Database Design & Development

Final Project

Team: *PM12*
Max Kornyev, Alex Nelson, Annie Spahn

Relational Model:

Primary keys are bolded, while foreign keys are underlined.

Users(**u_id**, venmo_balance)

Friends(**users.u_id1, users.u_id2**)

Transfers(**users.u_id, bankaccounts.account_number, timestamp**, amount)

BankAccounts(**account_number**, balance, overage_percent)

Businesses(**users.u_id**, biz_name)

Individuals(**users.u_id**, username, fname, lname, email)

Transactions(**transaction_ id,** users.u_id, users.t_id, timestamp)

Comments(**transactions.transaction_id**, text)

Payments(**transactions.transaction_id,** amount)

Requests(**transactions.transaction_id** amount, approved)

Description:

For each entity in our original model, we developed an equivalent relation. In our relational model, many associations get absorbed into other entities in the table. This includes the *transaction_id* attribute into the transaction table. Additionally, associative entities like *text* and *amount* also get absorbed into entirely new entities as a result of many to many relationships. Inheritance, with respect to Individuals and Businesses, passes the primary *u_id* from Users to these two entities. Note, the Friends relation is mirrored; ie. if *(i, j)* is an entry, *(j, i)* is also an entry.

The Transfers and Individuals relations are normalized in the following section. Their final relational models take the following form:

Transfers:

Transfers(**u_id**, **timestamp**, amount)

UserAccounts(**u_id**, account_number)

Individuals:

Individuals(**u_id**, username)

UserInfo(**username**, fname, lname, email)

Functional Dependencies:

Functional dependencies and the current normal form for each developed relation are listed below. Some assumptions we made are: Users only associate one bank account with their Venmo at a time.

Relation	Dependencies	Notes
Users: BCNF		
	u_id -> venmo_balance	
Friends: BCNF		
	No FD's	-- Full-Keyed Relation
Transfers: 1NF		
	u_id -> account_number	
	account_number -> u_id	-- Accounts are systematically unique
	u_id, account_number, timestamp -> amount	
BankAccounts: BCNF		
	account_number -> balance, overage_percent	
Businesses: BCNF		
	u_id -> biz_name	
	biz_name -> u_id	-- Biz names are systematically unique
Individuals: 2NF		
	u_id -> username, fname, lname, email	
	username -> fname, lname, email	-- Usernames are systematically unique
	username -> u_id	
Transactions: BCNF		
	transaction_ id -> u_id, t_id, timestamp	-- Unique id's identify transactions
Comments: BCNF		
	transaction_id -> text	
Payments: BCNF		
	transaction_id -> amount, approved	
Requests: BCNF		
	transaction_id -> amount, approved	
PaymentUpdates: BCNF		
	transaction_id -> amount	

Normalization:

All but two of our original relations were found to be in *Boyce-Codd Normal Form*. Below, we include explanations for why each relation is in its according normal form, with full-keyed relations omitted. Full-keyed relations are already in BCNF, as there are no non-trivial functional dependencies (FD's) on the super keys of our candidate key (the candidate key *is* the super key in the full-keyed case).

Users: The closure of the user id and it's super keys find the entire relation. No bad FD's.

Friends: *Full-keyed.*

Transfers: 1NF because *u_id -> account_number* is a bad transitive dependency. The normalized relation includes two tables: R1, linking u_id and account_number, and R2, linking u_id, timestamp, and amount. *With letters mapping back to the original Transfer relation*, it was normalized in Fig 1:

Fig 1, Transfers

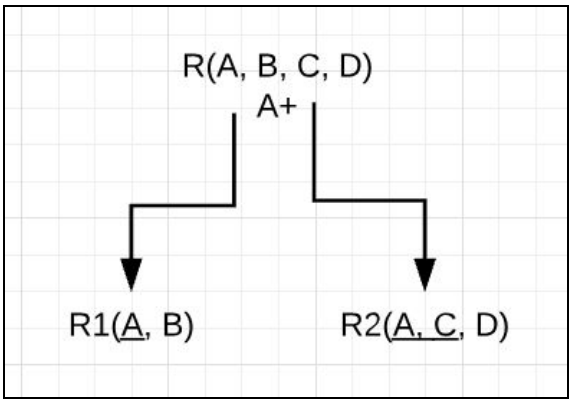
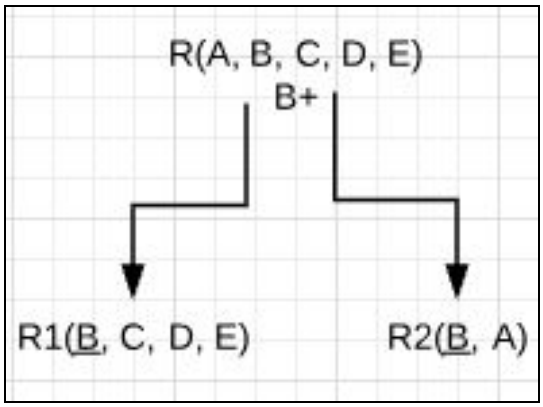


Fig 2, Individuals



Individuals: 2NF because a non-prime username determines non-prime attributes. Normalized in Fig 2. Considering more functional dependencies, like other combinations of non-primes, made the relation more convoluted. Using the specific relations in Fig 2 is a design choice. R1 links username, fname, lname, and email, while R2 links u_id and username.

BankAccounts: The account number and it's super keys find the entire relation. No bad FD's.

Businesses: The user id and it's super keys find the entire relation. No bad FD's.

Transactions: The transaction id and it's super keys find the entire relation. No bad FD's.

Comments: The transaction id uniquely identifies all rows in the relation. No bad FD's.

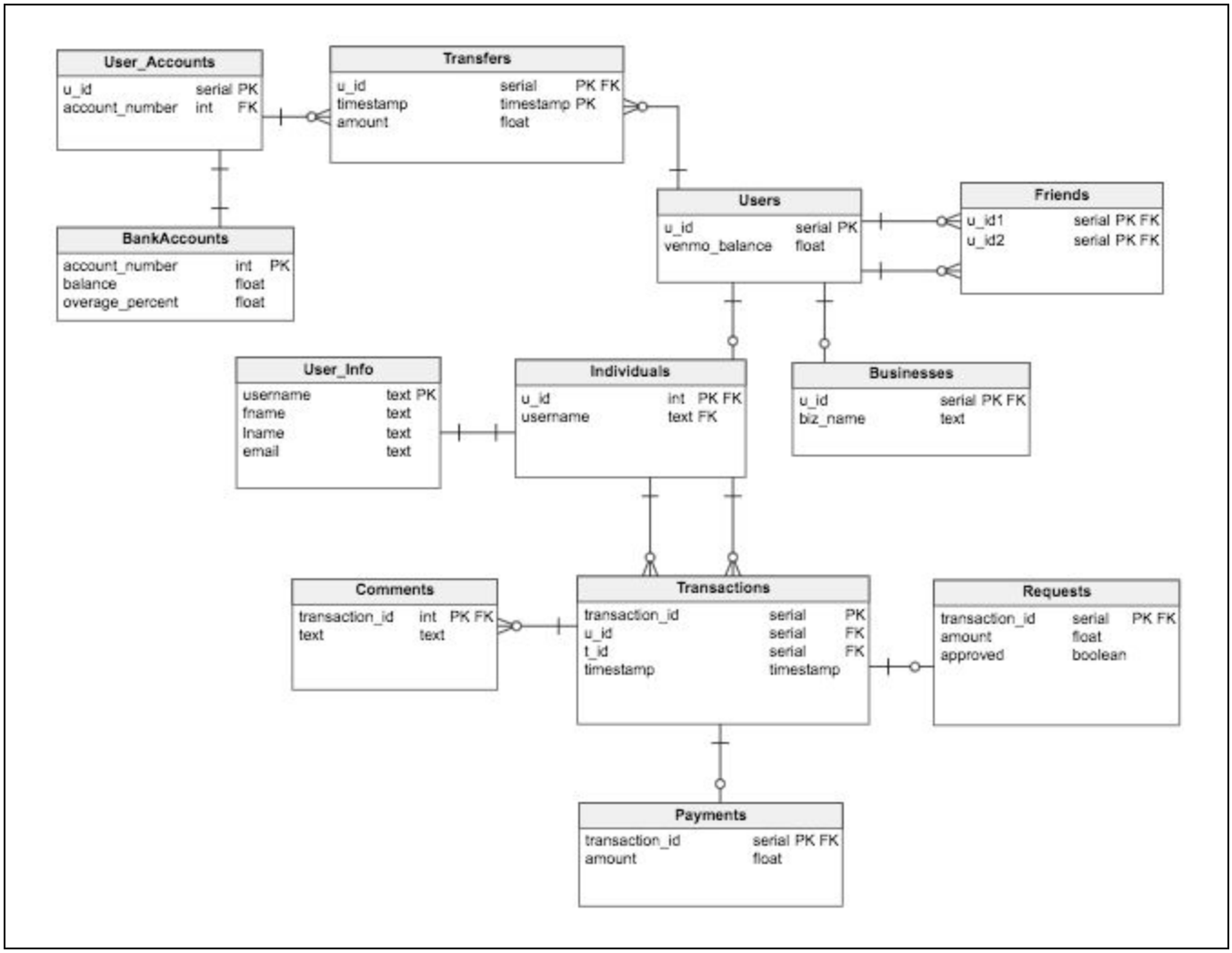
Payments: The transaction id uniquely identifies all rows in the relation. No bad FD's.

Requests: The transaction id uniquely identifies all rows in the relation. No bad FD's.

Physical Model:

We realized our final Relational Model using Vertabelo, and included all specific data types and normalized relations. The model is shown below. The descriptions of our queries in the following section better clarify how these entities work together.

Fig 3, Vertabelo Physical Model



Queries:

Our database has some fundamental triggers, declared at the bottom of the *show_all.sql* file, that help us create each user story. The descriptions of these triggers, along with each user story, are described below. Running the *initialize.sql* file will create all tables, populate the database, and create some essential triggers. After running a user story (*.py* file) appropriate changes in the database can be seen by running the show all script.

Triggers

Payment Trigger:

Updates the sender and receiver balances on an insert into the Payments table.

Request Trigger:

Once the new.approved boolean becomes *True* on an insert or update into the table, Payments are updated.

Stories

Simple 1. As a user, pay and request other users.

This story requires an insert into the transaction table and payments table for direct payments. Otherwise, requests are sent to the requests table after being inserted into Transactions.

Simple 2. See venmo withdrawals and deposits.

Reflects transactions in the database.

Simple 3. Add a user comment to a transaction.

Updates the comments table on an existing transaction.

Simple 4. As a User, add friends.

This updates the the Friends table with mirrored values.

Simple 5. As a business, receive a payment from a user.

Update the venmo_balance of a business in the users table.

Simple 6. As a user, updates a username.

Updates the UserInfo table given an input.

Complex 1. As a bank, see all overdrawn accounts and the calculated overage charge for the account.

Calculates a charge amount from the overdrawn bank accounts.

Complex 3. As a user, see my spending over a specific window of time.

Sums all payments to the a specific payments and returns average over indicated timespan.

Complex 4. Return the number of users that are friends that paid a specific business.

Crates a view with friends where both ids are in the list of transactions for the specific business. Then get count of t_id's where the u_id user is friends with the t_id user.

Appendix:

Original User Stories

Simple, Complex, or Analytical	As a <role>	I want <goal>	So that <reason>	NOTES
Simple	User	Requests and pay other users	I can maintain my and my friend's fair account balance	Pay: make a transaction(payment), Payments then updates User balances Request: make a request, wait for approve to = True (Indiv's must see requests), update payments
Complex & Analytical	Bank	See how many accounts have been overdrawn by Venmo transactions and by how much	See which bank accounts need to be charged overage fees and by how much	Select acc_nums where balance < 0, select (overage fee)*(overdraw balance)
Complex (2 inserts)	User	Transfer money to/from my bank account	I can use these funds inside and outside the Venmo platform	**amount in reference to BankAcc Transfer to bank (positive) Transfer from (negative)
Complex & Analytical	User	See my average spending over a specific time period or to specific users/businesses	Track how much and how quickly I spend over time	Select sum(payments.amt) where timestamp < (given timestamp) OR username = __ OR biz_name = __
Simple	Bank	See venmo withdrawals and deposits	Maintain accurate bank account balances	Return transfers (indicate withdrawal/deposit to bank acc)
Simple	User	Comment on user payments	To interact with other venmo users	
Simple	User	Add friends	I can interact with & view my friends transaction history	(i, j) (j, i)

Simple	Business	Receive payments from users	To pay for the debt of the services in full	Taken case of
Simple	User/ Individual	Edit and view my profile information	I can change data about my account	Update username only (for individuals)
Complex/Analytical	Business	See how many transactions are made by users that are friends	See how many customers recommended the business to their friends	Join friends on transactions

Conceptual Model