Post Office Database Scenario

by Taylor Merry, Zhen Liang, and Dai Tan Ngo

Introduction

Our group chose to create a database for an imaginary post office. The post office would need to keep track of its company branches, shipments, individual packages, vehicles, employees, customers, reviews, and transactions.

Data Schema and Non-Obvious Definitions

Our database consists of 12 entities: Branch, Shipment, Package, Vehicle, Employee, Employee_Phone, Employee_Email, Customer, Customer_Phone, Customer_Email, Review, and Transaction.

The Branch table contains information about all of the branch locations. It's attributes are Name, Address, City, State, and Country. Name is the primary key because each branch has a unique name. We decided to not normalize city, state, and country because we thought that would be overkill and annoying when querying for branch location information.

The Shipment relation contains information about each shipment. It consists of six attributes: ShipmentID, FromBranch, VehicleID, TotalWeight, TotalVolume, and DriverID. The primary key is ShipmentID. Foreign keys are: FromBranch references Branch.Name, VehicleID references Vehicle.VehicleID, and DriverID references Employee.EmployeeID.

The package table contains information about each package. It's attributes are PackageID, ShipmentID, ShippingAddress, Type, Weight, Volume, ArrivalDate, and LastBranchLocation. Type refers to type of shipping (same-day, 1-day, 2-day, etc.). ArrivalDate is the estimated arrival date for packages currently being shipped or the actual arrival date for packages that have been delivered. LastBranchLocation is the last branch that the package was at. PackageID is the primary key. Foreign keys are: ShipmentID references Shipment.ShipmentID and LastBranchLocation references Branch.Name.

The Vehicle relation contains information about the post office's vehicles. It consists of VehicleID, StorageSize, and CurrentBranchLocation as its three attributes. StorageSize is for small, medium, and large storage spaces per vehicle so the post office can figure out the total volume of packages they can put in a vehicle. CurrentBranchLocation is the name of the branch where the vehicle is stationed. The primary key is VehicleID. LastBranchLocation references Branch.Name.

The Employee table contains information for all of the post office's staff. Its attributes are EmployeeID, FirstName, LastName, Branch, Role, Address, Salary, Birthday, and Age. Role is their current position at the company (mailman, CEO, etc.). EmployeeID is the primary key. The one foreign key is Branch references Branch.Name. We normalized the Employee table to third normal form so therefore we had to create to separate tables Employee_Phone and Employee_Email. The Employee_Phone table has attributes of Employee_ID and Phone with Employee_ID being the primary key and a foreign key referencing Employee.EmployeeID. The

Employee_Email table has attributes of Employee_ID and Email with Employee_ID being the primary key and a foreign key referencing Employee.EmployeeID.

The Customer relation consists of information for all of the post office's registered customers. It has four attributes: CustomerID, Firstname, Lastname, and Address with CustomerID being the primary key. It does not contain a foreign key. We normalized the Customer relation to third normal form (just like Employee above) so therefore we made two separate relations Customer_Phone and Customer_Email. The Customer_Phone table has attributes of Customer_ID and Phone with Customer_ID being the primary key and a foreign key referencing Customer.CustomerID. The Customer_Email table has attributes of Customer_ID and Email with Customer_ID being the primary key and a foreign key referencing Customer.CustomerID.

The Review table contains the reviews that customers write for the post office. The reviews are limited to 1000 characters. The table consists of the attributes ReviewID, CustomerID, Review, RatingScore, and BranchName. RastingScore is a score of 0 to 5. The primary key is ReviewID and the foreign keys are CustomerID referencing Customer.CustomerID and BranchName referencing Branch.Name.

The Transaction relation contains the information for all of the transactions for the post office. It has nine attributes: TransactionID, Date, Cost, CustomerID, EmployeeID, PaymentMethod, Branch, Online, and PackageID. EmployeeID is the ID of the employee who helped make the transaction. PaymentMethod is a string that is cash, credit card, debit card, check, etc. Online is an int 0 or 1 that is a 0 for in-store purchases and a 1 for online purchases. In the case of an online purchase, the Branch and EmployeeID fields would be null. The primary key is TransactionID. The foreign keys are CustomerID references Customer.CustomerID, EmployeeID references Employee.EmployeeID, Branch references Branch.Name, and PackageID references Package.PackageID.

Indexes

We decided to index every foreign key in our database. This is because we have many questions that we must answer with joins. Therefore, it is much faster to join by an indexed foreign key. These indexes speed up the overall runtime of most of the queries used on this database.

We thought it would be a good idea to over-index tables such as Branch, Vehicle, Employee, Employee_Phone, Employee_Email, Customer, Customer_Phone, Customer_Email, and Feedback because those tables are used much more often for lookup than insertion. Therefore, it helps to have indexes in columns that are often used for specific lookups to speed up runtime. Some of the columns (that aren't foreign keys) we indexed for this reason were: Branch.City, Branch.State, Employee.LastName, Employee.Role, Customer.LastName, Customer.Phone, and Review.RatingScore.

We also wanted to under-index relations such as Package, Shipment, and Transaction because they are used much more often for insertion rather than lookups so we didn't want to slow down insertion time with more indexes. Unfortunately, these tables contain many foreign keys that are indexed. So, we decided to not index any of the other columns.

Common Ways of Accessing the Data

- **Branch Locations:** The post office can see how many branches and employees they have in each city, state, and country by joining Employee with Branch on Employee.Branch = Branch.Name.
- **Branch Statistics:** The post office can use aggregate functions on the Transaction table grouping by branch to see specific branch statistics.
- **Reviews on Branches:** The post office can use the Review table to see the ratings statistics for each branch. Then they can join with Employee on Review.BranchName = Employee.Branch to see employees that work for the branches and try to identify the reasons for good/bad reviews.
- **Specific Role Information:** The company can group by role on the Employee table and then look at the salaries and ages of each role to determine if they need replacements.
- **Online Ordering:** The company can use the Transaction to see the effect of online ordering.
- **Active vs. Delivered Packages:** On the Package table, the post office can filter by ArrivalDate is greater than or less than or equal to the current date to see information for active packages or information for delivered packages.
- **Size of Shipment:** The company can find the average and maximum sizes of shipment by using Aggregate functions on the Shipment table for each branch to decide which vehicles (and their individual storage space) should be allotted to each branch.
- **Locating Vehicles:** The post office would use the Vehicle table to locate every vehicle.
- **Contacting Customers:** The company can use the Customer_Email and Customer_Phone tables to contact customers. Also can join with Transaction to find customers who have actually made a transaction over the past year to contact only those customers.
- **Contacting Employees:** The post office can use the Employee_Email table to send out company-wide emails, and can join with Employee on Employee_Email.EmployeeID = Employee.EmployeeID and can group by or filter branch to send branch-wide emails or group by or filter role to send job information.

Questions our Database Can Answer

1. What is the maximum volume and weight of a shipment?
```
select max(S.TotalVolume) as 'Max Volume', max(S.TotalWeight) as 'Max Weight'
from Shipment S;
```

2. What are the emails for all of the employees in the “North Seattle” branch?
```
select E1.Email
from Employee_Email E1
join Employee E2 on E2.EmployeeID = E1.EmployeeID
where E2.Branch = 'North Seattle';
```

3. What is the address of the branch that has the most vehicles currently located there?
```
select top 1 B.[Address]
from Branch B
left join Vehicle V on V.CurrentBranchLocation = B.[Name]
group by B.[Address]
order by count(V.VehicleID) desc;
```

4. What is the first name and last name of employees who are NOT also customers?
```
select E.FirstName, E.LastName
from Employee E
left join Customer C on C.FirstName = E.FirstName
and C.LastName = E.LastName
where C.CustomerID IS NULL;
```

5. How many packages is each vehicle currently carrying?
```
select V.VehicleID, count(P.PackageID) as Packages
from Package P
left join Shipment S on S.ShipmentID = P.ShipmentID
left join Vehicle V on V.VehicleID = S.VehicleID
group by V.VehicleID
```

6. Find all branches who have not received a review.
```
SELECT B.[Name] FROM Branch B
LEFT JOIN Review R ON B.[Name] = R.BranchName
WHERE R.RatingScore IS NULL;
```

7. Find all packages that are going to an address that isn’t registered to a customer in our database.
```
SELECT P.PackageID, P.ShippingAddress FROM Package P
LEFT JOIN Customer C ON P.ShippingAddress = C.Address
WHERE C.CustomerID IS NULL;
```

8. Find branches that are located in a city with no state (international branches).
```
SELECT [Name], Country FROM Branch
WHERE [State] IS NULL;
```

9. Find all of the customers who made transactions with packages’ weights larger than 2.4 and packages’ volume larger than 2.4. Return their first names, last names, addresses, and emails.

```
SELECT C.FirstName, C.LastName, C.Address, C2.Email FROM Customer C
JOIN Customer_Email C2 ON C.CustomerID = C2.CustomerID
JOIN [Transaction] T ON C.CustomerID = T.CustomerID
JOIN Package P ON T.PackageID = P.PackageID
WHERE P.Weight > 2.4
AND P.Volume > 2.4;
```

10. Find all of the shipments which were driven by Peter Scarlet. Return the names of the branches that the shipment were delivered from.

```
SELECT S.FromBranch FROM Shipment S
JOIN Employee E ON S.DriverID = E.EmployeeID
WHERE E.FirstName = ‘Peter’
AND E.LastName = ‘Scarlet’;
```