Travel Reimbursement System, Relational database design and queries

Date: 04/11/2025

Changes to Entity Relationship Diagram

Our team updated the travel reimbursement EER diagram with several structural and semantic modifications to streamline the model. The major changes between the original design and the updated version include:

- **Consolidated expense details**: We removed the separate entities for **Transportation** and **Lodging**. Instead, the updated design introduces a multi-valued **Description** attribute within the travel expense report entity to capture these details. This attribute can hold multiple entries – each entry specifying the expense type (e.g. airfare, accommodation, meal), relevant dates (such as start/end dates for lodging or travel), visa information for international trips, and other necessary descriptors. By consolidating transportation and lodging into the report's description, we reduced the number of entities and kept all related data in one place.

- **Renamed and integrated accommodation data**: The entity previously labeled **Lodging** has been renamed to **Accommodation** for clarity, and its information is now linked through the **ReceiptFile** mechanism. In practice, this means we no longer treat lodging as a standalone entity with its own table. Instead, each accommodation expense (hotel stay) is recorded as a type of receipt entry with associated details like hotel name and cost. The **ReceiptFile** entity (or attribute) carries a field for expense type and amount, so a lodging receipt is simply one category of receipt. This change ties accommodation details directly to the receipt records, simplifying the relationship between expenses and their proof-of-purchase documents.

- **Removed Budget and Sponsored Project entities**: We dropped the **Budget** and **Sponsored Project** entities from the EER model. These entities were deemed non-essential for the core functionality of processing travel reimbursements. In the original diagram, they added extra complexity

– for example, tracking which budget or project funded a trip – but this level of detail wasn't necessary for our reimbursement scope. By removing these entities, the diagram now focuses on essential components (like employees, expense reports, approvals, and receipts) without the overhead of managing budget/project linkages that aren't directly used in the reimbursement process.

- **Clarified relationship cardinalities**: We refined the participation constraints and cardinalities in several relationships to improve clarity. The updated EER explicitly shows the correct **one-to-many (1:N)** or **many-to-many (M:N)** connections and indicates whether an entity's participation in a relationship is total (mandatory) or partial (optional). This removes any ambiguity present in the original diagram. For example, an **Employee** can submit multiple expense reports over time (an employee-to-report relationship of 1:N), whereas each **Travel Expense Report** is associated with exactly one employee (making the report's participation in that relationship total, since a report must have an employee). Similarly, we ensured that if a relationship is optional (say, an employee might have no travel reports if they haven't traveled), it's marked as such. These cardinality tweaks make the diagram easier to read and accurately reflect the business rules of the system.

<u>Why did we make the change</u>

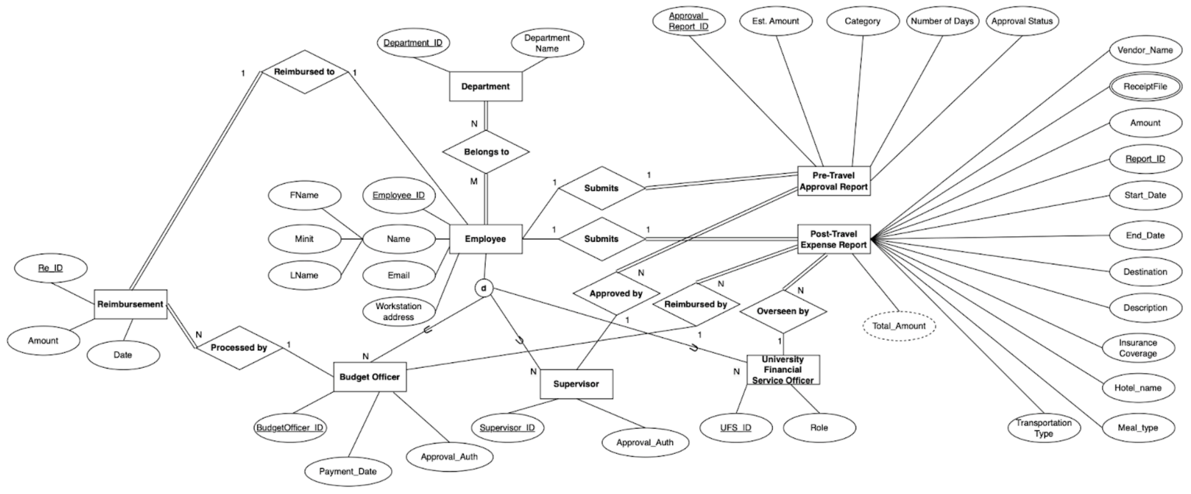
The driving motivations behind these adjustments were to simplify the design, enhance clarity, and ensure the model aligns with real needs and feedback:

- **Avoiding unnecessary complexity**: Our original EER design included a high level of detail – such as separate tables for every expense category and additional entities for budgets and projects – that ultimately did not add proportional value to the system. This over-detailing made the schema harder to understand and maintain without providing clear benefits to functionality. By eliminating or consolidating those extraneous elements, we streamlined the model. In other words, we focused on the data that actually gets used in processing travel reimbursements and removed the rest, thereby avoiding overengineering.

- **Improving readability and clarity**: Some aspects of the initial design were confusing or opaque, especially the cardinality and participation notations on relationships. For instance, it wasn't immediately obvious in the original diagram whether certain relationships were one-to-many or many-to-many, or which entities were optional in a relationship. We revised these to more standard and clear representations, making the EER diagram much easier to read at a glance. Now, any team member or stakeholder can look at the diagram and quickly grasp how many instances can be associated (e.g. one report has many receipts) and whether an association is mandatory or optional, reducing the chance of misinterpretation.

- **Incorporating feedback and real data needs**: We also made changes in response to feedback from our professor and in reflection of what the system genuinely requires. The professor pointed out that certain parts of our model were too detailed or beyond the project's scope, which helped us recognize features that weren't truly needed. For example, tracking **Budget** and **Sponsored Project** in detail was overkill for our purposes, so those were removed. By aligning the design with actual data and usage needs, we ensured that every entity and relationship in the EER serves a clear purpose. This feedback-driven refinement means the model is now tightly tailored to the travel reimbursement process as it will be implemented, with no superfluous components.

<u>EER DIAGRAM (Modified version)</u>:



Summary

In summary, the updated EER diagram is a **significant improvement** over the original version. By simplifying the entity structure and clarifying the relationships, we've created a model that is both easier to understand and more practical to implement. The refined design strips away unnecessary complexity and focuses on what really matters for the travel reimbursement system, thereby reducing the risk of overengineering. Overall, these changes make the database design more robust and user-friendly, developers can implement the schema more straightforwardly, and stakeholders can readily comprehend how travel reimbursements are represented and managed in the system.

Database Creation

Database Initialization

The database **travel** is designed to efficiently handle employee travel reimbursements and expense reporting at the University of Kentucky. It maintains details about departments, employees, financial roles, employees' travel records, and related reimbursement approvals.

MySQL Workbench Code

```
CREATE DATABASE travel;
USE travel;
```

```
CREATE SCHEMA IF NOT EXISTS `travel` DEFAULT CHARACTER SET
utf8 ;
USE `travel`;
```

Table Creation and Structure

1. Department

Purpose: Captures information about the university's departments, primarily for associating employees with their organizational units. This table plays a background role in linking employees to their respective departments but doesn't participate in direct travel processing workflows.

Attributes:

Name	Type	Description
Department_ID	INT	Primary key
Department_name	VARCHAR(255)	The full name of the department

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS `travel`.`Department` (
     `Department_ID` INT NOT NULL AUTO_INCREMENT,
     `Department_name` VARCHAR(255) NOT NULL,
     PRIMARY KEY (`Department_ID`),
     UNIQUE INDEX `Department_ID_UNIQUE`
     (`Department_ID` ASC) VISIBLE)
ENGINE = InnoDB;
```

2. Employee

Purpose: Holds the core personal and contact information of university employees who may act as travelers, supervisors, budget officers, or financial officers. It supports role-based associations and is the central entity in all travel-related operations.

Attributes:

Name	Type	Description
Employee_ID	INT	Primary key
FName	VARCHAR(50)	First name of the employee
Minit	VARCHAR(10)	Middle initial
LName	VARCHAR(50)	Last name of the employee
Email	VARCHAR(100)	Official email address
Workstation_address	VARCHAR(255)	Office location or desk address

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS `travel`.`Employee` (
  `Employee_ID` INT NOT NULL AUTO_INCREMENT,
  `FName` VARCHAR(50) NULL,
  `Minit` VARCHAR(10) NULL,
  `LName` VARCHAR(50) NULL,
  `Email` VARCHAR(100) NULL,
  `Workstation_address` VARCHAR(255) NULL,
  PRIMARY KEY (`Employee_ID`),
  UNIQUE INDEX `Employee_ID_UNIQUE`
  (`Employee_ID` ASC) VISIBLE)
ENGINE = InnoDB;
```

3. Budget Officer

Purpose: Captures payment-specific information for employees acting in the capacity of budget officers. They are responsible for disbursing reimbursement funds to travelers post-approval after evaluating the post travel expense report.

Attributes:

Name	Type	Description
BudgetOfficer_ID	INT	Primary key and Foreign key from Employee
Payment_Date	DATETIME	Date when reimbursement was processed
Approval_Auth	INT	Authorization status for financial decisions (1 means Approved, 0 means Rejected)

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS `travel`.`Budget Officer` (
  `BudgetOfficer_ID` INT NOT NULL,
  `Payment_Date` DATETIME NOT NULL,
  `Approval_Auth` INT NOT NULL,
   INDEX `fk_Budget Officer_Employee1_idx`
(`BudgetOfficer_ID` ASC) VISIBLE,
  PRIMARY KEY (`BudgetOfficer_ID`),
  CONSTRAINT `fk_Budget Officer_Employee1`
    FOREIGN KEY (`BudgetOfficer_ID`)
    REFERENCES `travel`.`Employee` (`Employee_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

4. Supervisor

Purpose: Stores supervisory-level roles of employees who approve pre-travel reports submitted by travelers. Supervisors play a gatekeeping role before travel occurs.

Attributes:

Name	Type	Description
Supervisor_ID	INT	Primary key and Foreign key from Employee
Approval_Auth	INT	Authorization status for pre-travel report decisions (1 means Approved, 0 means Rejected)

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS `travel`.`Supervisor` (
  `Supervisor_ID` INT NOT NULL,
  `Approval_Auth` INT NOT NULL,
  PRIMARY KEY (`Supervisor_ID`),
  INDEX `fk_Supervisor_Employee1_idx`
  (`Supervisor_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Supervisor_Employee1`
    FOREIGN KEY (`Supervisor_ID`)
    REFERENCES `travel`.`Employee` (`Employee_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;
```

5. **University Financial Service Officer**

Purpose: Contains financial officers who audit and oversee post-travel expense reports to ensure policy compliance of the post-travel expense reports before reimbursements are issued.

Attributes:

Name	Type	Description
UFS_ID	INT	Primary key and Foreign key from Employee
Role	VARCHAR(100)	Title or responsibility of the UFS officer. (Eg: Policy Monitor, Compliance Officer)

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS `travel`.`University
Financial Service Officer` (
  `UFS_ID` INT NOT NULL,
  `Role` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`UFS_ID`),
  INDEX `fk_University Financial
  Service Officer_Employee1_idx`
  (`UFS_ID` ASC)  VISIBLE,
  CONSTRAINT `fk_University Financial
  Service Officer_Employee1`
    FOREIGN KEY (`UFS_ID`)
    REFERENCES `travel`.`Employee` (`Employee_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;
```

6. Pre-Travel Approval Report

Purpose: Manage planned travel requests from employees. These entries record estimated expenses, trip categories, and approval status as reviewed by a supervisor.

Attributes:

Name	Type	Description
Approval_report_ID	INT	Primary key
Employee_ID	VARCHAR(100)	Foreign Key refers to the requesting employee
Supervisor_ID	INT	Foreign Key refers to the approving supervisor
Est_Amount	DECIMAL(8,2)	Estimated travel costs
Category	VARCHAR(100)	Nature of the travel (conference, research, etc)
Number_of_Days	INT	Duration of the travel
Approval_Status	INT	Approved by the supervisor or not (1 for approved, 0 for rejected)

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS `travel`.`Pre-Travel
Approval Report` (
  `Approval_report_ID` INT NOT NULL AUTO_INCREMENT,
  `Employee_ID` INT NOT NULL,
  `Supervisor_ID` INT NOT NULL,
  `Est. Amount` DECIMAL(8,2) NOT NULL,
  `Category` VARCHAR(100) NOT NULL,
  `Number of Days` INT NOT NULL,
  `Approval Status` INT NOT NULL,
  PRIMARY KEY (`Approval_report_ID`),
  INDEX `fk_Pre-Travel Approval Report_Employee1_
  idx`(`Employee_ID` ASC) VISIBLE,
  INDEX `fk_Pre-Travel Approval Report_Supervisor1_
  idx`(`Supervisor_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Pre-Travel Approval Report_Employee1`
    FOREIGN KEY (`Employee_ID`)
    REFERENCES `travel`.`Employee` (`Employee_ID`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Pre-Travel
  Approval Report_Supervisor1`
    FOREIGN KEY (`Supervisor_ID`)
    REFERENCES `travel`.`Supervisor` (`Supervisor_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

7. Post-Travel Expense Report

Purpose: Collect actual expense details once the employee returns from the trip. It includes receipts, destinations, vendors, and descriptions that support reimbursement processing.

Attributes:

Name	Type	Description
Report_ID	INT	Primary key
Employee_ID	INT	Foreign Key refers to the requesting employee
BudgetOfficer_ID	INT	Foreign Key refers to Budget officer
UFS_ID	INT	Foreign Key refers to UFS officer
Amount	DECIMAL(8,2)	Total claimed expenses
ReceiptFile	INT	ID to uploaded receipt files
Vendor_Name	VARCHAR(50)	Travel or service provider used
Start_Date	DATETIME	Start date of the trip
End_Date	DATETIME	End date of the trip
Destination	VARCHAR(50)	City/state or international location
Description	VARCHAR(255)	Brief note on each expense
Insurance_Coverage	VARCHAR(50)	Indicates whether insurance was applied
Hotel_name	VARCHAR(50)	Name of the accommodation
Meal_type	VARCHAR(100)	Type of meal coverage (Lunch, Dinner, Breakfast)
Transportation_Type	VARCHAR(50)	Mode of travel (flight, personal vehicle, public transport, etc.)

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS `travel`.`Post-Travel
Expense Report` (
  `Report_ID` INT NOT NULL AUTO_INCREMENT,
  `Employee_ID` INT NOT NULL,
  `BudgetOfficer_ID` INT NOT NULL,
  `UFS_ID` INT NOT NULL,
  `Amount` DECIMAL(8,2) NOT NULL,
  `ReceiptFile` INT NOT NULL,
  `Vendor_Name` VARCHAR(50) NOT NULL,
  `Start_Date` DATETIME NOT NULL,
  `End_Date` DATETIME NOT NULL,
  `Destination` VARCHAR(50) NOT NULL,
  `Description` VARCHAR(255) NOT NULL,
  `Insurance Coverage` VARCHAR(50) NULL,
  `Hotel_name` VARCHAR(50) NULL,
  `Meal_type` VARCHAR(100) NULL,
  `Transportation_Type` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`Report_ID`),
  INDEX `fk_Post-Travel Expense Report_Employee1_
  idx` (`Employee_ID` ASC) VISIBLE,
  INDEX `fk_Post-Travel Expense Report_
  Budget Officer1_idx` (`BudgetOfficer_ID`
  ASC) VISIBLE,
  INDEX `fk_Post-Travel Expense Report_
  University Financial Service _idx` (`UFS_ID`
  ASC) VISIBLE,
  CONSTRAINT `fk_Post-Travel Expense Report_Employee1`
    FOREIGN KEY (`Employee_ID`)
    REFERENCES `travel`.`Employee` (`Employee_ID`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Post-Travel Expense Report_
  Budget Officer1`
    FOREIGN KEY (`BudgetOfficer_ID`)
    REFERENCES `travel`.`Budget
    Officer` (`BudgetOfficer_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Post-Travel Expense Report_
  University Financial Service Of1`
    FOREIGN KEY (`UFS_ID`)
    REFERENCES `travel`.`University Financial
    Service Officer` (`UFS_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

8. **Reimbursement**

Purpose: Facilitate fund allocation to employees after their travel report has been processed and approved. This table connects the payment officer to the reimbursed employee.

Attributes:

Name	Type	Description
Re_ID	INT	Primary key and Foreign key from Employee
BudgetOfficer_ID	INT	Foreign Key refers to budget officer
Employee_ID	INT	Foreign Key refers to the requesting employee

Amount	DECIMAL(8,2)	Reimbursed amount
Date	DATETIME	Payment reimbursed date

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS `travel`.`Reimbursement` (
  `Re_ID` INT NOT NULL AUTO_INCREMENT,
  `BudgetOfficer_ID` INT NOT NULL,
  `Employee_ID` INT NOT NULL,
  `Amount` DECIMAL(8,2) NOT NULL,
  `Date` DATETIME NOT NULL,
  PRIMARY KEY (`Re_ID`),
  INDEX `fk_Reimbursement_Budget Officer1_
  idx` (`BudgetOfficer_ID` ASC) VISIBLE,
  INDEX `fk_Reimbursement_Employee1_idx` (`Employee_
  ID` ASC) VISIBLE,
  CONSTRAINT `fk_Reimbursement_Budget Officer1`
    FOREIGN KEY (`BudgetOfficer_ID`)
    REFERENCES `travel`.`Budget
    Officer` (`BudgetOfficer_ID`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Reimbursement_Employee1`
    FOREIGN KEY (`Employee_ID`)
    REFERENCES `travel`.`Employee` (`Employee_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Relationship Tables

1. Employee_Department

Purpose: Handles the many-to-many association between employees and departments. This is useful in cases where an employee may have roles or reporting responsibilities in more than one department.

Attributes:

Name	Type	Description
Department_ID	INT	Foreign key linking to the department
Employee_ID	INT	Foreign key referencing the employee

MySQL Workbench Script

```
CREATE TABLE IF NOT EXISTS
`travel`.`Employee_Department` (
  `Department_ID` INT NOT NULL,
  `Employee_ID` INT NOT NULL,
  PRIMARY KEY (`Department_ID`, `Employee_ID`),
  INDEX `fk_Employee_Department_Employee2_
  idx` (`Employee_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Employee_Department_Department1`
    FOREIGN KEY (`Department_ID`)
    REFERENCES `travel`.`Department` (`Department_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Employee_Department_Employee2`
    FOREIGN KEY (`Employee_ID`)
    REFERENCES `travel`.`Employee` (`Employee_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Questions & Explanations (10 Questions)

1. List all approved pre-travel reports with their estimated amounts and categories

Query Explanation: This query joins the Pre-Travel Approval Report table with the Employee table to display approved travel plans. It filters reports where the approval status equals 1 (approved) and displays employee names, estimated amounts, travel categories, and duration in days, sorted by highest estimated amount first.

Relevance: This query helps administrators track which travel plans have been approved and their associated costs, supporting budget planning and oversight. It addresses the requirement to "Request and record approval before travel" by providing visibility into the pre-travel approval process.

MySQL Workbench Script

```
use travel;

SELECT p.Approval_report_ID, e.FName, e.LName, p.`Est.
Amount`, p.Category, p.`Number of Days`
FROM `Pre-Travel Approval Report` p
JOIN Employee e ON p.Employee_ID = e.Employee_ID
WHERE p.`Approval Status` = 1
ORDER BY p.`Est. Amount` DESC;
```

2. Calculate the total reimbursement amount for each department

Query Explanation: This query aggregates reimbursement amounts by department through multiple joins. It connects reimbursements to employees, then to their departments through the junction table, and calculates the sum of reimbursements for each department.

Relevance: This query supports the financial oversight requirement by showing how travel expenses are distributed across departments. It helps "List all reimbursements for employees from a department for a time period" as specified in the requirements, enabling financial administrators to track departmental spending patterns.

MySQL Workbench Script

```
SELECT d.Department_name, SUM(r.Amount) as
Total_Reimbursement
FROM Reimbursement r
JOIN Employee e ON r.Employee_ID = e.Employee_ID
JOIN Employee_Department ed ON e.Employee_ID =
ed.Employee_ID
JOIN Department d ON ed.Department_ID = d.Department_ID
GROUP BY d.Department_name
ORDER BY Total_Reimbursement DESC;
```

3. Find the most common travel destinations

Query Explanation: This query counts the number of trips to each destination by grouping the post-travel expense reports by destination. It then orders the results to show the most frequently visited locations at the top.

Relevance: Understanding travel patterns is important for institutional planning and vendor negotiations. This query helps fulfill the requirement to "Audit travel expenses" by providing insights into where university travel resources are primarily being directed.

MySQL Workbench Script

```
SELECT Destination, COUNT(*) as Visit_Count
FROM `post-travel expense report`
GROUP BY Destination
ORDER BY Visit_Count DESC;
```

4. Compare estimated vs. actual travel expenses for each employee

Query Explanation: This query joins pre-travel approval reports with post-travel expense reports by employee ID to compare estimated and actual travel costs. It calculates the difference between estimated and actual amounts to identify discrepancies.

Relevance: This query addresses the need to "Reimburse all travel expenses for a trip" by helping administrators identify when actual expenses significantly differ from estimates. It supports financial accountability and can help improve the accuracy of future travel budgeting.

MySQL Workbench Script

```
SELECT e.Employee_ID, e.FName, e.LName,
p.`Est. Amount` as Estimated_Amount,
pt.Amount as Actual_Amount,
(pt.Amount - p.`Est. Amount`) as Difference
FROM `Pre-Travel Approval Report` p
JOIN Employee e ON p.Employee_ID = e.Employee_ID
JOIN `post-travel expense report` pt ON p.Employee_ID =
pt.Employee_ID
WHERE p.`Approval Status` = 1
ORDER BY Difference DESC;
```

5. Identify employees with the highest travel expenses

Query Explanation: This query aggregates total travel expenses for each employee by joining the employee table with post-travel expense reports. It groups results by employee and sorts them by their total expenses, showing the top spenders.

Relevance: This query supports the "Audit travel expenses" requirement by identifying high-spending travelers who may require additional scrutiny. It helps financial administrators monitor resource allocation and ensures compliance with university expense policies.

MySQL Workbench Script

```
SELECT e.Employee_ID, e.FName, e.LName, SUM(pt.Amount)
as Total_Expenses
FROM Employee e
JOIN `post-travel expense report` pt ON e.Employee_ID =
pt.Employee_ID
GROUP BY e.Employee_ID, e.FName, e.LName
ORDER BY Total_Expenses DESC
LIMIT 5;
```

6. Find the most commonly used transportation types and their average costs

Query Explanation: This query analyzes transportation preferences by counting the number of times each transportation type is used and calculating the average cost. It groups post-travel expense reports by transportation type and orders results by frequency.

Relevance: This query addresses the requirement to include "at least 2 different types of transportation" by providing insights into transportation usage patterns. It helps optimize transportation choices and budget allocation by identifying cost-efficient options.

MySQL Workbench Script

```
SELECT Transportation_Type, COUNT(*) as Usage_Count,
AVG(Amount) as Avg_Cost
FROM `post-travel expense report`
GROUP BY Transportation_Type
ORDER BY Usage_Count DESC;
```

7. List all supervisors and the number of travel requests they've approved

Query Explanation: This query identifies supervisors' workloads by counting approved travel requests. It joins the pre-travel approval report with supervisor and employee tables, filters for approved reports (status = 1), and groups results by supervisor.

Relevance: This query supports the role of "Department administers (authorize travel)" in the requirements by tracking supervisor approval activities. It helps ensure an equitable distribution of approval responsibilities and identifies potential bottlenecks in the process.

MySQL Workbench Script

```
SELECT s.Supervisor_ID, e.FName, e.LName, COUNT(*) as
Approval_Count
FROM `Pre-Travel Approval Report` p
JOIN Supervisor s ON p.Supervisor_ID = s.Supervisor_ID
JOIN Employee e ON s.Supervisor_ID = e.Employee_ID
WHERE p.`Approval Status` = 1
GROUP BY s.Supervisor_ID, e.FName, e.LName
ORDER BY Approval_Count DESC;
```

8. Calculate the average meal expenses by meal type

Query Explanation: This query calculates the average expense for each meal type by grouping post-travel expense reports by the meal_type field and computing the average amount spent.

Relevance: This query addresses the "per diem expenses" requirement by analyzing meal spending patterns. It helps financial administrators validate that meal expenses align with university per diem policies and identify any unusual expense patterns.

MySQL Workbench Script

```
SELECT Meal_type, AVG(Amount) as Avg_Expense
FROM `post-travel expense report`
GROUP BY Meal_type
ORDER BY Avg_Expense DESC;
```

9. Compare reimbursement processing by different budget officers

Query Explanation: This query evaluates budget officer performance by counting the number of reimbursements each has processed and calculating their average reimbursement amounts. It joins the Budget Officer table with Employee and Reimbursement tables to provide a comprehensive view.

Relevance: This query supports the "Departmental budget officer (allocate and manage correct funds for travel)" requirement by monitoring how different budget officers handle reimbursements. It helps identify potential process improvements and ensures consistent application of reimbursement policies.

MySQL Workbench Script

```
SELECT b.BudgetOfficer_ID, e.FName, e.LName,
COUNT(r.Re_ID) as Reimbursement_Count,
AVG(r.Amount) as Avg_AmouFROM `Budget Officer` b
JOIN Employee e ON b.BudgetOfficer_ID = e.Employee_ID
JOIN Reimbursement r ON b.BudgetOfficer_ID =
r.BudgetOfficer_ID
GROUP BY b.BudgetOfficer_ID, e.FName, e.LName
ORDER BY Reimbursement_Count DESC;
```

10. List travel reports that had both air transportation and included all meals

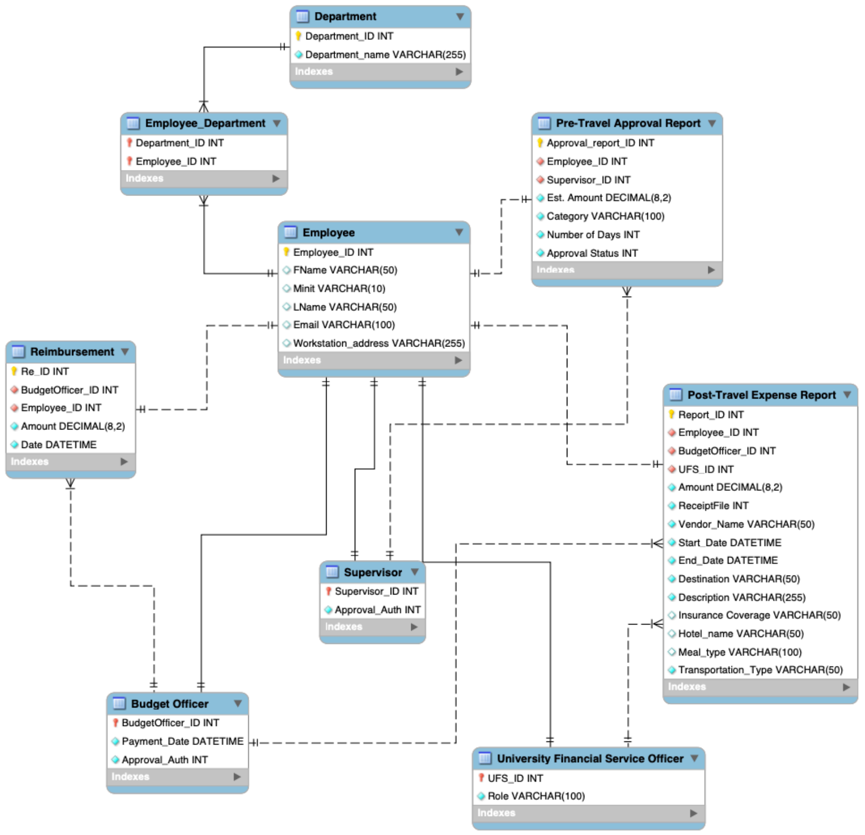
Query Explanation: This query identifies specific travel patterns by filtering post-travel expense reports for trips that used air transportation and included "All Meals" in their meal type. It joins with the Employee table to include traveler information.

Relevance: This query addresses both the transportation and meal expense tracking requirements. It helps administrators understand comprehensive travel packages that include both air travel and full meal coverage, supporting analysis of complete travel costs.

MySQL Workbench Script

```
SELECT pt.Report_ID, e.FName, e.LName, pt.Destination,
pt.Start_Date, pt.End_Date
FROM `post-travel expense report` pt
JOIN Employee e ON pt.Employee_ID = e.Employee_ID
WHERE pt.Transportation_Type = 'Air'
AND pt.Meal_type LIKE '%All Meals%';
```

<u>Relational schema Graph:</u>



Conclusion

The Travel Reimbursement System database was carefully structured to support the full cycle of travel expense management from pre-approval to final reimbursement within a university setting. Each table was designed to represent a specific process or role, with particular attention to maintaining clarity, data consistency, and referential integrity. Core entities such as Employee, Pre-Travel Approval Report, and Post-Travel Expense Report were supported by role-based tables and relationship mappings to reflect the multiple responsibilities that individuals can hold within the organization.

A key design challenge we encountered was how to accurately represent reimbursement flow to employees while accounting for their departmental affiliations. Initially, we considered embedding department references directly into transactional tables, but this approach introduced redundancy. To resolve this, we introduced a separate relationship table, Employee_Department, to cleanly handle the many-to-many association between departments and employees. Similarly, we debated breaking down travel components into separate entities but ultimately embedded them into descriptive attributes to preserve simplicity. These decisions reflect our effort to balance functionality with ease of implementation, resulting in a relational schema that is both robust and adaptable.