ;

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;

CREATE TABLE profili\_categoriaprimaria (

  IDProfilo int(11) NOT NULL,

  IDCategoriaPrimaria int(11) NOT NULL

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

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;

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;

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;

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 ;

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 ;

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 ;

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 $$

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()$$

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

            CALL create\_transaction\_on\_debit\_termination(debtCreditID);

        ELSE

            CALL create\_transaction\_on\_credit\_termination(debtCreditID);

        END IF;

    END LOOP;

    CLOSE cur;

END$$

DELIMITER ;