Database Project

Kartik Anant Kulkarni **210493** **kartik21@iitk.ac.in**
Sharvil Sachin Athaley **210961** **sharvilsa21@iitk.ac.in**
Sanath Salampuria **210919** **sanaths21@iitk.ac.in**
Samyak Singhania **210917** **samyaks21@iitk.ac.in**
Raghav Shukla **210800** **raghavs21@iitk.ac.in**

CS315 Final Project

Abstract

We present the design and implementation of an Asset Management System to track labs, activities, funds, inventory, requests, and purchases. The solution includes a comprehensive schema, stored procedures, triggers, audit-log tables, and ad-hoc reporting queries.

1 Motivation and Problem Statement

Various labs at IITK possess assets that are shared across users and labs. We aim to provide a platform to simplify the coordination and usage of such assets. The goal is to maintain information about all assets across labs at IITK. Our system provides interfaces for users (with different roles) to create activities within labs and request assets accordingly. We also support the generation and management of purchase orders for unavailable assets.

2 Methodology

The Asset Management System follows a structured procedural flow to manage users, labs, activities, funds, assets, and purchase orders. Each component is tied to a transactional routine ensuring traceability and control. The methodology is broken down into the following steps:

Initially, a new user is created using the `create_user` procedure. Users are assigned roles and associated with specific institutes and departments. Labs are then defined through the `create_lab` procedure, each linked to a department.

Fund allocation begins with `allocate_funds_to_lab`, which deposits an initial budget to the lab. Activities are project-level initiatives within labs and are created using `create_activity`. Once an activity is established, the lab can transfer part of its funds to the activity using `allocate_funds_to_activity`.

Users participating in an activity can request assets. If the requested item is already in the lab's inventory, it is directly issued using `issue_assets`. Otherwise, the item must be created in the database using `create_item`.

For assets not readily available, a request is submitted through `raise_request`. Authorized personnel review and approve this request using `approve_request`. Upon approval, a purchase order is initiated via `create_PO`, and its contents populated using `populate_POItem`. The PO is then finalized through `approve_PO`.

Once assets are received, `receive_PO` records the delivery and updates the inventory. This closes the original asset request via `close_request`, after which assets can be issued through `issue_assets`.

If assets are no longer required, users can return them using `return_assets`. Unused or remaining funds can be released with `deallocate_funds`. Assets that are obsolete or broken can be removed from the system using `destroy_asset`.

When an activity concludes, it is terminated using `close_activity`, which also ensures any remaining funds or returned assets are appropriately transferred back to the lab.

This procedural framework ensures a controlled, auditable, and extendable system for asset and fund management in institutional environments.

Once the schema was finalized, we created a relational database with foreign key constraints and logging mechanisms to maintain consistency and auditability.

A full pipeline was designed for all core functionalities—starting from lab creation, fund allocation, item request and issuance, to asset return and activity closure. Each user action is tracked and reflected in child log tables for auditing.

We implemented various stored procedures to encapsulate business logic and triggers to automate logging. A user interface was then created to allow real-time interaction with the system. This UI was tested using generated sample data to verify correctness and integrity.

3 Implementation and Results

We organized the implementation into three SQL files:

- Table creation scripts
- Stored procedures for various functionalities
- Triggers to update child tables (e.g., logs) upon parent updates

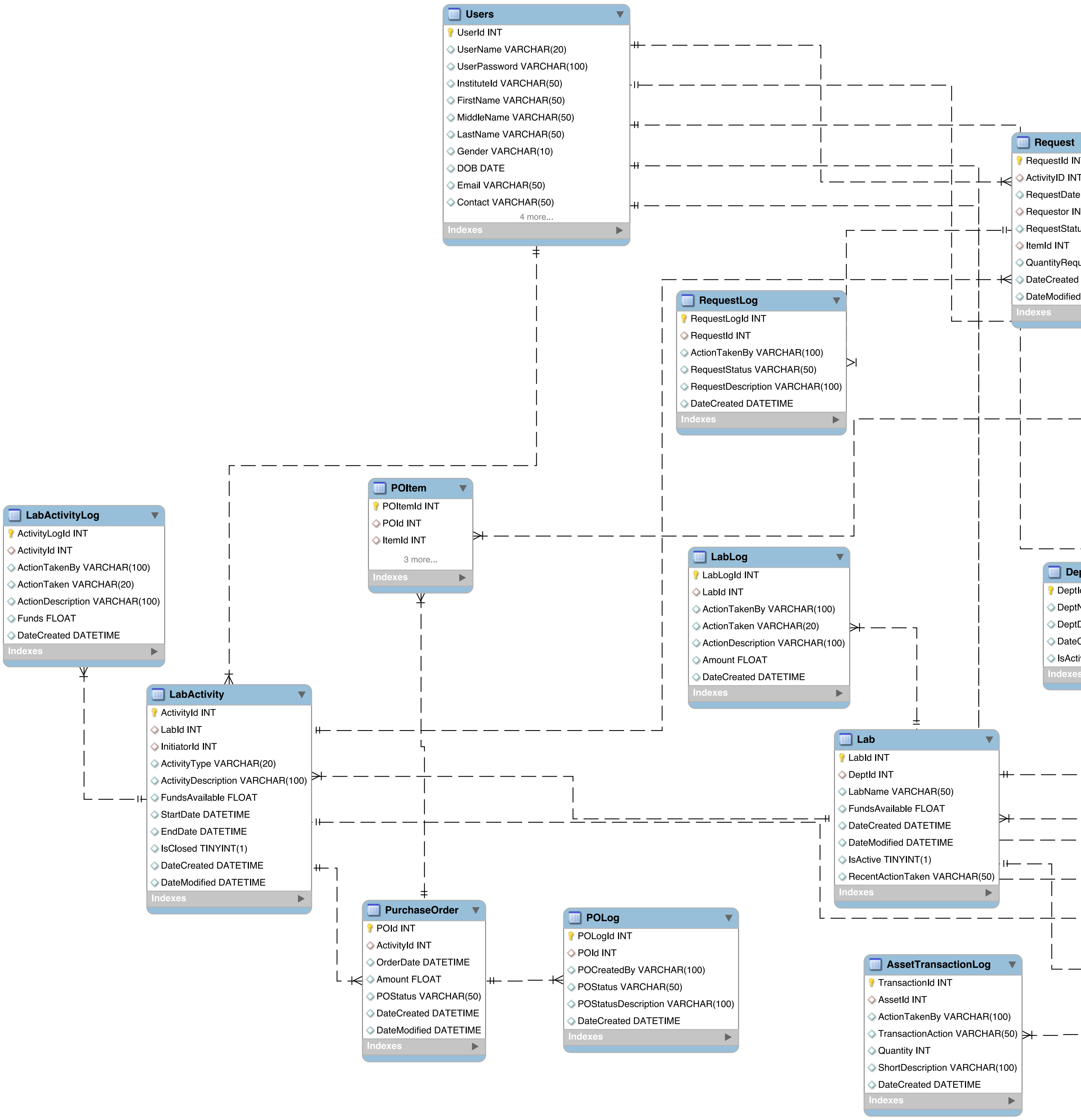

Figure 1: EER Diagram representing entities and relationships in the Asset Management System

Stored Procedures

1. `sp_create_user` – Inserts a new user, assigns role and lab via `UserRole`.
2. `sp_create_lab` – Inserts a new lab.
3. `sp_allocate_funds_to_lab` – Adds funds to a lab and logs the addition.
4. `sp_create_activity` – Creates a lab activity with initial details.
5. `sp_allocate_funds_to_activity` – Transfers lab funds to an activity if sufficient.
6. `sp_issue_assets` – Issues assets to user under an activity.
7. `sp_create_item` – Creates a new item if it is not present.
8. `sp_approve_request` – Requests are approved to purchase item or rejected if funds are not sufficient.
9. `sp_create_purchase_order` – Creates an order for an item to be purchased.
10. `sp_populate_POitem` – Populates a new item to the purchase table.
11. `sp_approve_PO` – Approve a purchase order if funds are sufficient.
12. `sp_receive_PO` – Receives purchase order of items.
13. `sp_issue_assets` – Logs issuance and updates quantity.
14. `sp_return_assets` – Logs returns and updates quantity.
15. `sp_deallocate_funds` – Returns the lab back to lab from an activity.
16. `sp_close_activity` – Closes an activity and refunds remaining funds.
17. `sp_deactivate_user` – Soft-deactivates a user.

Triggers

To ensure transparency, enforce logging, and maintain data consistency, the system makes use of several SQL triggers. These are automatically invoked in response to specific database events and are primarily used to log activities and monitor critical operations.

- **trg_lab_after_insert, trg_lab_after_update:**
  These triggers are executed when a new lab is created or an existing lab's details are updated. They log the lab's metadata into the `LabLog` table, capturing changes in available funds and modification timestamps.
- **trg_labactivity_after_insert, trg_labactivity_after_update:**
  Triggered on insertion or update of lab activities. These log actions such as creation of new activities or reallocation of funds in the `ActivityLog` table, maintaining historical tracking of activity-level changes.

- **trg_po_after_insert, trg_po_after_update:**
  These triggers log the creation and status changes of Purchase Orders into the `POLog` table. Details include PO ID, initiating or approving user, and timestamps, enabling financial transparency.
- **trg_request_after_insert, trg_request_after_update:**
  Captures asset request creation and approval status changes. The logs record the requester, item, quantity, and request outcome in the `RequestLog` for traceability.
- **trg_asset_after_insert, trg_asset_after_update:**
  Triggered on addition or modification of assets in the inventory. These triggers update logs to reflect quantity changes, asset issuance, returns, or destruction.

These triggers form a foundational layer for enforcing audit trails in the database. Their presence ensures that any operation involving labs, assets, purchase orders, or requests is captured automatically for accountability and historical reference.

- **LabLog Trigger:**
  This trigger is executed during fund allocation to a lab. It logs the lab identifier, allocated amount, and timestamp into a separate `LabLog` table for historical reference and auditing.
- **ActivityLog Trigger:**
  This trigger fires when a new activity is created or when funds are allocated to an activity. It captures the activity ID, lab ID, fund amount, and the user responsible for the action. The log entry is saved in the `ActivityLog` table.
- **POLog Trigger:**
  The `POLog` trigger is used to monitor and log the creation and approval of purchase orders. Whenever a PO is inserted or its status changes to 'approved', the trigger stores relevant metadata (PO ID, approving user, timestamps) into the `POLog` table.

These triggers provide automatic, tamper-proof audit trails for critical financial and asset-related transactions, thus supporting institutional accountability and simplifying compliance monitoring.

4 Entities

Below is the list of key entities used in the system, each representing a specific data element tracked in the asset management process:

- **RoleId, RoleName, RoleDesc, DateCreated, IsActive:** Represent user roles (e.g., Admin, Student) and their metadata like creation date and active status.
- **UserId, UserName, UserPassword, InstituteId, FirstName, MiddleName, LastName, Gender, DOB, Email, Contact, UserAddress, IsAppUser:** Define user credentials and profile information. `IsAppUser` flags whether the user has system login access.
- **DeptId, DeptName, DeptDesc:** Departments within the institute to which labs and users can belong.
- **LabId, LabName, FundsAvailable, DateModified, RecentActionTaken:** Labs associated with departments. Tracks available funds and last action on funds (e.g., Added, Refunded).

- **UserRoleId, DateJoined, DateLeft:** Assigns users to labs/departments with roles and tracks their period of association.
- **ActionTakenBy, ActionTaken, ActionDescription, Amount:** Captures actions (like fund allocation or item issuance) performed by users along with a description and amount affected.
- **ItemId, Category, Make, Model, WarrantyPeriodMonths, ItemDescription, CreatedBy:** Describe each unique item that can be used in a lab. Includes details such as brand, warranty, and creator.
- **ActivityId, InitiatorId, ActivityType, ActivityDescription, StartDate, EndDate, IsClosed:** Represents lab-based activities or projects. `IsClosed` indicates if the activity has concluded.
- **ActivityLogId, Funds:** Logs the history of fund allocation and usage per activity.
- **RequestId, RequestDate, Requestor, RequestStatus, QuantityRequested:** Represents a request made by a user for an item under a specific activity.
- **RequestLogId, RequestDescription:** Logs changes or actions on requests, such as approvals or rejections.
- **POId, OrderDate, POStatus:** Represents a Purchase Order raised to acquire items not available in the lab.
- **POItemId, QuantityOrdered, CostPerUnit:** Describes each line item in a purchase order.
- **POCreatedBy, POStatusDescription:** Details who created the PO and logs its status transition description.
- **AssetId, SerialNo, QuantityAvailable, StorageLocation, ShortDescription:** Defines a specific asset (based on an item), its stock level, and location in the lab.
- **TransactionId, TransactionAction:** Tracks asset transactions such as issuance or return.
- **ActivityTrasId, ProcessedBy, ActionDate:** Represents transactions performed on an activity involving asset issue/return, including the responsible lab assistant and date.

5 Discussions and Limitations

The system successfully demonstrates a modular and traceable approach to managing funds, assets, and activities in a laboratory environment. The use of structured procedures and SQL triggers enables a high level of auditability and process automation.

However, certain limitations were observed. One notable constraint is the system's inability to account for market-driven fluctuations in item prices. Currently, item costs are assumed static once entered into the purchase order, which may not reflect real-time pricing dynamics, especially in volatile procurement environments. This could lead to budget mismatches during actual procurement.

Additionally, scalability to enterprise-wide usage and integration with accounting systems or external vendors (via APIs or EDI) remains unimplemented. The current UI is functional but minimal and may require enhancements for better accessibility .

Future improvements can involve the inclusion of price update APIs, notification systems for procurement delays, and advanced analytics dashboards to support strategic decision-making.

6 Contributions

- Student 1: Made ER diagram and report . . .
- Student 2: Data generation . . .
- Student 3: SQL queries . . .
- Student 4: SQL queries . . .
- Student 5: Made UI . . .

Code repository: `https://github.com/kartik-iitk/Asset-Management`