# Library Billing System

# STAGE 1

## Project Overview

This project represents a **Library Billing System** designed to track subscriptions, penalties, employee wages, assets, and procurement processes within a library. It focuses on managing the core aspects of library operations and their financial transactions.

---

## Features

- **Reader Management**: Track readers' subscriptions, renewals, and penalties.
- **Employee Management**: Manage employee wages, payroll, and compensation.
- **Asset Tracking**: Monitor physical library assets, including insurance and procurement.
- **Financial Tracking**: Record costs associated with penalties, procurement, and employee wages.

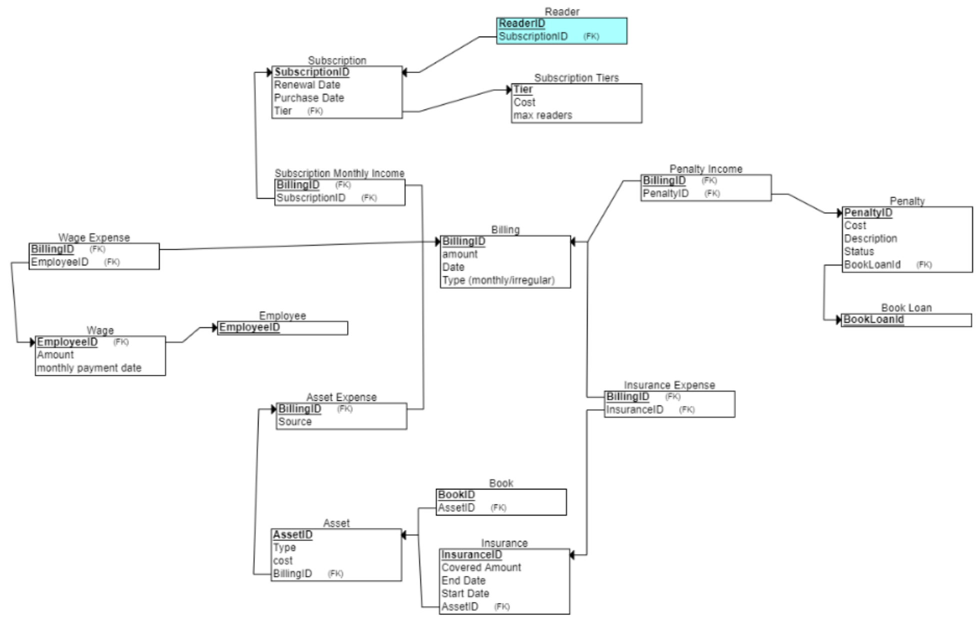
---

## Design Specifications and Entity Relationship Diagram

### Data Structure Diagram (DSD)

Below is the Data Structure Diagram (DSD) representing the logical flow and structure of the system:

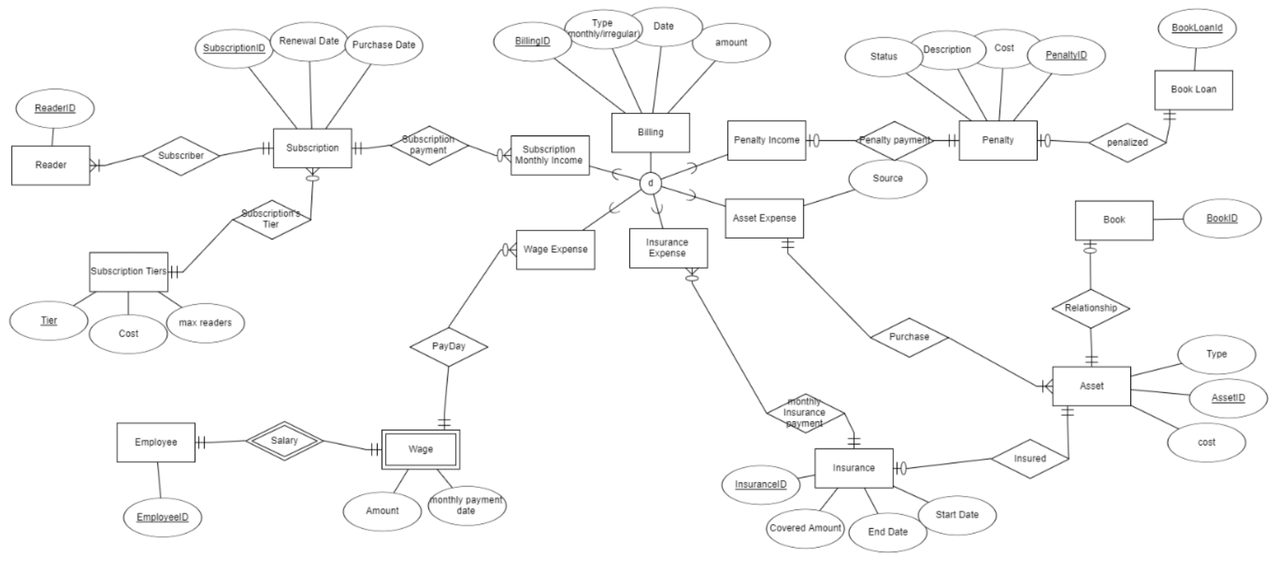
**Data Structure Diagram**



### Entity Relationship Diagram (ERD)

The following Entity Relationship Diagram (ERD) illustrates the relationships between the various entities in the system:

**Entity Relationship Diagram**



---

## Entities and Relationships

### Entities

The system includes the following key entities:

# Library Database Schema

This database schema represents a comprehensive library management system that includes subscriptions, billing, penalties, asset management, and more.

## Tables Overview

- **Reader**: Represents an individual who subscribes to the library.
  - **Attributes**: `ReaderID` (Primary Key), `SubscriptionID` (Foreign Key)
- **Subscription**: Tracks library subscription details.
  - **Attributes**: `SubscriptionID` (Primary Key), `Renewal_Date`, `Purchase_Date`, `Tier` (Foreign Key)
- **Subscription_Tiers**: Defines different subscription plans offered by the library.
  - **Attributes**: `Tier` (Primary Key), `Cost`, `max_readers`
- **Employee**: Represents a library employee.
  - **Attributes**: `EmployeeID` (Primary Key)

- **Wage**: Tracks wages paid to employees.
  - **Attributes**: `Amount`, `monthly_payment_date`, `EmployeeID` (Primary Key and Foreign Key)
- **Wage_Expense**: Links employee wages to billing records.
  - **Attributes**: `BillingID` (Primary Key, Foreign Key), `EmployeeID` (Foreign Key)
- **Book**: Represents books in the library.
  - **Attributes**: `BookID` (Primary Key), `AssetID` (Foreign Key)
- **Book_Loan**: Tracks loans of books to readers.
  - **Attributes**: `BookLoanId` (Primary Key)
- **Penalty**: Represents penalties incurred by readers for overdue or damaged books.
  - **Attributes**: `PenaltyID` (Primary Key), `Cost`, `Description`, `Status`, `BookLoanId` (Foreign Key)
- **Penalty_Income**: Links penalties to billing records.
  - **Attributes**: `BillingID` (Primary Key, Foreign Key), `PenaltyID` (Foreign Key)
- **Billing**: Tracks all financial transactions in the library.
  - **Attributes**: `BillingID` (Primary Key), `amount`, `Date`
- **Asset**: Represents physical assets owned by the library.
  - **Attributes**: `AssetID` (Primary Key), `Type`, `cost`, `BillingID` (Foreign Key)
- **Asset_Expense**: Links assets to billing records.
  - **Attributes**: `Source`, `BillingID` (Primary Key and Foreign Key)
- **Insurance**: Tracks insurance policies for library assets.
  - **Attributes**: `InsuranceID` (Primary Key), `Covered_Amount`, `EndDate`, `StartDate`, `AssetID` (Foreign Key)
- **Insurance_Expense**: Links insurance policies to billing records.
  - **Attributes**: `BillingID` (Primary Key and Foreign Key), `InsuranceID` (Foreign Key)
- **Subscription_Monthly_Income**: Tracks income from subscription renewals.
  - **Attributes**: `BillingID` (Primary Key and Foreign Key), `SubscriptionID` (Foreign Key)

---

## Design Rationale

### Why These Entities Were Chosen

The design of the Library Billing System is centered around the essential operations of the library, ensuring efficient management and tracking of both financial and operational data.

- **Reader**: This entity tracks individuals who interact with the library's services. The reader’s subscription details are essential for determining billing and penalties.
- **Subscription**: Subscriptions define the reader’s relationship with the library and are crucial for tracking renewal dates and managing service usage.
- **Penalty**: Penalties are needed to enforce library rules and track costs incurred due to overdue or damaged items.
- **Employee and Wage**: Employees are central to library operations, and tracking wages is necessary for payroll and compensation purposes.
- **Asset and Insurance**: Assets represent library property, and insurance coverage is necessary for risk management.
- **Procurement**: This entity is critical for tracking new acquisitions and maintaining inventory levels.

---

## Relationships

1. **Reader ↔ Subscription**

   - A **Reader** is part of a **Subscription** that can have multiple readers.
   - Links readers to their active or previous subscriptions.

2. **Subscription ↔ Subscription Tiers**

   - A **Subscription** belongs to a specific **Subscription Tier**.
   - Defines different subscription plans based on cost and maximum number of readers.

3. **Subscription ↔ Billing**

   - Each **Subscription Payment** contributes to **Subscription Monthly Income** tracked under **Billing**.

4. **Penalty ↔ Billing**

   - Penalties are recorded in **Penalty Income** under **Billing**.
   - The **Penalty** entity tracks details such as status, description, cost, and penalty payment.

5. **Penalty ↔ Book Loan**

   - A **Penalty** is associated with a **Book Loan** (e.g., overdue fines or damages).

6. **Asset ↔ Billing**

   - Expenses for **Assets** are recorded under **Asset Expense** in **Billing**.
   - Represents costs incurred by purchasing or maintaining assets.

7. **Book ↔ Asset**

   - A **Book** is a **Asset** that belongs to the library (e.g., purchased as part of the library's inventory).
   - Represents the library's physical collection.

8. **Employee ↔ Wage**
   - An **Employee** receives a **Wage**.
   - Represents the library's staff and their associated salary payments.
9. **Wage ↔ Billing**
   - Wages paid to employees are recorded as **Wage Expense** in **Billing**.
10. **Insurance ↔ Billing**
    - Payments for **Insurance** are recorded under **Insurance Expense** in **Billing**.
    - Allows tracking of regular insurance payments for covered items, rooms, or the building.
11. **Asset ↔ Insurance**
    - An **Asset** can be **Insured**.
    - Represents coverage for valuable library assets.
12. **Insurance ↔ Billing**
    - The **Insurance Expense** links payments for monthly insurance premiums to **Billing**.
13. **Billing** has a with various expense and income categories:
    1. **Subscription Monthly Income**: Captures income from subscriptions.
    2. **Penalty Income**: Captures income from penalties.
    3. **Asset Expense**: Captures expenses for purchasing or maintaining assets.
    4. **Wage Expense**: Captures expenses related to employee wages.
    5. **Insurance Expense**: Captures expenses related to library insurance.

---

## Use Cases

The Library Billing System supports the following use cases:

- **Subscription Management**: Library administrators can track reader subscriptions, renewals, and penalties.
- **Employee Payroll**: Administrators can manage employee wages, ensuring timely payments.
- **Asset Management**: Keep track of library assets and their associated insurance coverage.
- **Billing Process**: Record details about the acquisition of new assets and incomes, as well as recording the library's expenses.

---

## Target Users

- **Library Administrators**: They will manage subscriptions, penalties, employee wages, and assets.
- **Employees**: Employees can view their wage records and interact with the system for asset-related tasks.
- **Readers**: Readers' information will be managed through their subscriptions and penalties.

---

## Data Population Process

### Data Population Overview

The data population script generates random data for multiple tables using the **Faker** library and inserts it into the PostgreSQL database using **psycopg2**. The script populates data for tables such as **Insurance**, **Wage**, **Penalty**, **Subscription**, **Reader**, **Asset**, **Employee** and more.

### Tables Populated

1. **Subscription** - Tracks subscription plans for readers, including renewal and purchase dates, and the subscription tier.
2. **Subscription Tiers** - Defines various subscription levels, specifying their cost and the maximum number of readers allowed.
3. **Billing** - Records financial transactions, including subscription payments, penalties, and other expenses.
4. **Subscription Monthly Income** - Links subscriptions to their corresponding monthly billing entries.
5. **Penalty** - Details penalties issued for late returns or damaged books, including cost and description.
6. **Penalty Income** - Tracks penalties collected through billing records.
7. **Asset** - Tracks library-owned items, including books, equipment, or other resources, with associated costs.
8. **Asset Expense** - Records expenses related to library assets and their sources.
9. **Insurance** - Details insurance policies for library assets, including covered amounts and policy periods.
10. **Insurance Expense** - Tracks insurance costs paid for assets, linked to billing records.
11. **Wage** - Tracks employee wages and payment dates.
12. **Wage Expense** - Records wage payments linked to the billing system.

(the following are tables that are managed by other teams) 13. **Book** - Maintains details about library books, including their associated assets.
14. **Reader** - Stores information about individual readers, including their subscription details.
15. **Employee** - Stores employee data, including identifiers and wage details.
16. **Book Loan** - Represents information about books borrowed by readers.

### Data Size

The script populates the following number of records for each table:

- **employees** = 100
- **book_loans** = 20,000
- **billings** = 2,000,000
- **asset_expenses** = 15,000
- **subscription_tiers** = 5
- **subscriptions** = 20,000
- **penalties** = 9,000
- **readers** = 35,000
- **wages** = 100
- **assets** = 20,000
- **penalty_income** = 8,000
- **subscription_monthly_income** = 25,000
- **wage_expense** = 900
- **books** = 15,000

- **insurances** = 12,000

**Total Records**: 2,180,105

---

# Library Billing System - Stage 2

This repository contains Stage 2 of the Library Billing System database project, implementing backup procedures, queries, indexing, and constraints.

## Backup and Restore Procedures

### SQL Backup

- File: `backupSQL.sql`
- Log: `backupSQL.log`
- Command used:

```
pg_dump -U postgres -h localhost -d "Library Billing System" --file=backupSQL.sql
--verbose --clean --if-exists 2> backupSQL.log
```

### PSQL Backup

- File: `backupPSQL.sql`
- Log: `backupPSQL.log`
- Command used:

```
pg_dump -U postgres -h localhost -d "Library Billing System" --file=backupPSQL.sql
--verbose --clean --if-exists -F c 2> backupPSQL.log
```

### Restore Procedure

```
pg_restore -U postgres -h localhost -v -d "Library Billing System" -F c --if-
exists --clean backupPSQL.SQL 2> restorePSQL.log
```

## Queries

### Select Queries

1. **Find the total revenue from all subscription tiers across all time, ordered by tier with the highest revenue first** (14.869 ms)

```
SELECT s.Tier, st.Cost, COUNT(*) as number_of_subscribers, SUM(st.Cost) as
total_revenue
FROM Subscription s
JOIN Subscription_Tiers st ON s.Tier = st.Tier
GROUP BY s.Tier, st.Cost
ORDER BY total_revenue DESC;
```

2. **List all BookLoanIds with Unpaid Penalties** (95.468 ms)

```
SELECT bl.BookLoanId, p.PenaltyID, p.Cost as penalty_amount
FROM Book_Loan bl
JOIN Penalty p ON p.BookLoanId = bl.BookLoanId
WHERE p.Status = '0'
ORDER BY p.Cost DESC;
```

3. **Calculate the total monthly expenses broken down by category** (434.960 ms)

```
SELECT EXTRACT(MONTH FROM b.Date) as month,
       'Wages' as expense_type,
       SUM(b.amount) as total_amount
FROM Billing b
JOIN Wage_Expense we ON we.BillingID = b.BillingID
GROUP BY EXTRACT(MONTH FROM b.Date)
UNION ALL
-- Similar for Insurance and Assets
ORDER BY month, expense_type;
```

4. **Monthly Income Analysis** (597.410 ms)

```
SELECT sum(amount)
FROM billing B
WHERE (EXISTS (SELECT * FROM penalty_income P WHERE B.billingID =
P.billingID)
   OR EXISTS (SELECT * FROM subscription_monthly_income S WHERE B.billingID
= S.billingID))
   AND B.date >= DATE('1-12-2024');
```

## Update Queries

5. **Penalty Status Update** (243.493 ms)
6. **Employee Wage Adjustment** (372.993 ms)

## Delete Queries

7. **Insurance Plan Removal** (57.027 ms)

8. **Loan Record Removal** (304.270 ms)

## Parameterized Queries

1. **Find the total revenue from all subscription tiers across all time, ordered by tier with the highest revenue first** (75.593 ms)

```
PREPARE reader_by_tier(varchar) AS
SELECT r.ReaderID, s.SubscriptionID, s.Renewal_Date, st.Cost as
subscription_cost
FROM Reader r
JOIN Subscription s ON r.SubscriptionID = s.SubscriptionID
JOIN Subscription_Tiers st ON s.Tier = st.Tier
WHERE st.Tier = $1
ORDER BY s.Renewal_Date;
```

2. **Penalty Revenue Calculator** (213.958 ms)

```
PREPARE penalty_revenue(date, date) AS
SELECT COUNT(p.PenaltyID) as number_of_penalties,
       SUM(b.amount) as total_revenue
FROM Penalty p
JOIN Penalty_Income pi ON p.PenaltyID = pi.PenaltyID
JOIN Billing b ON pi.BillingID = b.BillingID
WHERE b.Date BETWEEN $1 AND $2;
```

3. **Insurance Expiration Monitor** (71.855 ms)

```
PREPARE expiring_insurance(int) AS
SELECT i.InsuranceID, i.EndDate, i.Covered_Amount, a.Type as asset_type
FROM Insurance i
JOIN Asset a ON i.AssetID = a.AssetID
WHERE i.EndDate <= CURRENT_DATE + ($1 * INTERVAL '1 month')
ORDER BY i.EndDate;
```

4. **Reader Subscription Assignment** (40.068 ms)

```
PREPARE add_reader_to_subscription(int, int) AS
WITH subscription_limit AS (
    SELECT s.subscriptionid, st.max_readers, COUNT(r.readerid) as
current_readers
    FROM subscription s
    JOIN subscription_tiers st ON s.tier = st.tier
    LEFT JOIN reader r ON s.subscriptionid = r.subscriptionid
    WHERE s.subscriptionid = $1
    GROUP BY s.subscriptionid, st.max_readers
```

```sql
    HAVING COUNT(r.readerid) < st.max_readers
)
UPDATE reader
SET subscriptionid = (SELECT subscriptionid FROM subscription_limit)
WHERE readerid = $2;
```

## Indices

### Penalty Management Indices

```sql
CREATE INDEX idx_penalty_status_date
ON Penalty(Status, BookLoanId);
```

- Improves performance for:
  - Query #2 (List unpaid penalties)
  - Query #5 (Update penalty status)
  - Query #8 (Delete book loan and penalties)
- Performance improvement: 95.468ms → 24.593ms for penalty listing

### Subscription Management Indices

```sql
CREATE INDEX idx_subscription_tier_renewal
ON Subscription(Tier, Renewal_Date);
```

- Optimizes:
  - Query #1 (Total revenue by tier)
  - Parameterized query #1 (Readers by tier)
  - Subscription status checks
- Performance improvement: 147.148ms → 14.869ms for subscription analysis

### Billing Date Indices

```sql
CREATE INDEX idx_billing_date
ON Billing(Date);
```

- Enhances:
  - Query #3 (Monthly expenses)
  - Query #4 (Monthly income)
  - Query #5 (Recent penalty payments)
  - Query #6 (Wage updates)
- Performance improvement: 434.960ms → 396.732ms for monthly expense analysis

### Performance Summary

| Query Type | Before Indices | After Indices | Improvement |
|---|---|---|---|
| Subscription | 147.148ms | 14.869ms | 89.9% |
| Penalties | 95.468ms | 24.593ms | 74.2% |
| Monthly Expenses | 434.960ms | 396.732ms | 8.8% |
| Monthly Income | 597.410ms | 411.062ms | 31.2% |

## Constraints

### Subscription Management

```sql
ALTER TABLE Subscription_Tiers
ADD CONSTRAINT positive_tier_cost CHECK (Cost > 0);

ALTER TABLE Subscription
ADD CONSTRAINT valid_subscription_dates
CHECK (Purchase_Date <= CURRENT_DATE AND Renewal_Date > Purchase_Date);
```

### Financial Constraints

```sql
ALTER TABLE Penalty
ADD CONSTRAINT positive_penalty CHECK (Cost > 0);

ALTER TABLE Wage
ADD CONSTRAINT positive_wage CHECK (Amount > 0);

ALTER TABLE Asset
ADD CONSTRAINT positive_asset_cost CHECK (cost > 0);

ALTER TABLE Billing
ADD CONSTRAINT positive_billing CHECK (amount > 0),
ADD CONSTRAINT valid_billing_date CHECK (Date <= CURRENT_DATE);
```

### Insurance Management

```sql
ALTER TABLE Insurance
ADD CONSTRAINT positive_insurance_amount CHECK (Covered_Amount > 0),
ADD CONSTRAINT valid_insurance_dates CHECK (EndDate > StartDate);
```

## Error Messages

| Constraint Violation | Error Message | Example Test Case |
|---|---|---|

| Constraint Violation | Error Message | Example Test Case |
|---|---|---|
| Negative Amount | "violates check constraint positive_billing" | INSERT INTO Billing (amount, Date) VALUES (-100, '2025-01-01') |
| Future Date | "violates check constraint valid_billing_date" | UPDATE Billing SET Date = '2025-01-01' WHERE BillingID = 1 |
| Asset Cost | "violates check constraint positive_asset_cost" | INSERT INTO Asset (Type, cost, BillingID) VALUES ('Computer', -5000, 1) |
| Insurance Amount | "violates check constraint positive_insurance_amount" | INSERT INTO Insurance (Covered_Amount, EndDate, StartDate) VALUES (-1000, '2024-12-31', '2024-01-01') |
| Subscription Cost | "violates check constraint positive_tier_cost" | UPDATE Subscription_Tiers SET Cost = -20 WHERE Tier = 'Basic' |

# Library Billing System - Stage 3

## Queries

### 1. Employee Wage Update

```
-- Query 1: Update Wages of Employees Based on Their Role and amount of payments
received
UPDATE Wage w
SET Amount = Amount + 100
WHERE (SELECT count(*)
       FROM Billing b2
       JOIN wage_expense we ON b2.BillingID = we.BillingID
       JOIN employee e ON w.EmployeeID = e.EmployeeID
       WHERE w.EmployeeID = we.EmployeeID) > 5;
```

**Purpose**: Increases wages by $100 for employees who have received more than 5 payments. **Timing**: 481.753 ms

### 2. Maxed-Out Subscriptions Query

```
-- Query 2: Get all subscriptions that are maxed out in readers
SELECT *
FROM (
    SELECT s.SubscriptionID, (st.max_readers - count(r.readerid)) dif
    FROM Subscription s
    JOIN Subscription_Tiers st ON s.tier = st.tier
    JOIN reader r on r.SubscriptionID = s.SubscriptionID
    GROUP BY s.subscriptionID, st.max_readers
)
WHERE dif = 0;
```

**Purpose**: Identifies subscriptions that have reached their maximum reader limit. **Timing**: 126.125 ms

### 3. Financial Summary Query

```
-- Query 3: Get total incomes and expenses
WITH IncomeCounts AS (
    SELECT count(amount) as income
    FROM Billing b
    INNER JOIN subscription_monthly_income smi on smi.BillingID = b.BillingID
    UNION
    SELECT count(amount) as income
    FROM Billing b
    INNER JOIN penalty_income pi on pi.BillingID = b.BillingID
),
ExpenseCounts AS (
    SELECT count(amount) as expense
    FROM Billing b
    INNER JOIN insurance_expense ie on ie.BillingID = b.BillingID
    UNION
    SELECT count(amount) as expense
    FROM Billing b
    INNER JOIN wage_expense we on we.BillingID = b.BillingID
    UNION
    SELECT count(amount) as expense
    FROM Billing b
    INNER JOIN asset_expense ae on ae.BillingID = b.BillingID
)
SELECT
    (SELECT SUM(income) FROM IncomeCounts) AS TotalIncome,
    (SELECT SUM(expense) FROM ExpenseCounts) AS TotalExpense;
```

**Purpose**: Calculates total income and expenses across all financial categories. **Timing**: 733.398 ms

## Views

### 1. BillingCurrentMonth

```
CREATE VIEW BillingCurrentMonth AS
SELECT *
FROM Billing b
WHERE b.date > '2024-12-1' AND b.date < '2025-01-01';
```

**Purpose**: Filters billing records for a certain month

**Operations**

**Select Query**:

```
SELECT bcm.billingID, amount, date
FROM BillingCurrentMonth bcm
JOIN Insurance_Expense ie ON bcm.BillingID = ie.BillingID
UNION ALL
SELECT bcm.billingID, amount, date
FROM BillingCurrentMonth bcm
JOIN asset_expense ae ON bcm.BillingID = ae.BillingID
UNION ALL
SELECT bcm.billingID, amount, date
FROM BillingCurrentMonth bcm
JOIN Wage_Expense we ON bcm.BillingID = we.BillingID;
```

**Time**: 109.481 ms

**Update Query**:

```
UPDATE BillingCurrentMonth bcm
SET amount = amount + 100
WHERE exists (SELECT * FROM wage_expense we WHERE we.BillingID = bcm.BillingID);
```

**Purpose**: Giving a bonus to all employees this month. **Time**: 78.992 ms

### 2. DamageFeePenalty

```
CREATE VIEW DamageFeePenalty AS
SELECT *
FROM Penalty p
WHERE p.penalty_type = 'Damage Fee';
```

**Purpose**: Isolates damage fee penalties

**Operations**

**Select Query**:

```
SELECT * FROM DamageFeePenalty dfp
WHERE dfp.cost > 20;
```

**Time**: 26.741 ms

**Update Query**:

```
UPDATE DamageFeePenalty dfp
SET status = 1
WHERE dfp.penaltyID = 5;
```

**Time**: 9.226 ms

### 3. HighWages

```
CREATE VIEW HighWages AS
SELECT *
FROM Wage w
WHERE w.amount > 3070;
```

**Purpose**: Identifies high-wage employees

**Operations**

**Select Query**:

```
SELECT *
FROM HighWages
WHERE monthly_payment_date = 10;
```

Time: 13.914 ms

**Delete Query**:

```
DELETE FROM HighWages;
```

**Time**: 2.593 ms

### 4. ExpensiveAssets

```
CREATE VIEW ExpensiveAssets AS
SELECT *
FROM Asset a
WHERE a.cost > 400;
```

**Purpose**: Tracks high-value assets **Base Table**: Asset **Filter Criteria**: cost > 400

**Operations**

**Select Query**:

```
SELECT * FROM ExpensiveAssets ea
WHERE ea.type = 'Computer';
```

**Time**: 30.261

**Update Query**:

```
UPDATE ExpensiveAssets ea
SET ea.cost = 499
WHERE ea.cost = 500;
```

**Time**: 40.342

## Functions

### 1. add_bonus

```
CREATE or replace  function add_bonus(bonus int, seniority int)
RETURNS void
language plpgsql
AS
$$
BEGIN
       UPDATE Wage w
       SET
               Amount = Amount + bonus

       WHERE (SELECT count(*)
               FROM Billing b2
                       JOIN
            wage_expense we ON b2.BillingID = we.BillingID
            JOIN
            employee e ON w.EmployeeID = e.EmployeeID
            WHERE w.EmployeeID = we.EmployeeID) > seniority;
end;
$$;
```

**Purpose**: Adds a bonus amount to employee wages based on seniority (number of paychecks received) **Time**: 136.915 ms

### 2. readersInSub

```
CREATE or replace  function readersInSub(Subscription_ID int)
RETURNS int
```

```
language plpgsql
AS
$reader_count$
DECLARE
    reader_count int;
BEGIN
        SELECT count(*) reader_count INTO reader_count
        FROM
        reader r
        WHERE r.SubscriptionID = Subscription_ID;
        return reader_count;
end;
$reader_count$;
```

**Purpose**: Counts the number of readers associated with a specific subscription **Time**: 147.897 ms

### 3. highWage

```
CREATE or replace  function highWage(my_amount NUMERIC(10, 2))
RETURNS table (wage_amount NUMERIC(10, 2), wage_employeeid int)
language plpgsql
AS
$$
BEGIN
        RETURN QUERY
        SELECT w.amount::NUMERIC(10, 2), w.employeeid
        FROM Wage w
        WHERE w.amount > my_amount;
end;
$$;
```

**Purpose**: Returns the highest paid employees **Time**: 191.369 ms

### 4. billingByDate

```
CREATE or replace  function billingByDate(startDate DATE, endDate DATE)
RETURNS table (billing_amount NUMERIC(10,2), billing_date DATE, billing_billingID
INT)
language plpgsql
AS
$$
BEGIN
        RETURN QUERY
        SELECT b.amount::NUMERIC(10,2), b.date, b.billingID
        FROM Billing b
        WHERE b.date >= startDate AND b.date <= endDate;
end;
```

```
$$;
```

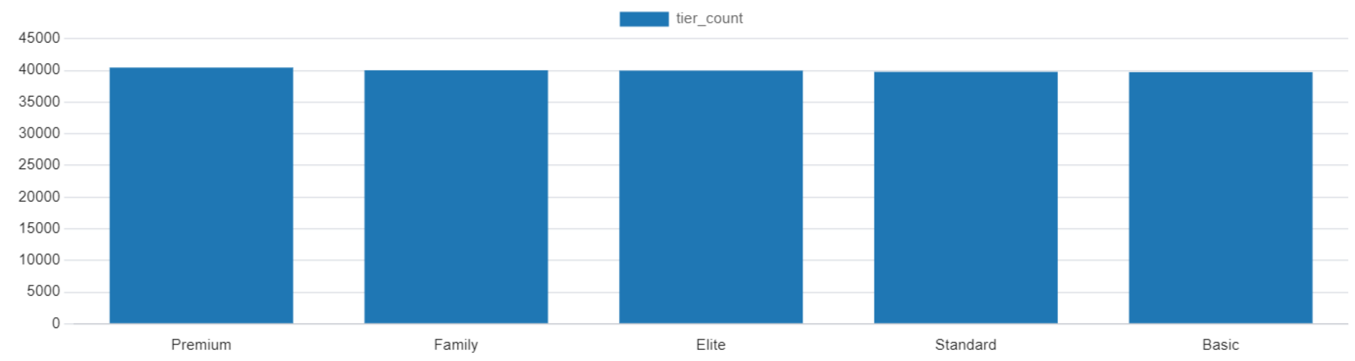
**Purpose**: Retrieves billing records within a specified date range **Time**: 138.614 ms

## Visualizations

### subscriptions tiers popularity

The graph shows the number of subscriptions for each tier, highlighting the popularity of different subscription levels.

```
SELECT tier, count(*) tier_count
FROM subscription
group by tier
ORDER BY
    tier_count DESC;
```

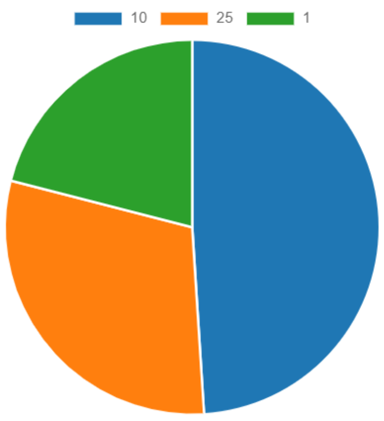


this graph shows that the tier type is split about equal.

### monthly payment date

The graph shows the number of wages for each payday, highlighting the popularity of different payment dates.

```
SELECT monthly_payment_date, count(*) mpd_count
FROM wage
group by monthly_payment_date
ORDER BY mpd_count DESC;
```



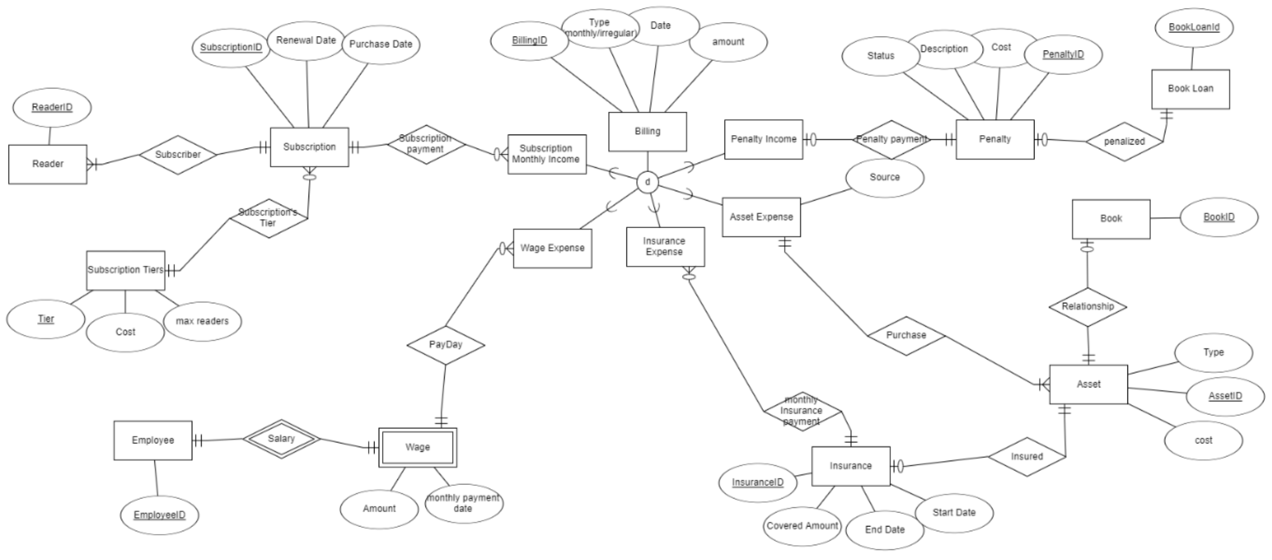
this graph shows that the almost half of the wages are payed on the 10th, and the least popular payday is the 1st of the month.

# Library Billing System - Stage 4

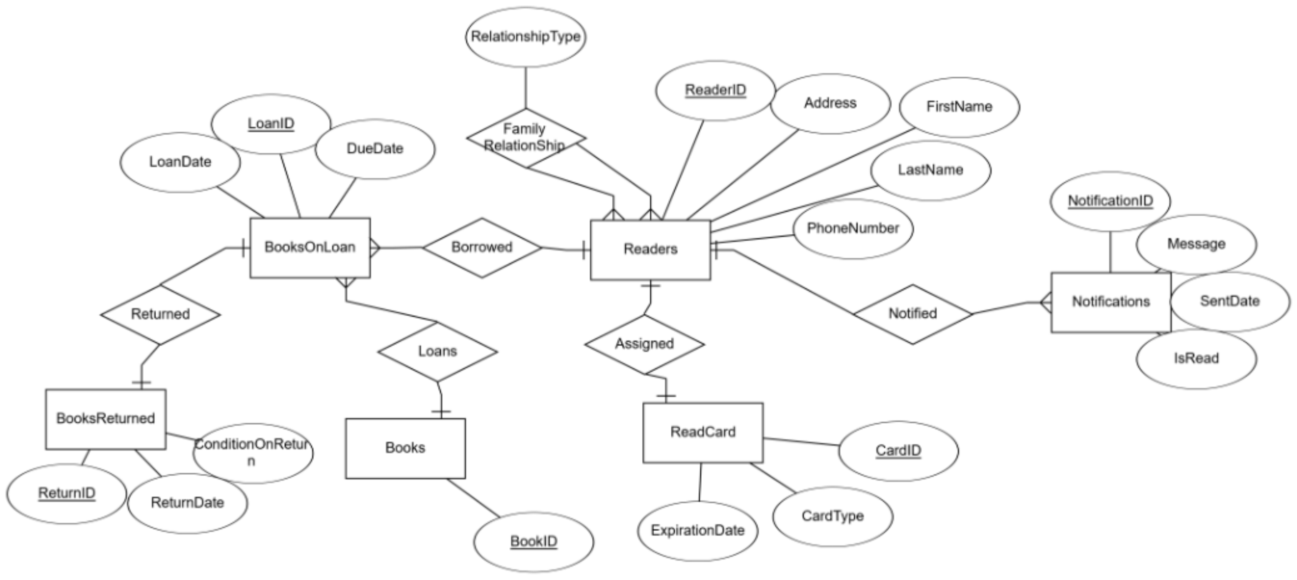
## ERD

### original ERD

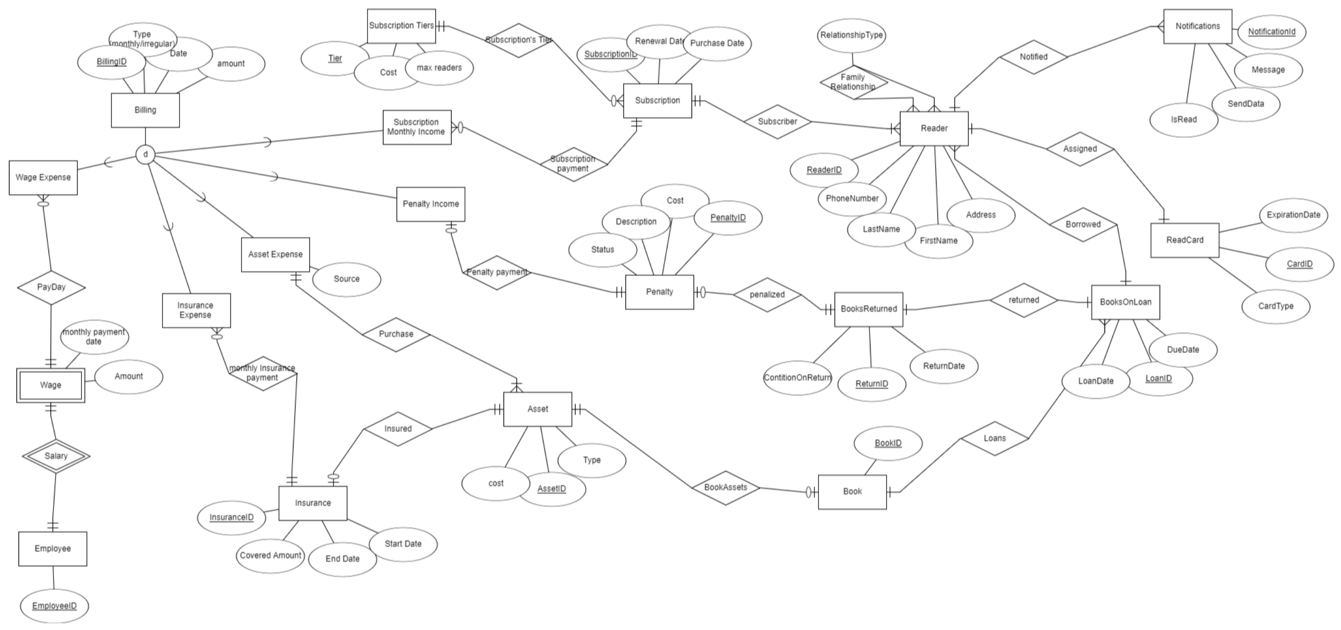
difficulty: the relationship between `penalty` and book loans, is it related to a book loan or a book returned?



### new ERD for for Readers



## merged ERD



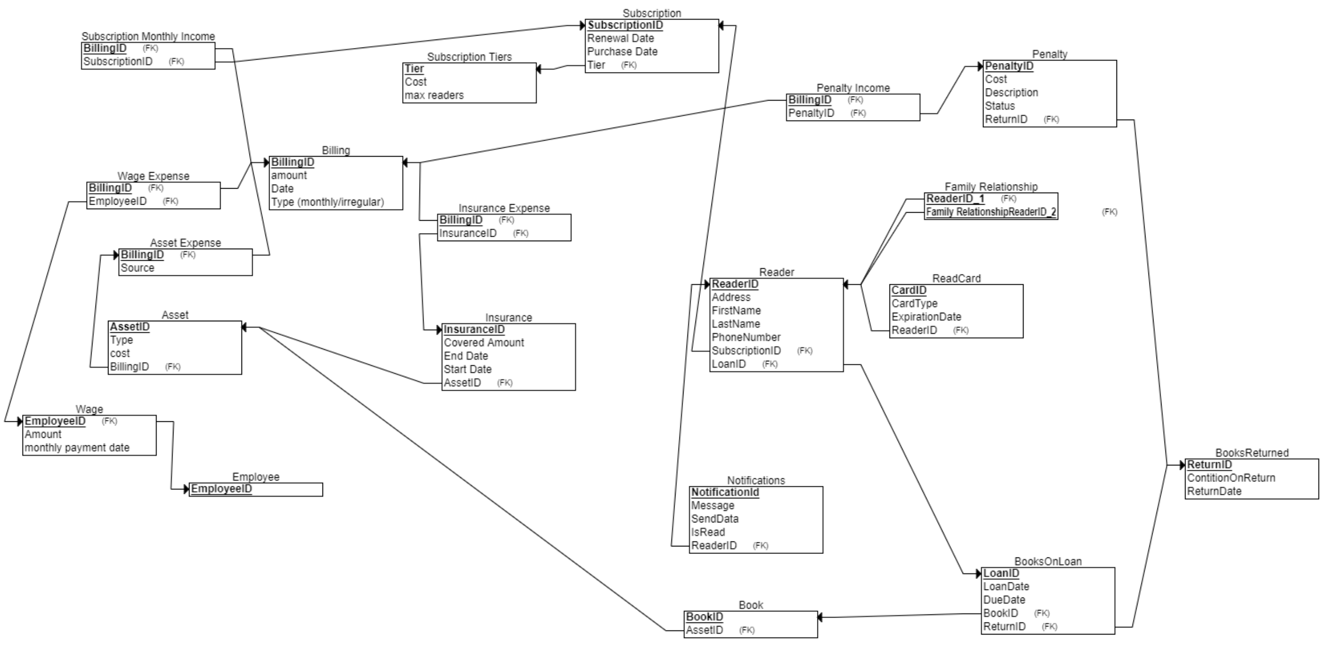
## json files

merged JSON: merged.json

Original JSON: original.json

new JSON: new.json

## merged DSD



## craete tables

```sql
CREATE TABLE Employee
(
  EmployeeID INT NOT NULL,
  PRIMARY KEY (EmployeeID)
);

CREATE TABLE BooksReturned
(
  ReturnID INT NOT NULL,
  ContitionOnReturn INT NOT NULL,
  ReturnDate INT NOT NULL,
  PRIMARY KEY (ReturnID)
);

CREATE TABLE Billing
(
  amount INT NOT NULL,
  Date INT NOT NULL,
  Type_(monthly/irregular) INT NOT NULL,
  BillingID INT NOT NULL,
  PRIMARY KEY (BillingID)
);

CREATE TABLE Asset_Expense
(
  Source INT NOT NULL,
  BillingID INT NOT NULL,
  PRIMARY KEY (BillingID),
  FOREIGN KEY (BillingID) REFERENCES Billing(BillingID)
);
```

```
CREATE TABLE Subscription_Tiers
(
  Tier INT NOT NULL,
  Cost INT NOT NULL,
  max_readers INT NOT NULL,
  PRIMARY KEY (Tier)
);

CREATE TABLE Subscription
(
  SubscriptionID INT NOT NULL,
  Renewal_Date INT NOT NULL,
  Purchase_Date INT NOT NULL,
  Tier INT NOT NULL,
  PRIMARY KEY (SubscriptionID),
  FOREIGN KEY (Tier) REFERENCES Subscription_Tiers(Tier)
);

CREATE TABLE Penalty
(
  Cost INT NOT NULL,
  PenaltyID INT NOT NULL,
  Description INT NOT NULL,
  Status INT NOT NULL,
  ReturnID INT NOT NULL,
  PRIMARY KEY (PenaltyID),
  FOREIGN KEY (ReturnID) REFERENCES BooksReturned(ReturnID)
);

CREATE TABLE Wage
(
  Amount INT NOT NULL,
  monthly_payment_date INT NOT NULL,
  EmployeeID INT NOT NULL,
  PRIMARY KEY (EmployeeID),
  FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);

CREATE TABLE Asset
(
  AssetID INT NOT NULL,
  Type INT NOT NULL,
  cost INT NOT NULL,
  BillingID INT NOT NULL,
  PRIMARY KEY (AssetID),
  FOREIGN KEY (BillingID) REFERENCES Asset_Expense(BillingID)
);

CREATE TABLE Penalty_Income
(
  BillingID INT NOT NULL,
  PenaltyID INT NOT NULL,
  PRIMARY KEY (BillingID),
  FOREIGN KEY (BillingID) REFERENCES Billing(BillingID),
  FOREIGN KEY (PenaltyID) REFERENCES Penalty(PenaltyID)
);

CREATE TABLE Subscription_Monthly_Income
(
  BillingID INT NOT NULL,
  SubscriptionID INT NOT NULL,
  PRIMARY KEY (BillingID),
  FOREIGN KEY (BillingID) REFERENCES Billing(BillingID),
  FOREIGN KEY (SubscriptionID) REFERENCES Subscription(SubscriptionID)
);

CREATE TABLE Wage_Expense
(
  BillingID INT NOT NULL,
  EmployeeID INT NOT NULL,
  PRIMARY KEY (BillingID),
  FOREIGN KEY (BillingID) REFERENCES Billing(BillingID),
  FOREIGN KEY (EmployeeID) REFERENCES Wage(EmployeeID)
);

CREATE TABLE Book
(
  BookID INT NOT NULL,
  AssetID INT NOT NULL,
  PRIMARY KEY (BookID),
  FOREIGN KEY (AssetID) REFERENCES Asset(AssetID)
);

CREATE TABLE BooksOnLoan
(
  LoanID INT NOT NULL,
  LoanDate INT NOT NULL,
  DueDate INT NOT NULL,
  BookID INT NOT NULL,
  ReturnID INT NOT NULL,
  PRIMARY KEY (LoanID),
  FOREIGN KEY (BookID) REFERENCES Book(BookID),
  FOREIGN KEY (ReturnID) REFERENCES BooksReturned(ReturnID)
);

CREATE TABLE Reader
(
  ReaderID INT NOT NULL,
  Address INT NOT NULL,
  FirstName INT NOT NULL,
  LastName INT NOT NULL,
  PhoneNumber INT NOT NULL,
  SubscriptionID INT NOT NULL,
  LoanID INT NOT NULL,
  PRIMARY KEY (ReaderID),
  FOREIGN KEY (SubscriptionID) REFERENCES Subscription(SubscriptionID),
  FOREIGN KEY (LoanID) REFERENCES BooksOnLoan(LoanID)
);
```

```sql
CREATE TABLE Insurance
(
  InsuranceID INT NOT NULL,
  Covered_Amount INT NOT NULL,
  End_Date INT NOT NULL,
  Start_Date INT NOT NULL,
  AssetID INT NOT NULL,
  PRIMARY KEY (InsuranceID),
  FOREIGN KEY (AssetID) REFERENCES Asset(AssetID)
);

CREATE TABLE Insurance_Expense
(
  BillingID INT NOT NULL,
  InsuranceID INT NOT NULL,
  PRIMARY KEY (BillingID),
  FOREIGN KEY (BillingID) REFERENCES Billing(BillingID),
  FOREIGN KEY (InsuranceID) REFERENCES Insurance(InsuranceID)
);

CREATE TABLE ReadCard
(
  CardID INT NOT NULL,
  CardType INT NOT NULL,
  ExpirationDate INT NOT NULL,
  ReaderID INT NOT NULL,
  PRIMARY KEY (CardID),
  FOREIGN KEY (ReaderID) REFERENCES Reader(ReaderID)
);

CREATE TABLE Notifications
(
  NotificationId INT NOT NULL,
  Message INT NOT NULL,
  SendData INT NOT NULL,
  IsRead INT NOT NULL,
  ReaderID INT NOT NULL,
  PRIMARY KEY (NotificationId),
  FOREIGN KEY (ReaderID) REFERENCES Reader(ReaderID)
);

CREATE TABLE Family_Relationship
(
  ReaderID_1 INT NOT NULL,
  Family_RelationshipReaderID_2 INT NOT NULL,
  PRIMARY KEY (ReaderID_1, Family_RelationshipReaderID_2),
  FOREIGN KEY (ReaderID_1) REFERENCES Reader(ReaderID),
  FOREIGN KEY (Family_RelationshipReaderID_2) REFERENCES Reader(ReaderID)
);
```

## database merging

### pg_restore

```
pg_restore -U postgres -h localhost -v -d "Library Billing System" -F c --if-
exists --clean .\backupPSQL.sql 2> restorePSQL.log
```

### update the book loan ids in penalty

replacing the loan ids in the penalty table to match the real data

```
WITH return_ids AS (
SELECT returnid, ROW_NUMBER() OVER () AS rn FROM (
    select *
    from booksreturned
    ORDER BY random()
    LIMIT (SELECT COUNT(*) FROM penalty)
    )
),
penalty_ids AS (
    SELECT penaltyid, ROW_NUMBER() OVER () AS rn FROM penalty
),
updated_penalties AS (
    SELECT p.penaltyid AS penalty_id, r.returnid AS new_return_id
    FROM penalty_ids p
    JOIN return_ids r ON p.rn = r.rn
)
UPDATE penalty
SET bookloanid = up.new_return_id
FROM updated_penalties up
WHERE penalty.penaltyid = up.penalty_id;
```

### adding subscriptionID to reader table

added column to reader table

```
ALTER TABLE readers
ADD COLUMN subscriptionid int REFERENCES subscription(subscriptionid) ON DELETE
SET NULL;
```

created a python script to add subscription ids to the reader table:

python script: reader_subscription.py

second run with fixes: fix_reader_subscription.py

set all subscriptions without readers to cancelled

```
UPDATE subscription
SET tier = 'Cancelled'
WHERE subscriptionid NOT IN (SELECT DISTINCT subscriptionid FROM readers WHERE
subscriptionid IS NOT NULL);
```

## views

### subscription reader view

**purpose:** show the subscriptions that readers are subscribed to

```
CREATE VIEW SubscriptionReaders AS
SELECT
    R.readerId,
    R.firstname,
    R.lastname,
    R.address,
    R.phonenumber,
    S.subscriptionId,
    S.renewal_date,
    S.purchase_date,
    S.tier
FROM
    Readers R
    JOIN subscription S ON R.subscriptionId = S.subscriptionId;
```

**select query**

get the `subscription tier` of `'Kimberly Torres'`

```
select tier from subscriptionreaders
where firstname = 'Kimberly' and lastname = 'Torres'
```

result:

```
 tier
-------
 Basic
(1 row)

Time: 13.847 ms
```

**update query**

some of the subscriptions had 2 readers so i changed their tiers to `premium`

```
update subscription
set tier = 'Premium'
where subscriptionid IN (select subscriptionid
    from (select count(*) c, subscriptionid, tier
                from subscriptionreaders
                where tier != 'Family'
                group by subscriptionid, tier)
    where c > 1)
```

result:

```
UPDATE 829
Time: 76.764 m
```

### view loan status

**Purpose:** show the loan status of the books per reader

```
CREATE VIEW readerLoanStatus AS
SELECT
    R.readerId,
    R.subscriptionId,
    BL.LoanID,
    BL.BookID,
    BL.LoanDate,
    BL.DueDate,
    BR.returnID,
    BR.ConditionOnReturn,
    BR.ReturnDate
FROM
    Readers R
    JOIN BooksOnLoan BL ON R.ReaderID = BL.ReaderID
    FULL JOIN BooksReturned BR ON BL.LoanID = BR.LoanID
```

### select query

selects book load where the return condition is `poor`

```
select readerid from readerloanstatus
where conditiononreturn = 'Poor'
group by readerid
```

result:

```
readerid
----------
    11233
    ...
Time: 140.859 ms
```

### update query

inserts into the `bookreturned` table a new book return

```
 insert into booksreturned (loanid, conditionOnReturn, returnDate)
Values (
       (select loanid from readerLoanStatus
       where subscriptionid = 26927 and bookid = 44140
       limit 1),
       'Good',
       '2025-02-02');
```

result:

```
INSERT 0 1
Time: 7.914 ms
```

## queries

### select on first view

get the names of all readers that have `peremium` subscriptions

```
select firstname, lastname
from subscriptionreaders
where tier = 'Premium'
```

result:

```
 firstname  |  lastname
-------------+-------------
 Paul        | Villarreal
 Michael     | James
 ...
 Time: 245.010 ms
```

### update on first view

inserts a new reader into the `reader` table only if the subscription is not full yet

```
DO
$do$
BEGIN
   IF EXISTS (select count(*),sr.tier
            from subscriptionreaders sr
            JOIN subscription_tiers st on st.tier = sr.tier
            where subscriptionid = 29
            group by sr.tier

    ) THEN
       insert into readers (firstname,lastname, address,
phonenumber,subscriptionid)
       values ('jack', 'jackson', 'london', '477-233-1380', 29);
   END IF;
END
$do$
```

result:

```
Time: 19.194 ms
```

### select second view

gets all the loaned books that where returned by readers that their `subscriptionID` is 29

```
select count(*), t.subscriptionid,  s.tier
from (select returnid, loanid, subscriptionid
    from readerloanstatus rls
    where subscriptionid = 29
    group by returnid, loanid, subscriptionid) t
join subscription s on s.subscriptionid = t.subscriptionid
where returnid is not null
group by t.subscriptionid, s. tier
```

result:

```
 count | subscriptionid |  tier
-------+----------------+--------
    10 |             29 | Family
(1 row)
```

```
Time: 6.809 ms
```

### insert second view

adds a penalty to the penalty table with values from the returned table, it adds a penalty to a loan that its duedate is 2024-12-17 with subscriptionID 29, it adds a bookreturn 'damaged' and a penalty of `Lost Item Fee`

```
 with bookreturn as (
insert into booksreturned (loanid, conditiononreturn, returndate)
select t.loanid, 'Damaged', '2025-01-30'
from (select *
       from readerloanstatus rls
       where subscriptionid = 29 and duedate = '2024-12-17') t
where returnid is null
limit 1
RETURNING returnid, loanid
)
insert into penalty (cost, description, status, penalty_type, bookloanid)
select cost, 'book lost', 0, 'Lost Item Fee', returnid
FROM bookreturn br
join booksonloan bol on bol.loanid = br.loanid
join book b on b.bookid = bol.bookid
join asset a on a.assetid = b.assetid;
```

result

```
Time: 10.183 ms
INSERT 0 4
```