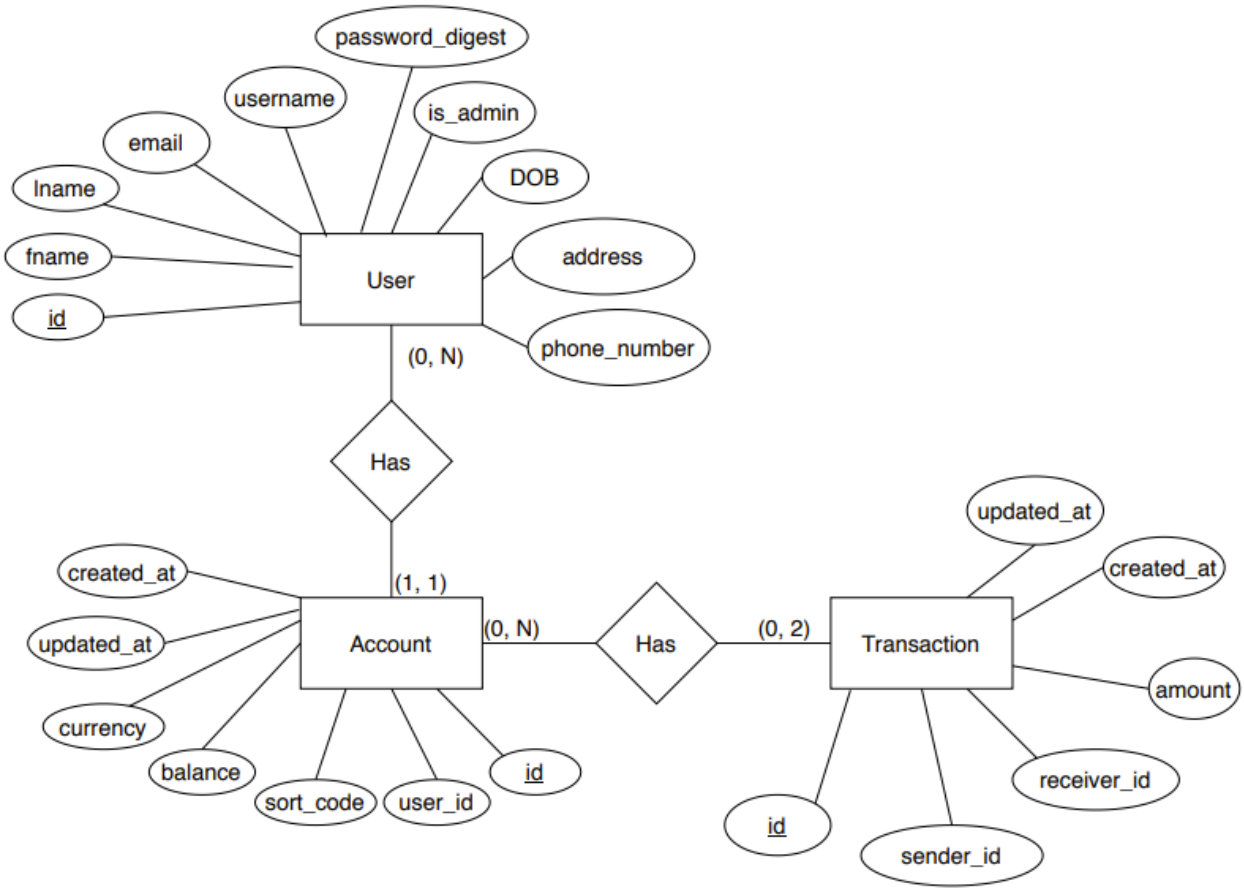
Database ER Diagram



The relationships are as followed:

1. A user can have 0 or many accounts
2. An account must belong to 1 user.
3. An account can have 0 or many transactions.
4. A transaction can belong to 0, 1 or 2 accounts. This is because a user can send a transaction to a fake or nonexistent account to forge/fake transactions and transaction histories.

Ruby on Rails Representation of the Database

There are 3 models in the application:

1. User
2. Transaction
3. Account

1. The user model has the following relationships:

```
has_many :accounts
has_many :sent_transactions, through: :accounts
has_many :received_transactions, through: :accounts
```

This clearly states that each user can have many accounts. It can also have many sent and received transactions "through" the accounts model. This allows for statements as follows:

Assume there is a user called Alice, accessing Alice's accounts can be done simply using alice.accounts. To retrieve all the sent transactions Alice has, this can be achieved through alice.sent_transactions. Similarly, for received transactions, alice.received_transactions. The model relationships allow for these extremely useful statements.

2. The transaction model has the following relationships:

```
belongs_to :sender, :class_name => 'Account',  optional: :true
belongs_to :receiver, :class_name => 'Account', optional: :true
```

Each transaction belongs to 0, 1 or 2 different accounts, a sender, and a receiver. They 'optionally' belong to an account because fake transactions can belong to 0, 1 or 2 accounts (sender and receiver). This is because when generating fake transactions, for the sake of a scam baiting website, it is not needed that one, two, or any, account map to an individual transaction. This allows the admin or site owner to freely create fake transactions and transaction histories for an individual account without needing a significant number of accounts pre-populated. Given an arbitrary transaction "transaction_one", to access the sender account one can write account_one.sender, likewise for receiver account_one.receiver.

3. The account model has the following relationships:

```
belongs_to :user, :foreign_key => :user_id
has_many :sent_transactions, :foreign_key => :sender_id, :class_name => 'Transaction'
has_many :received_transactions, :foreign_key => :receiver_id, :class_name => 'Transaction'
```

Each account belongs to one user, with the user_id being the foreign key. An account also has many sent transactions and received transactions, which is a relationship to the transaction model, using the foreign keys sender_id and receiver_id. Once again, given an arbitrary account "account_one", to access the user of that account, one can write "account_one.user". To access sent transactions it is enough to write account_one.sent_transactions. Similarly for received transactions account_one.received_transactions.

Ruby allows for these extremely powerful commands.