Database Schema Design

Library Book Loan System

Document Information

- **Document Version**: 1.0
- **Date**: July 13, 2025
- **Project**: Library Book Loan Automation System
- **Database Type**: PostgreSQL (Primary)
- **Status**: Draft

---

1. Executive Summary

This document defines the complete database schema for the Library Book Loan System, including all tables, relationships, constraints, indexes, and data integrity rules necessary to support the automated loan processing workflow.

---

2. Database Design Principles

2.1 Design Philosophy

- **Normalization**: Third Normal Form (3NF) for data integrity
- **Performance**: Strategic denormalization where needed
- **Scalability**: Designed for horizontal scaling
- **Auditability**: Complete audit trails for all transactions
- **Data Integrity**: Comprehensive constraints and validation

2.2 Naming Conventions

- **Tables**: Singular nouns in snake_case (e.g., `book`, `member`)
- **Columns**: Descriptive names in snake_case (e.g., `first_name`, `due_date`)
- **Primary Keys**: `id` for single-column primary keys
- **Foreign Keys**: `[table_name]_id` format (e.g., `member_id`)
- **Indexes**: `idx_[table]_[column(s)]` format
- **Constraints**: `[type]_[table]_[column]` format

---

3. Core Entity Tables

3.1 Member Table

```sql
CREATE TABLE member (
    id SERIAL PRIMARY KEY,
    member_id VARCHAR(20) UNIQUE NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(20),
    address_line1 VARCHAR(255),
    address_line2 VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(50),
    postal_code VARCHAR(20),
    country VARCHAR(50) DEFAULT 'USA',
    date_of_birth DATE,
    membership_type_id INTEGER REFERENCES membership_type(id),
    membership_start_date DATE NOT NULL,
    membership_end_date DATE NOT NULL,
    status VARCHAR(20) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE', 'EXPIRED', 'SUSPENDED', 'CANCEL
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by INTEGER REFERENCES staff_user(id),
    updated_by INTEGER REFERENCES staff_user(id)
);

-- Indexes
CREATE INDEX idx_member_member_id ON member(member_id);
CREATE INDEX idx_member_email ON member(email);
CREATE INDEX idx_member_status ON member(status);
CREATE INDEX idx_member_membership_type ON member(membership_type_id);
CREATE INDEX idx_member_membership_dates ON member(membership_start_date, membership_end
```

3.2 Book Table

```sql
CREATE TABLE book (
    id SERIAL PRIMARY KEY,
    isbn VARCHAR(20) UNIQUE,
    isbn13 VARCHAR(20) UNIQUE,
    title VARCHAR(500) NOT NULL,
    subtitle VARCHAR(500),
    author VARCHAR(255) NOT NULL,
    co_authors TEXT,
    publisher VARCHAR(255),
    publication_date DATE,
    edition VARCHAR(50),
    language VARCHAR(50) DEFAULT 'English',
    pages INTEGER,
    category_id INTEGER REFERENCES book_category(id),
    description TEXT,
    keywords TEXT,
    dewey_decimal VARCHAR(20),
    location_code VARCHAR(50),
    status VARCHAR(20) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE', 'INACTIVE', 'DAMAGED', 'LOST', 'W
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by INTEGER REFERENCES staff_user(id),
    updated_by INTEGER REFERENCES staff_user(id)
);

-- Indexes
CREATE INDEX idx_book_isbn ON book(isbn);
CREATE INDEX idx_book_isbn13 ON book(isbn13);
CREATE INDEX idx_book_title ON book USING gin(to_tsvector('english', title));
CREATE INDEX idx_book_author ON book USING gin(to_tsvector('english', author));
CREATE INDEX idx_book_category ON book(category_id);
CREATE INDEX idx_book_status ON book(status);
CREATE INDEX idx_book_publication_date ON book(publication_date);
```

3.3 Book Copy Table

```sql
CREATE TABLE book_copy (
    id SERIAL PRIMARY KEY,
    book_id INTEGER NOT NULL REFERENCES book(id) ON DELETE CASCADE,
    copy_number VARCHAR(50) NOT NULL,
    barcode VARCHAR(50) UNIQUE NOT NULL,
    condition VARCHAR(20) DEFAULT 'GOOD' CHECK (condition IN ('EXCELLENT', 'GOOD', 'FAIR', 'POOR',
    status VARCHAR(20) DEFAULT 'AVAILABLE' CHECK (status IN ('AVAILABLE', 'CHECKED_OUT', 'RESERVE
    location_code VARCHAR(50),
    acquisition_date DATE DEFAULT CURRENT_DATE,
    cost DECIMAL(10,2),
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(book_id, copy_number)
);

-- Indexes
CREATE INDEX idx_book_copy_book_id ON book_copy(book_id);
CREATE INDEX idx_book_copy_barcode ON book_copy(barcode);
CREATE INDEX idx_book_copy_status ON book_copy(status);
CREATE INDEX idx_book_copy_location ON book_copy(location_code);
```

3.4 Loan Table

```sql
```

```sql
CREATE TABLE loan (
    id SERIAL PRIMARY KEY,
    loan_number VARCHAR(50) UNIQUE NOT NULL,
    member_id INTEGER NOT NULL REFERENCES member(id),
    book_copy_id INTEGER NOT NULL REFERENCES book_copy(id),
    loan_date DATE NOT NULL DEFAULT CURRENT_DATE,
    due_date DATE NOT NULL,
    return_date DATE,
    renewal_count INTEGER DEFAULT 0,
    max_renewals INTEGER DEFAULT 2,
    status VARCHAR(20) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE', 'RETURNED', 'OVERDUE', 'LOST', 'R
    notes TEXT,
    checked_out_by INTEGER REFERENCES staff_user(id),
    checked_in_by INTEGER REFERENCES staff_user(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes
CREATE INDEX idx_loan_member_id ON loan(member_id);
CREATE INDEX idx_loan_book_copy_id ON loan(book_copy_id);
CREATE INDEX idx_loan_status ON loan(status);
CREATE INDEX idx_loan_due_date ON loan(due_date);
CREATE INDEX idx_loan_loan_date ON loan(loan_date);
CREATE INDEX idx_loan_number ON loan(loan_number);
```

3.5 Fine Table

sql

```sql
CREATE TABLE fine (
    id SERIAL PRIMARY KEY,
    member_id INTEGER NOT NULL REFERENCES member(id),
    loan_id INTEGER REFERENCES loan(id),
    fine_type_id INTEGER NOT NULL REFERENCES fine_type(id),
    amount DECIMAL(10,2) NOT NULL CHECK (amount >= 0),
    description TEXT,
    fine_date DATE NOT NULL DEFAULT CURRENT_DATE,
    due_date DATE,
    status VARCHAR(20) DEFAULT 'UNPAID' CHECK (status IN ('UNPAID', 'PAID', 'WAIVED', 'PARTIALLY_PAI
    payment_date DATE,
    waived_date DATE,
    waived_by INTEGER REFERENCES staff_user(id),
    waived_reason TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by INTEGER REFERENCES staff_user(id)
);

-- Indexes
CREATE INDEX idx_fine_member_id ON fine(member_id);
CREATE INDEX idx_fine_loan_id ON fine(loan_id);
CREATE INDEX idx_fine_status ON fine(status);
CREATE INDEX idx_fine_type ON fine(fine_type_id);
CREATE INDEX idx_fine_due_date ON fine(due_date);
CREATE INDEX idx_fine_amount ON fine(amount);
```

4. Lookup and Reference Tables

4.1 Membership Type Table

```
sql
```

```sql
CREATE TABLE membership_type (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT,
    duration_months INTEGER NOT NULL,
    max_books INTEGER DEFAULT 5,
    loan_period_days INTEGER DEFAULT 14,
    renewal_limit INTEGER DEFAULT 2,
    fine_rate_per_day DECIMAL(5,2) DEFAULT 0.50,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Default membership types
INSERT INTO membership_type (name, description, duration_months, max_books, loan_period_days) VA
('Standard Adult', 'Standard adult membership', 12, 5, 14),
('Student', 'Student membership with extended loan period', 12, 8, 21),
('Senior', 'Senior citizen membership', 12, 5, 21),
('Child', 'Children membership with parental supervision', 12, 3, 10),
('Premium', 'Premium membership with extended privileges', 12, 10, 21);
```

4.2 Book Category Table

```
sql
```

```
CREATE TABLE book_category (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT,
    dewey_range VARCHAR(50),
    parent_category_id INTEGER REFERENCES book_category(id),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Sample categories
INSERT INTO book_category (name, description, dewey_range) VALUES
('Fiction', 'Fiction literature', '800-899'),
('Non-Fiction', 'Non-fiction books', '000-799'),
('Science', 'Science and technology', '500-599'),
('History', 'Historical works', '900-999'),
('Biography', 'Biographical works', '920-929'),
('Children', 'Children literature', 'J'),
('Reference', 'Reference materials', 'REF');
```

4.3 Fine Type Table

```
sql
```

```
CREATE TABLE fine_type (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT,
    default_amount DECIMAL(10,2),
    is_daily_rate BOOLEAN DEFAULT FALSE,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Default fine types
INSERT INTO fine_type (name, description, default_amount, is_daily_rate) VALUES
('Overdue', 'Daily overdue fine', 0.50, TRUE),
('Lost Book', 'Lost book replacement cost', 25.00, FALSE),
('Damaged Book', 'Book damage fee', 10.00, FALSE),
('Processing Fee', 'Administrative processing fee', 5.00, FALSE),
('Late Return', 'One-time late return fee', 2.00, FALSE);
```

4.4 Staff User Table

```
sql
```

```
CREATE TABLE staff_user (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    role VARCHAR(50) NOT NULL CHECK (role IN ('LIBRARIAN', 'ASSISTANT', 'MANAGER', 'ADMIN')),
    password_hash VARCHAR(255) NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    last_login TIMESTAMP,
    failed_login_attempts INTEGER DEFAULT 0,
    account_locked_until TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes
CREATE INDEX idx_staff_username ON staff_user(username);
CREATE INDEX idx_staff_email ON staff_user(email);
CREATE INDEX idx_staff_role ON staff_user(role);
CREATE INDEX idx_staff_active ON staff_user(is_active);
```

5. Transaction and Payment Tables

5.1 Payment Table

```
sql
```

```sql
CREATE TABLE payment (
    id SERIAL PRIMARY KEY,
    payment_number VARCHAR(50) UNIQUE NOT NULL,
    member_id INTEGER NOT NULL REFERENCES member(id),
    amount DECIMAL(10,2) NOT NULL CHECK (amount > 0),
    payment_method VARCHAR(20) NOT NULL CHECK (payment_method IN ('CASH', 'CREDIT_CARD', 'DE
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    transaction_id VARCHAR(100),
    gateway_response TEXT,
    status VARCHAR(20) DEFAULT 'PENDING' CHECK (status IN ('PENDING', 'COMPLETED', 'FAILED', 'REFU
    processed_by INTEGER REFERENCES staff_user(id),
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes
CREATE INDEX idx_payment_member_id ON payment(member_id);
CREATE INDEX idx_payment_status ON payment(status);
CREATE INDEX idx_payment_date ON payment(payment_date);
CREATE INDEX idx_payment_number ON payment(payment_number);
CREATE INDEX idx_payment_transaction_id ON payment(transaction_id);
```

5.2 Fine Payment Table (Junction Table)

sql

```sql
CREATE TABLE fine_payment (
    id SERIAL PRIMARY KEY,
    fine_id INTEGER NOT NULL REFERENCES fine(id),
    payment_id INTEGER NOT NULL REFERENCES payment(id),
    amount_applied DECIMAL(10,2) NOT NULL CHECK (amount_applied > 0),
    applied_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(fine_id, payment_id)
);

-- Indexes
CREATE INDEX idx_fine_payment_fine_id ON fine_payment(fine_id);
CREATE INDEX idx_fine_payment_payment_id ON fine_payment(payment_id);
```

5.3 Reservation Table

```
sql
```

```
CREATE TABLE reservation (
    id SERIAL PRIMARY KEY,
    member_id INTEGER NOT NULL REFERENCES member(id),
    book_id INTEGER NOT NULL REFERENCES book(id),
    reservation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expiry_date DATE NOT NULL,
    status VARCHAR(20) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE', 'FULFILLED', 'EXPIRED', 'CANCELLE
    fulfilled_date TIMESTAMP,
    fulfilled_by INTEGER REFERENCES staff_user(id),
    priority_level INTEGER DEFAULT 1,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes
CREATE INDEX idx_reservation_member_id ON reservation(member_id);
CREATE INDEX idx_reservation_book_id ON reservation(book_id);
CREATE INDEX idx_reservation_status ON reservation(status);
CREATE INDEX idx_reservation_expiry_date ON reservation(expiry_date);
CREATE INDEX idx_reservation_priority ON reservation(priority_level);
```

6. Audit and Logging Tables

6.1 Audit Log Table

```
sql
```

```
CREATE TABLE audit_log (
    id SERIAL PRIMARY KEY,
    table_name VARCHAR(100) NOT NULL,
    record_id INTEGER NOT NULL,
    action VARCHAR(20) NOT NULL CHECK (action IN ('INSERT', 'UPDATE', 'DELETE')),
    old_values JSONB,
    new_values JSONB,
    changed_by INTEGER REFERENCES staff_user(id),
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ip_address INET,
    user_agent TEXT
);

-- Indexes
CREATE INDEX idx_audit_table_record ON audit_log(table_name, record_id);
CREATE INDEX idx_audit_changed_by ON audit_log(changed_by);
CREATE INDEX idx_audit_changed_at ON audit_log(changed_at);
CREATE INDEX idx_audit_action ON audit_log(action);
```

6.2 System Log Table

```
sql
```

```
CREATE TABLE system_log (
    id SERIAL PRIMARY KEY,
    level VARCHAR(20) NOT NULL CHECK (level IN ('DEBUG', 'INFO', 'WARN', 'ERROR', 'FATAL')),
    message TEXT NOT NULL,
    context JSONB,
    source VARCHAR(100),
    user_id INTEGER REFERENCES staff_user(id),
    session_id VARCHAR(100),
    ip_address INET,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes
CREATE INDEX idx_system_log_level ON system_log(level);
CREATE INDEX idx_system_log_timestamp ON system_log(timestamp);
CREATE INDEX idx_system_log_source ON system_log(source);
CREATE INDEX idx_system_log_user_id ON system_log(user_id);
```

---

7. Configuration Tables

7.1 System Configuration Table

```
sql
```

```sql
CREATE TABLE system_config (
    id SERIAL PRIMARY KEY,
    config_key VARCHAR(100) UNIQUE NOT NULL,
    config_value TEXT NOT NULL,
    description TEXT,
    data_type VARCHAR(20) DEFAULT 'STRING' CHECK (data_type IN ('STRING', 'INTEGER', 'DECIMAL', 'BO
    is_encrypted BOOLEAN DEFAULT FALSE,
    updated_by INTEGER REFERENCES staff_user(id),
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Default configuration values
INSERT INTO system_config (config_key, config_value, description, data_type) VALUES
('FINE_THRESHOLD', '10.00', 'Maximum outstanding fine amount for loan approval', 'DECIMAL'),
('DEFAULT_LOAN_PERIOD', '14', 'Default loan period in days', 'INTEGER'),
('MAX_RENEWALS', '2', 'Maximum number of renewals allowed', 'INTEGER'),
('OVERDUE_FINE_RATE', '0.50', 'Daily overdue fine rate', 'DECIMAL'),
('EMAIL_NOTIFICATIONS', 'true', 'Enable email notifications', 'BOOLEAN'),
('SMS_NOTIFICATIONS', 'false', 'Enable SMS notifications', 'BOOLEAN'),
('MAX_BOOKS_PER_MEMBER', '5', 'Maximum books per member', 'INTEGER');
```

7.2 Holiday Table

```
sql
```

```
CREATE TABLE holiday (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    holiday_date DATE NOT NULL,
    is_recurring BOOLEAN DEFAULT FALSE,
    recurrence_pattern VARCHAR(50),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(holiday_date, name)
);

-- Sample holidays
INSERT INTO holiday (name, holiday_date, is_recurring) VALUES
('New Year''s Day', '2025-01-01', TRUE),
('Independence Day', '2025-07-04', TRUE),
('Christmas Day', '2025-12-25', TRUE),
('Thanksgiving', '2025-11-27', FALSE);
```

---

8. Views for Common Queries

8.1 Member Status View

```
sql
```

```
CREATE VIEW v_member_status AS
SELECT
    m.id,
    m.member_id,
    m.first_name,
    m.last_name,
    m.email,
    m.status,
    m.membership_end_date,
    mt.name AS membership_type,
    COALESCE(SUM(f.amount), 0) AS total_outstanding_fines,
    COUNT(l.id) AS active_loans,
    CASE
        WHEN m.status != 'ACTIVE' THEN FALSE
        WHEN m.membership_end_date < CURRENT_DATE THEN FALSE
        WHEN COALESCE(SUM(f.amount), 0) > 10.00 THEN FALSE
        ELSE TRUE
    END AS can_borrow
FROM member m
LEFT JOIN membership_type mt ON m.membership_type_id = mt.id
LEFT JOIN fine f ON m.id = f.member_id AND f.status = 'UNPAID'
LEFT JOIN loan l ON m.id = l.member_id AND l.status = 'ACTIVE'
GROUP BY m.id, m.member_id, m.first_name, m.last_name, m.email, m.status, m.membership_end_date,
```

8.2 Book Availability View

```
sql
```

```
CREATE VIEW v_book_availability AS
SELECT
    b.id AS book_id,
    b.isbn,
    b.title,
    b.author,
    b.publisher,
    COUNT(bc.id) AS total_copies,
    COUNT(CASE WHEN bc.status = 'AVAILABLE' THEN 1 END) AS available_copies,
    COUNT(CASE WHEN bc.status = 'CHECKED_OUT' THEN 1 END) AS checked_out_copies,
    COUNT(CASE WHEN bc.status = 'RESERVED' THEN 1 END) AS reserved_copies,
    CASE
        WHEN COUNT(CASE WHEN bc.status = 'AVAILABLE' THEN 1 END) > 0 THEN TRUE
        ELSE FALSE
    END AS is_available
FROM book b
LEFT JOIN book_copy bc ON b.id = bc.book_id
WHERE b.status = 'ACTIVE'
GROUP BY b.id, b.isbn, b.title, b.author, b.publisher;
```

8.3 Overdue Loans View

```
sql
```

```
CREATE VIEW v_overdue_loans AS
SELECT
    l.id AS loan_id,
    l.loan_number,
    m.member_id,
    m.first_name,
    m.last_name,
    m.email,
    b.title,
    b.author,
    bc.barcode,
    l.loan_date,
    l.due_date,
    CURRENT_DATE - l.due_date AS days_overdue,
    (CURRENT_DATE - l.due_date) * 0.50 AS calculated_fine
FROM loan l
JOIN member m ON l.member_id = m.id
JOIN book_copy bc ON l.book_copy_id = bc.id
JOIN book b ON bc.book_id = b.id
WHERE l.status = 'ACTIVE'
AND l.due_date < CURRENT_DATE;
```

---

9. Triggers and Functions

9.1 Update Timestamps Trigger

```
sql
```

```
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$ language 'plpgsql';

-- Apply to all tables with updated_at column
CREATE TRIGGER update_member_updated_at BEFORE UPDATE ON member
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_book_updated_at BEFORE UPDATE ON book
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_loan_updated_at BEFORE UPDATE ON loan
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_fine_updated_at BEFORE UPDATE ON fine
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

9.2 Audit Trail Trigger

```
sql
```

```
CREATE OR REPLACE FUNCTION audit_trigger_function()
RETURNS TRIGGER AS $
BEGIN
  IF TG_OP = 'DELETE' THEN
    INSERT INTO audit_log (table_name, record_id, action, old_values, changed_at)
    VALUES (TG_TABLE_NAME, OLD.id, TG_OP, row_to_json(OLD), CURRENT_TIMESTAMP);
    RETURN OLD;
  ELSIF TG_OP = 'UPDATE' THEN
    INSERT INTO audit_log (table_name, record_id, action, old_values, new_values, changed_at)
    VALUES (TG_TABLE_NAME, NEW.id, TG_OP, row_to_json(OLD), row_to_json(NEW), CURRENT_TIMES
    RETURN NEW;
  ELSIF TG_OP = 'INSERT' THEN
    INSERT INTO audit_log (table_name, record_id, action, new_values, changed_at)
    VALUES (TG_TABLE_NAME, NEW.id, TG_OP, row_to_json(NEW), CURRENT_TIMESTAMP);
    RETURN NEW;
  END IF;
  RETURN NULL;
END;
$ LANGUAGE plpgsql;

-- Apply audit triggers to critical tables
CREATE TRIGGER audit_member AFTER INSERT OR UPDATE OR DELETE ON member
  FOR EACH ROW EXECUTE FUNCTION audit_trigger_function();

CREATE TRIGGER audit_loan AFTER INSERT OR UPDATE OR DELETE ON loan
  FOR EACH ROW EXECUTE FUNCTION audit_trigger_function();

CREATE TRIGGER audit_fine AFTER INSERT OR UPDATE OR DELETE ON fine
  FOR EACH ROW EXECUTE FUNCTION audit_trigger_function();
```

9.3 Book Copy Status Update Function

sql

```
CREATE OR REPLACE FUNCTION update_book_copy_status()
RETURNS TRIGGER AS $
BEGIN
    IF TG_OP = 'INSERT' AND NEW.status = 'ACTIVE' THEN
        -- Mark book copy as checked out
        UPDATE book_copy
        SET status = 'CHECKED_OUT', updated_at = CURRENT_TIMESTAMP
        WHERE id = NEW.book_copy_id;
    ELSIF TG_OP = 'UPDATE' AND NEW.status = 'RETURNED' AND OLD.status = 'ACTIVE' THEN
        -- Mark book copy as available
        UPDATE book_copy
        SET status = 'AVAILABLE', updated_at = CURRENT_TIMESTAMP
        WHERE id = NEW.book_copy_id;
    END IF;
    RETURN COALESCE(NEW, OLD);
END;
$ LANGUAGE plpgsql;

CREATE TRIGGER loan_status_update AFTER INSERT OR UPDATE ON loan
    FOR EACH ROW EXECUTE FUNCTION update_book_copy_status();
```

10. Constraints and Business Rules

10.1 Check Constraints

```
sql
```

```
-- Ensure loan due date is after loan date
ALTER TABLE loan ADD CONSTRAINT chk_loan_dates
CHECK (due_date > loan_date);

-- Ensure return date is after loan date when provided
ALTER TABLE loan ADD CONSTRAINT chk_return_date
CHECK (return_date IS NULL OR return_date >= loan_date);

-- Ensure membership end date is after start date
ALTER TABLE member ADD CONSTRAINT chk_membership_dates
CHECK (membership_end_date > membership_start_date);

-- Ensure fine amount is positive
ALTER TABLE fine ADD CONSTRAINT chk_fine_amount
CHECK (amount >= 0);

-- Ensure payment amount is positive
ALTER TABLE payment ADD CONSTRAINT chk_payment_amount
CHECK (amount > 0);
```

10.2 Unique Constraints

```sql
-- Ensure no duplicate active loans for same book copy
CREATE UNIQUE INDEX idx_unique_active_loan
ON loan(book_copy_id)
WHERE status = 'ACTIVE';

-- Ensure member ID is unique and follows format
ALTER TABLE member ADD CONSTRAINT chk_member_id_format
CHECK (member_id ~ '^[A-Z]{2}[0-9]{6}');
```

---

11. Indexes for Performance

11.1 Composite Indexes

```sql
```

```sql
-- Frequently queried combinations
CREATE INDEX idx_member_status_membership ON member(status, membership_end_date);
CREATE INDEX idx_loan_member_status ON loan(member_id, status);
CREATE INDEX idx_fine_member_status ON fine(member_id, status);
CREATE INDEX idx_book_copy_book_status ON book_copy(book_id, status);
CREATE INDEX idx_reservation_book_status ON reservation(book_id, status);

-- Search optimization
CREATE INDEX idx_book_search ON book USING gin(
    to_tsvector('english', title || ' ' || author || ' ' || COALESCE(keywords, ''))
);
```

11.2 Partial Indexes

```sql
-- Index only active records
CREATE INDEX idx_active_members ON member(id) WHERE status = 'ACTIVE';
CREATE INDEX idx_active_loans ON loan(id) WHERE status = 'ACTIVE';
CREATE INDEX idx_unpaid_fines ON fine(id) WHERE status = 'UNPAID';
CREATE INDEX idx_available_copies ON book_copy(id) WHERE status = 'AVAILABLE';
```

---

12. Data Integrity and Validation

12.1 Foreign Key Constraints

All foreign key relationships are defined with appropriate CASCADE or RESTRICT actions:

- Member deletions are restricted if active loans exist
- Book deletions cascade to book copies
- Loan deletions are restricted if fines exist
- Staff user deletions are restricted if referenced in audit logs

12.2 Business Logic Validation

```sql
```

```
-- Function to validate loan eligibility
CREATE OR REPLACE FUNCTION can_member_borrow(p_member_id INTEGER)
RETURNS BOOLEAN AS $
DECLARE
    member_status VARCHAR(20);
    membership_expired BOOLEAN;
    outstanding_fines DECIMAL(10,2);
BEGIN
    SELECT
        m.status,
        m.membership_end_date < CURRENT_DATE,
        COALESCE(SUM(f.amount), 0)
    INTO member_status, membership_expired, outstanding_fines
    FROM member m
    LEFT JOIN fine f ON m.id = f.member_id AND f.status = 'UNPAID'
    WHERE m.id = p_member_id
    GROUP BY m.id, m.status, m.membership_end_date;

    RETURN (member_status = 'ACTIVE'
            AND NOT membership_expired
            AND outstanding_fines <= 10.00);
END;
$ LANGUAGE plpgsql;
```

13. Performance Considerations

13.1 Partitioning Strategy

```sql
-- Partition audit_log table by month for better performance
CREATE TABLE audit_log_y2025m01 PARTITION OF audit_log
FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');

CREATE TABLE audit_log_y2025m02 PARTITION OF audit_log
FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');
```

13.2 Materialized Views for Reporting

```sql
-- Pre-calculated statistics for dashboard
CREATE MATERIALIZED VIEW mv_daily_stats AS
SELECT
    DATE(created_at) AS stat_date,
    COUNT(*) FILTER (WHERE status = 'ACTIVE') AS loans_issued,
    COUNT(*) FILTER (WHERE status = 'RETURNED') AS books_returned,
    SUM(amount) FILTER (WHERE fine.status = 'PAID') AS fines_collected
FROM loan
LEFT JOIN fine ON loan.id = fine.loan_id
GROUP BY DATE(created_at);

-- Refresh daily
CREATE INDEX idx_mv_daily_stats_date ON mv_daily_stats(stat_date);
```

---

14. Security Considerations

14.1 Row-Level Security

```sql
-- Enable RLS on sensitive tables
ALTER TABLE member ENABLE ROW LEVEL SECURITY;
ALTER TABLE fine ENABLE ROW LEVEL SECURITY;
ALTER TABLE payment ENABLE ROW LEVEL SECURITY;

-- Policies for data access control
CREATE POLICY member_access_policy ON member
FOR ALL TO library_staff
USING (TRUE);

CREATE POLICY member_self_access ON member
FOR SELECT TO library_members
USING (member_id = current_setting('app.current_member_id'));
```

14.2 Data Encryption

```sql
-- Encrypt sensitive fields
ALTER TABLE member ALTER COLUMN email TYPE TEXT;
ALTER TABLE member ALTER COLUMN phone TYPE TEXT;

-- Use pgcrypto for sensitive data
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

---

15. Backup and Recovery Strategy

15.1 Backup Schedule

- **Full Backup**: Daily at 2 AM
- **Incremental Backup**: Every 4 hours
- **Transaction Log Backup**: Every 15 minutes
- **Retention**: 30 days full, 7 days incremental

15.2 Recovery Procedures

- Point-in-time recovery capability
- Automated backup verification
- Disaster recovery site synchronization
- Regular recovery testing procedures

This database schema provides a robust foundation for the Library Book Loan System with comprehensive data integrity, performance optimization, and security features.