AKPay Proposal, Group 70
Database Implementation Report

[Database design overview and schemas] The purpose of this database project is to utilize the ID issues by universities to all residents of campus; The card, and the app that is made for it, is used in all the stores and eateries present on the campus, making life ultimately easier.

The application being designed implements an SQL schema with normalized tables being categorized as Users, UserAccounts, Vendors, VendorStatuses, and four different transaction types (i.e., RegularTx, Top-ups, Vendor Payment, and User2User). AKPay also has integrated status flags to control payment cycles and syncing properly (Pending, Sync Pending, Invalid, Sync failed, and Accepted), and support for both online (through our app) and offline payments (through LUMS ID card). The app is also designed to implement a notification system which informs users of all transactions taking place on their end.

A detailed design overview of the transactions presents the various features implemented in the app; the Regular transaction shows the payments done by the LUMS ID to the stores; the top-up transactions show how finances are transferred into a users account via various means; the Vendor Payment is to show the payment from the source to the vendor; and the user-to-user payments system demonstrates transactions from one user to another in a very simple mechanism. Tables are populated to demonstrate every example of transaction on a very extensive list of users (crossing over 5000 in number), with all transactions and users being unique since the data was formulated without any form of cross joins in order to inflate the number.

Data generated for each of the tables with the following constraints:

USERS TABLE:

- userID: It is supposed to be unique, never NULL, autoincrements starting from 1
- Email, also not NULL, and always unique.
- Phone number, also unique and not NULL, starting from the number 3, with 11 numbers in it, and no alphabets
- passwordHash for encrypted security, never NULL
- isDeleted and deletedAt is used for soft deletion, the prior being a bit value and the latter being a GETDATE()

USERACCOUNTS TABLE:

- It has a 1-1 relation with the Users
- The UserID is the primary key here, so one user can only have one account. The constraints it has is the userID must exist in the Users table first.
- The userbalance is prevented from ever dropping below 0.

- The isActive flag defaults to 0, requiring manual activation after creation

VendorStatus TABLE:

It is a lookup table to define states for vendors i.e. open, closed, suspended

- Constraints:
    - UNIQUE: No two statuses can have the same statusName
    - tillWheN: A field intended for logic to autoswitch status after a certain time

Vendors Table:

- Stores' profile for the merchants receiving payments. It has constraints that state the status assigned must exist in the vendorstatus table; it allows the manager phone number ot be NULL OR the name to be null (only one, not both at the same time); and the vendor balance can not be negative.

Transaction Status Table:

- Records payments from User to vendor; payment mode is offline by default, and the txStatusID defaults to 1.
- Constraints are: Foreign keys links fromuserID to VendorID and txStatusID. The transaction can not be of a negative amount or 0, and the input string is restricted to exactly either offline or online at one time.

TopUpSources table

- Defines where money comes from when users load their wallets (e.g., 'Credit Card', 'Bank Transfer', 'Admin').
- Constraints: Names must be unique.

TopUpTransactions Table:

- Records money being added to a User's wallet; Constraints are deposit amounts being greater than zero

UsertouserTransactions table:

Records peer-to-peer transactions; It has a constraint to prevent a user from sending money to themselves.

- The transfer amount must be greater than 0

VendorpaymentTransaction Table:

- Records payouts from the system/admin to the vendor; Payout must be greater than zero.

Notifications Table:

- A unified notification center; it restricts the target to only user or vendor; restricts categories to success failure or info; isRead is done in boolean; notifs_txtype: Ensures the transaction type string is one of the four valid tables

Lookup tables were also constructed for specific things like statuses, vendor titles, etc

[Feature descriptions] (bullet points with description) **[20]**

Stored Procedures

- **First Sprocs file:**

    - contains validation procedures via ValidateUser/ ValidateVendor/ValidateTopUpSource. It has simple stored procedures that check if a specific ID exists in its respective table; if not, it throws an error (50005 if the user is invalid, 50006 if the vendor is invalid). This prevents repetitive IF EXISTS checks and ensures data integrity before operations even start. Essentially, these serve as helper stored procedures to remove repetitive blocks of code from the Transaction stored procedures.
    - Transaction processing contains a unique procedure for all payment types individually; top-up adds funds to a user's wallet from an external source; U2U_tx involves transfer of funds between two users, and logic to lock rows, check for sufficient balance, and atomic deduction/addition of money; Regular transactions handled by Regular_tx handle payments from User to Vendor; Vendorpay_tx allows payment from source to vendor. This is the basic setup of the banking system of our project, and uses explicit transactions and commits to ensure money is never lost/doubled/mishandled during concurrent accessing.
    - TX_handling is the master transaction router, which takes a transaction type string and routes the request to the transaction type-specific stored procedure.
    - TransactionHistory is handled by GetUserTransactionHistory and GetVendorTransactionHistory where a unified list of all transactions is retrieved and provided based on User and Vendor displays.
    - The sproc files get daily spending trends, it takes a specific month and year and returns a day by day breakdown of successful transaction activity. It filters RegularTransactions for the calculated date range and the accepted status; It calculates three metrics per day i.e. TransactionVolume, Totalspent, and average ticket size.

The other sprocs contains sprocs for user management procedures such as:

- SP_softdeleteuser: Mark a user as isdeleted = 1 and deactivate their account, but only after passing strict checks such as the user not having pending transactions or a positive money balance. This is a standard way to remove a user while preserving history. The checks ensure we don't steal users' money by closing the account.
- SP_Purgeuser physically deleted the user record from the database for compliance and correction. Because government laws and stuff.
- Privileged stored procedure that atomically creates a User and associated Financial Account by temporarily suspending 'Block Direct Insert' triggers to enforce relational integrity. Related to the trigger in BLOCKDIRECTCREATE. Creation of accounts happens with SP_CreateUse

Functions

    - The functions contain display formatting using GetSenderName, GetReceiverName, where the function takes transactionID and transactionType

to join the correct tables to return a human-readable name so data presentation is normalized. Good for frontend.
    - GetTXSign is for financial signage, containing logic that determines if a transaction should be a credit or debit based on who is viewing it. For instance, if you pay a vendor it'll be +50 for them and -50 for you.z

Triggers

The triggers file contains triggers for the Data Integrity Using:

- Time consistency via TR_UserAccounts_ValidLastUpdate, TR_Vendors_ValidLastUpdate. They are triggers that run on updates to check if the new lastUpdateTime is EARLIER than the previous one, if so, it rolls back the transaction. The reason for using it is to prevent accidental backdating of records where old timestamps overwrite new ones; it also prevents erroneous transactions.
- Automated Account Creation via BLOCKDIRECTCREATE, defined below
- Deletion prevention using TR_USERS_BLOCKDIRECTDELETE, TR_USERACCOUNTS_BLOCKDIRECTDELETE; it has an INSTEAD OF DELETE trigger that throws an error if someone tries to run a standard DELETE statement to enforce a soft delete policy.
- Direct Creation prevention using BLOCKDIRECTCREATE: It is an interceptor, it disables INSERT statements for the users and useraccounts tables.

Views

- The views file contains a CheckBalance View, which joins users with useraccounts to provide a quick snapshot of current user balance, and a VendorStatus View to join vendor with vendorstatuses to allow the system to view operational details of the vendors business. These views allow the user to check their current balance, and check whether a vendor is closed or open respectively.

- The indexes and analytics SQL files are responsible for performance tuning and analytical views; The targeted lookups using IX_regularTransactions_fromUser and IX_regulartransactions_tovendor give non-clustered indexes on foreign keys (fromuserid and tovendorID) to improve performance of common operations, without which the whole database would have to be scanned
- The covering index is done by the IX_RegularTransactions_analytics, which indexes on txTimeStamp. It uses *INCLUDE* with txstatus and userID to speed up retrieval, which is used in stored procedure for data visualization.
- vw_VendorPerformanceRankings uses a common table expression and rank() function to order vendors by total revenue; it's used for business intelligence and creates a leaderboard of sorts
- vw_UserMonthlyStats: it aggregates individual transactions into summaries per user; used for user spending dashboards and to help users track their spending per place. It allows them to budget better.

- sp_GetDailySpendingTrends: A stored procedure taking a month and returns daily spending totals, also used for the same reasons as the previous wala

Common Table Expressions

Types of indexes

The indexes file contains auxiliary indexes for:

- Soft delete optimization with IX_Users_isDeleted: It indexes on the boolean flag isDeleted.
- Conditional uniqueness with IX_Users_Email_Active, IX_Users_Phone_Active: Unique indexes on email and phone that strictly apply WHERE ISDELETED = 0; allows for re-use of credentials. If a user deletes their account their email is still in the table; this index allows a new user to sign up with that same email later while still preventing two active users from sharing the same mail.
- Notifications using IX_Notifications_RecipientID: It indexes on the recipient of a notif, it speeds up the *my notifications* feed

Table Partitioning

The partition function uses timestamp to partition tables into different disks. A timestamp i.e '2024-01-01' can be used to partition data.

Names	**Roll numbers**
Ammar.	27100051
Muhammad Shahbaz Aziz Khan	27100049
Fajr Fatima	27020334
Muhammad Hassan Musa Gondal	27100456