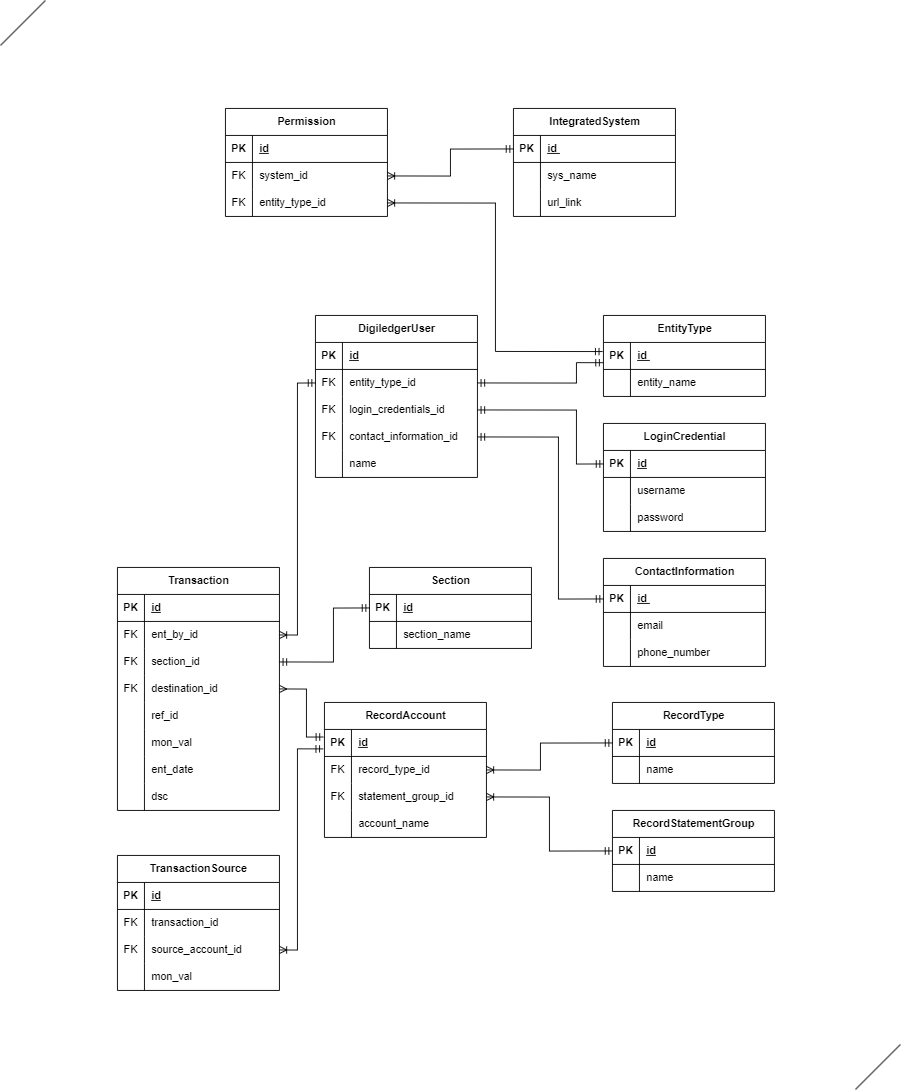
**1. ERD**



**3. SQL Scripts**

**Database and Tables Creation:**

Django automates database table management (creation, modification, deletion) through an abstraction layer controlled in models.py. The code presented below, Django's equivalent of a SQL script, outlines the database structure, specifically for creating tables.

**models.py:**

from django.db import models

*# Create your models here.*

*class* LoginCredential(*models*.*Model*):

    username = models.CharField(*max\_length*=255, *unique*=True)

    password = models.CharField(*max\_length*=255)

*def* \_\_str\_\_(*self*):

        return *self*.username  *# Or any other field that uniquely identifies a credential*

*class* ContactInformation(*models*.*Model*):

    email = models.CharField(*max\_length*=255, *unique*=True)

    phone\_number = models.CharField(*max\_length*=20, *unique*=True)

*def* \_\_str\_\_(*self*):

        return *self*.email *# Or phone\_number*

*class* EntityType(*models*.*Model*):

    entity\_name = models.CharField(*max\_length*=255, *unique*=True)

*def* \_\_str\_\_(*self*):

        return *self*.entity\_name

*class* DigiledgerUser(*models*.*Model*):

    name = models.CharField(*max\_length*=255)

    entity\_type = models.ForeignKey(EntityType, *on\_delete*=models.CASCADE)

    login\_credentials = models.OneToOneField(LoginCredential, *on\_delete*=models.CASCADE, *unique*=True)

    contact\_information = models.OneToOneField(ContactInformation, *on\_delete*=models.CASCADE, *unique*=True)

*def* \_\_str\_\_(*self*):

        return *self*.name *#  Or a combination of fields*

*class* IntegratedSystem(*models*.*Model*):

    sysName = models.CharField(*max\_length*=255)

    url\_link = models.CharField(*max\_length*=255)

*def* \_\_str\_\_(*self*):

        return *self*.sysName *#  Or a combination of fields*

*class* Permission(*models*.*Model*):

    entity\_type = models.ForeignKey(EntityType, *on\_delete*=models.CASCADE)

    system = models.ForeignKey(IntegratedSystem, *on\_delete*=models.CASCADE)

*def* \_\_str\_\_(*self*):

        return *f*"Ent: {*self*.entity\_type.entity\_name} Sys: {*self*.system.sysName}"

*class* Section(*models*.*Model*):

    section\_name = models.CharField(*max\_length*=255, *unique*=True)

*def* \_\_str\_\_(*self*):

        return *self*.section\_name

*class* RecordType(*models*.*Model*):

    name = models.CharField(*max\_length*=255, *unique*=True)

*def* \_\_str\_\_(*self*):

        return *self*.name

*class* RecordStatementGroup(*models*.*Model*):

    name = models.CharField(*max\_length*=255, *unique*=True)

*def* \_\_str\_\_(*self*):

        return *self*.name

*class* RecordAccount(*models*.*Model*):

    account\_name = models.CharField(*max\_length*=255, *unique*=True)

    record\_type = models.ForeignKey(RecordType, *on\_delete*=models.CASCADE)

    statement\_group = models.ForeignKey(RecordStatementGroup, *on\_delete*=models.CASCADE)

*def* \_\_str\_\_(*self*):

        return *self*.account\_name

*class* Transaction(*models*.*Model*):

    ref\_id = models.CharField(*max\_length*=255, *unique*=True)

    mon\_val = models.DecimalField(*max\_digits*=10, *decimal\_places*=2)

    ent\_date = models.DateTimeField()

    dsc = models.CharField(*max\_length*=255)

    destination = models.ForeignKey(RecordAccount, *related\_name*='destination\_transactions', *on\_delete*=models.CASCADE)

    section = models.ForeignKey(Section, *on\_delete*=models.CASCADE)

    ent\_by = models.ForeignKey(DigiledgerUser, *on\_delete*=models.SET\_NULL, *null*=True, *blank*=True)

*def* \_\_str\_\_(*self*):

        return *f*"Txn {*self*.ref\_id} - {*self*.mon\_val} on {*self*.ent\_date.strftime('%Y-%m-%d %H:%M:%S')}" *#Added time*

*class* TransactionSource(*models*.*Model*):

    mon\_val = models.DecimalField(*max\_digits*=10, *decimal\_places*=2)

    transaction = models.ForeignKey(Transaction, *related\_name*='sources', *on\_delete*=models.CASCADE)

    source\_account = models.ForeignKey(RecordAccount, *on\_delete*=models.CASCADE)

*class* Meta:

        unique\_together = ('transaction', 'source\_account')  *# Prevent duplicate sources per transaction*

*def* \_\_str\_\_(*self*):

        return *f*"{*self*.transaction.ref\_id} - Source: {*self*.source\_account}"

**Initial Sample Data Insertion:**

-- LoginCredential

INSERT INTO digiledger\_logincredential (username, password) VALUES

('arvey', 'arveypogi'),

('kenneth', 'amboya'),

('cbrown', 'mng123'),

('dlee', 'owner456'),

('ewilliams', 'cashier022');

-- ContactInformation

INSERT INTO digiledger\_contactinformation (email, phone\_number) VALUES

('email@site.com', '09123456789'),

('kennethamboya101@gmail.com', '+639457818690'),

('admin@server.org', '09475829163'),

('guest@nowhere.dev', '09103376049'),

('employee5@work.com', '09698012574');

-- EntityType

INSERT INTO digiledger\_entitytype (entity\_name) VALUES

('Owner'),

('Manager'),

('Cashier'),

('Admin');

-- DigiledgerUser

INSERT INTO digiledger\_digiledgeruser (name, entity\_type\_id, login\_credentials\_id, contact\_information\_id) VALUES

('arvey', 1, 1, 1),

('kenneth', 2, 2, 2),

('Charlie Brown', 1, 3, 3),

('Diana Lee', 3, 4, 4),

('Eve Williams', 5, 5, 5);

-- Permission

INSERT INTO digiledger\_permission (perm\_name, entity\_type\_id) VALUES

(1, 1),

(1, 2),

(3, 1),

(3, 2),

(3, 3),

(4, 3),

(4, 2),

(4, 1),

(1, 3);

-- Section

INSERT INTO digiledger\_section (section\_name) VALUES

('General');

-- RecordType

INSERT INTO digiledger\_recordtype (name) VALUES

('Asset'),

('Equity'),

('Expenses'),

('Liabilities'),

('Revenue');

-- RecordStatementGroup

INSERT INTO digiledger\_recordstatementgroup (name) VALUES

('Balance Sheet'),

('Income Statement'),

('Cash Flow'),

('Others');

-- RecordAccount

INSERT INTO digiledger\_recordaccount (account\_name, record\_type\_id, statement\_group\_id) VALUES

('Cash', 3, 1),

('Accounts Receivable', 3, 1),

('Inventory', 3, 1),

('Prepaid Expenses', 3, 1),

('Land, Buildings, and Equipment', 3, 1),

('Accounts Payable', 4, 1),

('Notes Payable', 4, 1),

('Wages Payable', 4, 1),

('Common Stock', 5, 1),

('Retained Earnings', 5,1),

('Sales Revenue', 6, 2),

('Service Fee', 6, 2),

('Salaries Expense', 2, 2),

('Rent Expense', 2, 2),

('Cost of Goods Sold', 2, 2);

-- Transaction

INSERT INTO digiledger\_transaction (ref\_id, mon\_val, ent\_date, ent\_by\_id, section\_id, destination\_id, dsc) VALUES

(1, 5000.00, '2025-05-01 13:35:58.000000', 1, 1, 13, 'Owner''s investment'),

(2, 1000.00, '2025-05-06 13:37:12.000000', 1, 1, 15, 'Purchased Supplies on Credit'),

(3, 500.00, '2025-05-06 14:23:10.000000', 1, 1, 26, 'paid for a month of rent'),

(4, 800.00, '2025-05-06 14:25:10.000000', 1, 1, 13, 'Received cash for services'),

(5, 1800.00, '2025-05-07 14:26:26.000000', 1, 1, 13, 'Received cash for craft sales'),

(6, 1200.00, '2025-05-07 14:27:11.000000', 1, 1, 28, 'Cost of crafts sold'),

(7, 500.00, '2025-05-09 14:38:08.000000', 1, 1, 17, 'Partial cash payment for equipment'),

(8, 1500.00, '2025-05-09 14:40:20.000000', 1, 1, 17, 'Long-term debt for equipment'),

(9, 300.00, '2025-05-10 14:41:55.000000', 1, 1, 18, 'Partial payment to Crafty Suppliers'),

(10, 600.00, '2025-05-11 14:42:32.000000', 1, 1, 14, 'Services provided on credit'),

(11, 800.00, '2025-05-11 14:43:11.000000', 1, 1, 25, 'Paid salaries for first half of June'),

(12, 400.00, '2025-05-11 14:43:50.000000', 1, 1, 13, 'Received cash from client on account'),

(13, 150.00, '2025-05-12 14:46:32.000000', 1, 1, 16, 'Paid for three months of advertising'),

(14, 50.00, '2025-05-15 14:48:20.000000', 1, 1, 20, 'Wages incurred but not yet paid'),

(15, 1500.00, '2025-05-15 14:54:47.000000', 1, 1, 14, 'Crafts sold on credit'),

(16, 900.00, '2025-05-16 14:55:34.000000', 1, 1, 28, 'Cost of crafts sold on credit'),

(17, 1100.00, '2025-05-16 15:13:20.000000', 1, 1, 15, 'restock');

-- TransactionSource

INSERT INTO digiledger\_transactionsource (mon\_val, transaction\_id, source\_account\_id) VALUES

(5000.00, 26, 21),

(1000.00, 27, 18),

(500.00, 30, 13),

(800.00, 31, 24),

(1800.00, 33, 23),

(1200.00, 34, 15),

(500.00, 36, 13),

(1500.00, 37, 19),

(300.00, 38, 13),

(600.00, 39, 24),

(800.00, 41, 13),

(400.00, 41, 14),

(150.00, 42, 13),

(50.00, 43, 25),

(1500.00, 44, 23),

(900.00, 45, 15),

(1100.00, 47, 13);

INSERT INTO digiledger\_integratedsystems (sysName, url\_link) VALUES

("Point of Sale", "digiledger:dashboard"),

("Digiledger", "digiledger:dashboard"),

("Expense Management System", "digiledger:dashboard");

**Advanced SQL:**

The *transaction.atomic()* in Django acts as the system’s TCL, ensuring all database operations within its block succeed together or fail entirely (rollback).

below are two functions found within the views.py that uses transaction.atomic():

*def* new\_entry(*request*):

    if request.method == 'POST':

        ref\_id = request.POST.get('ref\_id', '').strip()

        mon\_val\_str = request.POST.get('mon\_val')

        section\_name = request.POST.get('section')

        destination\_account\_name = request.POST.get('destination')

        description = request.POST.get('dsc\_textarea', '').strip()

*# --- Access Multi-Value Source Fields ---*

        source\_mon\_vals\_str = request.POST.getlist('src\_mon\_val[]')

        source\_account\_names = request.POST.getlist('src\_account[]')

*# --- Basic Validation & Type Conversion ---*

        if not all([ref\_id, mon\_val\_str, section\_name, destination\_account\_name, source\_mon\_vals\_str, source\_account\_names]):

*# Handle missing required fields (return an error response or message)*

            return HttpResponseBadRequest("Missing required form fields.")

        if len(source\_mon\_vals\_str) != len(source\_account\_names):

            return HttpResponseBadRequest("Mismatch between number of source values and source accounts.")

        try:

*# Convert monetary values to Decimal*

            total\_mon\_val = Decimal(mon\_val\_str)

            source\_mon\_vals = [Decimal(val) for val in source\_mon\_vals\_str]

*# Validate that the sum of source values equals the total monetary value*

            if sum(source\_mon\_vals) != total\_mon\_val:

                return HttpResponseBadRequest("Sum of source monetary values does not match the total entry monetary value.")

        except InvalidOperation:

            return HttpResponseBadRequest("Invalid monetary value format. Please enter numbers.")

*# --- Database Operations (within an atomic transaction) ---*

        try:

            with transaction.atomic(): *# Ensures all operations succeed or fail together*

*# 1. Fetch related objects (Section, Destination Account)*

*# Assumes they exist. Use get\_or\_create or try-except for more robustness.*

                section\_obj = Section.objects.get(*section\_name*=section\_name)

                destination\_account\_obj = RecordAccount.objects.get(*account\_name*=destination\_account\_name)

                current\_user\_id = request.session.get('current\_user\_id')

                current\_user = DigiledgerUser.objects.get(*id*=current\_user\_id)

*# 2. Create the main Transaction instance*

                new\_transaction = Transaction.objects.create(

*ref\_id*=ref\_id,

*mon\_val*=total\_mon\_val,

*ent\_date*=timezone.now(), *# Use current time*

*dsc*=description,

*destination*=destination\_account\_obj,

*section*=section\_obj,

*ent\_by*=current\_user *# Assign the logged-in user*

                )

*# 3. Create the related TransactionSource instances*

                for i in range(len(source\_mon\_vals)):

                    src\_val = source\_mon\_vals[i]

                    src\_acc\_name = source\_account\_names[i]

*# Fetch the source RecordAccount object*

                    source\_account\_obj = RecordAccount.objects.get(*account\_name*=src\_acc\_name)

                    TransactionSource.objects.create(

*mon\_val*=src\_val,

*transaction*=new\_transaction,  *# Link to the newly created transaction*

*source\_account*=source\_account\_obj

                    )

*# --- Success ---*

*# Redirect to a success page or display a success message*

*# messages.success(request, f"Transaction {ref\_id} created successfully!")*

        except Section.DoesNotExist:

            return HttpResponseBadRequest(*f*"Error: Section '{section\_name}' not found.")

        except RecordAccount.DoesNotExist as e:

*# Identify if it was the destination or a source account*

*# This requires a bit more logic, but a simple message can work initially:*

            return HttpResponseBadRequest(*f*"Error: A specified Record Account was not found.")

        except *Exception* as e:

*# Catch other potential errors during database operations*

*# Log the error e*

            print(*f*"An unexpected error occurred: {e}") *# For debugging*

            return HttpResponseServerError("An unexpected error occurred while saving the transaction.")

        else:

            current\_user\_id = request.session.get('current\_user\_id')

            current\_user = DigiledgerUser.objects.get(*id*=current\_user\_id)

            context = {

                'CurrentUser': current\_user,

                'Sections': Section.objects.all(),

                'Accounts': RecordAccount.objects.all(),

                'Transactions': get\_transactions(),

            }

            return redirect('digiledger:dashboard') *# Replace with your actual success URL*

    else:

        current\_user\_id = request.session.get('current\_user\_id')

        current\_user = DigiledgerUser.objects.get(*id*=current\_user\_id)

        context = {

            'CurrentUser': current\_user,

            'Sections': Section.objects.all(),

            'Accounts': RecordAccount.objects.all(),

            'Transactions': get\_transactions(),

        }

        return render(request, 'digiledger/entNew.html', *context*=context)

*def* signUp(*request*):

    if request.method == "POST":

        user\_Name = request.POST.get("user\_Name")

        ent\_Type = request.POST.get("ent\_Type")

        username = request.POST.get("username").strip()

        password = request.POST.get("password").strip()

        email = request.POST.get("email").strip()

        phoneNumber = request.POST.get("phoneNumber").strip()

        if not all([user\_Name, ent\_Type, username, password, email, phoneNumber]):

*# Handle missing required fields (return an error response or message)*

            return HttpResponseBadRequest("Missing required form fields.")

        try:

            with transaction.atomic(): *# Ensures all operations succeed or fail together*

                entity\_obj = EntityType.objects.get(*entity\_name*=ent\_Type)

                newUser = DigiledgerUser.objects.create(

*name*=user\_Name,

*entity\_type*=entity\_obj,

*login\_credentials*=LoginCredential.objects.create(

*username*=username,

*password*=password

                    ),

*contact\_information*=ContactInformation.objects.create(

*email*=email,

*phone\_number*=phoneNumber

                    )

                )

        except Section.DoesNotExist:

            return HttpResponseBadRequest(*f*"Error: Section '{section\_name}' not found.")

        else:

            return render(request, 'digiledger/newUser.html', {'EntityType': EntityType.objects.all()})

    else:

        return render(request, 'digiledger/newUser.html', {'EntityType': EntityType.objects.all()})

**4.DigiLedger Database Back-up File**

* Export the final version of your database.
* Using MySQL Workbench

1. Open MySQL Workbench app.
2. Go to server -> Data Export.

A screenshot of a computer

AI-generated content may be incorrect.

1. A screenshot of a computer

   AI-generated content may be incorrect.Select the file you want to export.
2. Click start export.

A screenshot of a computer

AI-generated content may be incorrect.

* Ensure that it can be imported or restored into a local or remote SQL environment.

1. Open MySQL Workbench app.
2. Just like the first step, go to server, instead of data export, select data import.

A screenshot of a computer

AI-generated content may be incorrect.

1. Choose the SQL file you want to import.

A screenshot of a computer

AI-generated content may be incorrect.

1. Select default target schema.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Click start import.

* Here’s the imported schema.

A screenshot of a computer

AI-generated content may be incorrect.

**5. Explanation of Normalization**

**1. (0NF):**

Here's the structure of the DigiledgerData table:

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| user\_name | VARCHAR(255) |
| entity\_type | VARCHAR(255) |
| login\_username | VARCHAR(255) |
| login\_password | VARCHAR(255) |
| Email | VARCHAR(255) |
| phone\_number | VARCHAR(20) |
| permission\_name | VARCHAR(255) |
| section\_name | VARCHAR(255) |
| record\_type | VARCHAR(255) |
| record\_account\_name | VARCHAR(255) |
| record\_statement\_group | VARCHAR(255) |
| transaction\_ref\_id | VARCHAR(255) |
| transaction\_mon\_val | DECIMAL(10,2) |
| transaction\_ent\_date | DATETIME |
| transaction\_dsc | VARCHAR(255) |
| transaction\_destination\_account | VARCHAR(255) |
| transaction\_ent\_by\_name | VARCHAR(255) |
| source\_account\_name\_1 | VARCHAR(255) |
| source\_mon\_val\_1 | DECIMAL(10,2) |
| source\_account\_name\_2 | VARCHAR(255) |
| source\_mon\_val\_2 | DECIMAL(10,2) |
| source\_account\_name\_3 | VARCHAR(255) |
| source\_mon\_val\_3 | DECIMAL(10,2) |

**2. First Normal Form (1NF):**

Reduced the repeating groups and created a separate table, TransactionSource.

**Simplified Tables (1NF):**

**Transaction Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| transaction\_ref\_id | VARCHAR(255) |
| transaction\_mon\_val | DECIMAL(10, 2) |
| transaction\_ent\_date | DATETIME |
| transaction\_dsc | VARCHAR(255) |
| transaction\_destination\_account | VARCHAR(255) |
| transaction\_ent\_by\_name | VARCHAR(255) |

**Transaction Source Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| transaction\_ref\_id | VARCHAR(255) |
| source\_account\_name | VARCHAR(255) |
| source\_mon\_val | DECIMAL(10, 2) |

**3. Second Normal Form (2NF):**

The tables are then split so that each piece of information depends fully on the main identifier of its own table. Separate the tables and create LoginCredential and ContactInformation. New tables EntityType, Section, RecordType and RecordStatementGroup are created. The table DigiledgerUser is created to hold user related information

**Simplified Tables (2NF):**

**LoginCredential Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| login\_username | VARCHAR(255) |
| login\_password | VARCHAR(255) |

**ContactInformation Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| email | VARCHAR(255) |
| phone\_number | VARCHAR(20) |

**EntityType Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| entity\_type | VARCHAR(255) |

**DigiledgerUser Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| user\_name | VARCHAR(255) |
| login\_username | VARCHAR(255) |
| email | VARCHAR(255) |
| phone\_number | VARCHAR(20) |
| entity\_type | VARCHAR(255) |

Section Table

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| section\_name | VARCHAR(255) |

RecordType Table

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| record\_type | VARCHAR(255) |

RecordStatementGroup Table

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| record\_statement\_group | VARCHAR(255) |

Transaction Table

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| transaction\_ref\_id | VARCHAR(255) |
| transaction\_mon\_val | DECIMAL(10,2) |
| transaction\_ent\_date | DATETIME |
| transaction\_dsc | VARCHAR(255) |
| transaction\_destination\_account | VARCHAR(255) |
| transaction\_ent\_by\_name | VARCHAR(255) |
| section\_name | VARCHAR(255) |

Transaction Source Table

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| transaction\_ref\_id | VARCHAR(255) |
| source\_account\_name | VARCHAR(255) |
| source\_mon\_val | DECIMAL(10,2) |

**4. Third Normal Form (3NF):**

The design prevents any non-key information from being dependent on other non-key information within the same table. Your current structure of separate tables seems to achieve this.

**Simplified Tables (3NF):**

**LoginCredential Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| login\_username | VARCHAR(255) |
| login\_password | VARCHAR(255) |

**ContactInformation Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| email | VARCHAR(255) |
| phone\_number | VARCHAR(20) |

**EntityType Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| entity\_type | VARCHAR(255) |

**DigiledgerUser Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| user\_name | VARCHAR(255) |
| login\_username | VARCHAR(255) |
| email | VARCHAR(255) |
| phone\_number | VARCHAR(20) |
| entity\_type | VARCHAR(255) |

**Section Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| section\_name | VARCHAR(255) |

**RecordType Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| record\_type | VARCHAR(255) |

**RecordStatementGroup Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| record\_statement\_group | VARCHAR(255) |

**Transaction Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| transaction\_ref\_id | VARCHAR(255) |
| transaction\_mon\_val | DECIMAL(10,2) |
| transaction\_ent\_date | DATETIME |
| transaction\_dsc | VARCHAR(255) |
| transaction\_destination\_account | VARCHAR(255) |
| transaction\_ent\_by\_name | VARCHAR(255) |
| section\_name | VARCHAR(255) |

**Transaction Source Table**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| transaction\_ref\_id | VARCHAR(255) |
| source\_account\_name | VARCHAR(255) |
| source\_mon\_val | DECIMAL(10,2) |

**6. Security Features Implemented:**

- SQL Intjection Protection is not possible as django uses an abstraction layer to get and send data to the Database.  
sample of code that uses the abstraction layer to send data from db

with transaction.atomic(): *# Ensures all operations succeed or fail together*

*# 1. Fetch related objects (Section, Destination Account)*

*# Assumes they exist. Use get\_or\_create or try-except for more robustness.*

                section\_obj = Section.objects.get(*section\_name*=section\_name)

                destination\_account\_obj = RecordAccount.objects.get(*account\_name*=destination\_account\_name)

                current\_user\_id = request.session.get('current\_user\_id')

                current\_user = DigiledgerUser.objects.get(*id*=current\_user\_id)

*# 2. Create the main Transaction instance*

                new\_transaction = Transaction.objects.create(

*ref\_id*=ref\_id,

*mon\_val*=total\_mon\_val,

*ent\_date*=timezone.now(), *# Use current time*

*dsc*=description,

*destination*=destination\_account\_obj,

*section*=section\_obj,

*ent\_by*=current\_user *# Assign the logged-in user*

                )

*# 3. Create the related TransactionSource instances*

                for i in range(len(source\_mon\_vals)):

                    src\_val = source\_mon\_vals[i]

                    src\_acc\_name = source\_account\_names[i]

*# Fetch the source RecordAccount object*

                    source\_account\_obj = RecordAccount.objects.get(*account\_name*=src\_acc\_name)

                    TransactionSource.objects.create(

*mon\_val*=src\_val,

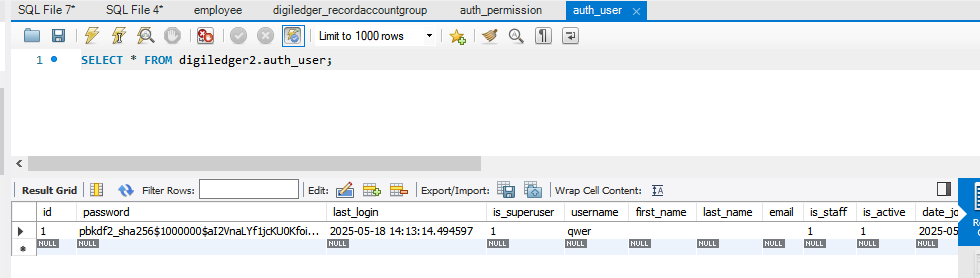
*transaction*=new\_transaction,  *# Link to the newly created transaction*

*source\_account*=source\_account\_obj

                    )

Password Hashing

-Django also Hashes passwords at default



Datavalidation

-Django also uses the one abstraction layer to validate data

-backend

try:

*# code from sql injection protection par*

        except Section.DoesNotExist:

            return HttpResponseBadRequest(*f*"Error: Section '{section\_name}' not found.")

        except RecordAccount.DoesNotExist as e:

*# Identify if it was the destination or a source account*

*# This requires a bit more logic, but a simple message can work initially:*

            return HttpResponseBadRequest(*f*"Error: A specified Record Account was not found.")

        except *Exception* as e:

*# Catch other potential errors during database operations*

*# Log the error e*

            print(*f*"An unexpected error occurred: {e}") *# For debugging*

            return HttpResponseServerError("An unexpected error occurred while saving the transaction.")

        else:

            current\_user\_id = request.session.get('current\_user\_id')

            current\_user = DigiledgerUser.objects.get(*id*=current\_user\_id)

            context = {

                'CurrentUser': current\_user,

                'Sections': Section.objects.all(),

                'Accounts': RecordAccount.objects.all(),

                'Transactions': get\_transactions(),

            }

            return redirect('digiledger:dashboard') *# Replace with your actual success URL*

    else:

        current\_user\_id = request.session.get('current\_user\_id')

        current\_user = DigiledgerUser.objects.get(*id*=current\_user\_id)

        context = {

            'CurrentUser': current\_user,

            'Sections': Section.objects.all(),

            'Accounts': RecordAccount.objects.all(),

            'Transactions': get\_transactions(),

        }

        return render(request, 'digiledger/entNew.html', *context*=context)