

Ledger-Centered Architecture for Fee Management

Overview

The **Ledger-Centered Architecture** is a comprehensive financial tracking system where the **Student Ledger** serves as the single source of truth for all fee-related transactions. This architecture ensures financial integrity, complete auditability, and flexibility for Indian school fee management.

Core Principles

- 1. **Single Source of Truth** - Student ledger always reflects accurate financial position
- 2. **Immutable Audit Trail** - Every transaction is permanently recorded
- 3. **Payment Traceability** - Complete visibility into which payment covered which fee
- 4. **Flexible Allocation** - Payments can be allocated to multiple fees or left unallocated
- 5. **Financial Reporting** - Easy generation of reports from ledger transactions
- 6. **Manual Adjustments** - Support for waivers, refunds, and corrections

Architecture Components

1. Student Ledger (Central Hub)

Table: `student_ledgers`

The master record for each student's financial position.

Key Fields:

- `student_id` - Reference to student
- `academic_year_id` - Scoped to academic year
- `total_billed` - Sum of all fees assigned
- `total_paid` - Sum of all payments received
- `total_outstanding` - Remaining balance (billed - paid - adjusted)
- `total_adjusted` - Sum of waivers, discounts, refunds
- `last_payment_date` - Most recent payment timestamp
- `is_defaulter` - Boolean flag for overdue fees (>90 days)

Purpose:

- Acts as the financial dashboard for each student
- Provides instant access to current balance status
- Supports quick queries for defaulter lists and outstanding reports

2. Ledger Transactions (Audit Trail)

Table: `ledger_transactions`

Immutable record of every financial event.

Transaction Types:

- `fee_charge` - Fee assigned to student
- `payment` - Payment received from student/parent
- `adjustment` - Manual correction or adjustment
- `refund` - Money returned to student/parent
- `waiver` - Fee waived/written off
- `manual_debit` - Manual charge added
- `manual_credit` - Manual credit applied

Key Fields:

- `ledger_id` - Reference to student ledger
- `transaction_type` - Type of transaction (enum)
- `amount` - Transaction amount (positive/negative)
- `transaction_date` - When the transaction occurred
- `description` - Human-readable description
- `reference_type` - What caused this transaction (fee_session, payment, adhoc_fee)
- `reference_id` - ID of the reference entity
- `created_by` - User who created the transaction
- `is_reversed` - Flag if transaction was reversed
- `reversal_transaction_id` - Link to reversal transaction

Purpose:

- Complete audit trail for compliance and accounting
- Enables transaction history reports
- Supports reconciliation and dispute resolution
- Allows transaction reversal with full traceability

3. Payment Allocation (Junction Table)

Table: `payment_allocations`

Explicit tracking of which payment paid which fee.

Key Fields:

- `payment_id` - Reference to payment
- `student_id` - Reference to student
- `fee_type` - Type of fee ('fee_session' or 'adhoc_fee')
- `fee_session_id` - Reference to fee session (nullable)
- `adhoc_fee_id` - Reference to adhoc fee (nullable)
- `allocated_amount` - Amount allocated to this fee
- `fee_description` - Description of the fee
- `allocated_by` - User who made the allocation
- `allocation_notes` - Optional notes

Purpose:

- Links payments to specific fees (polymorphic association)
- Enables payment history per fee

- Supports partial payments across multiple fees
- Allows unallocated payments (advance payments)

4. Dual Fee Sources

Fee Sessions (Structured/Recurring Fees)

Table: `fee_sessions`

Bulk assignment of recurring fees to groups of students.

Examples:

- Annual tuition fees
- Quarterly exam fees
- Monthly transport fees
- Hostel fees

Workflow:

1. Admin creates fee session with fee structure
2. Session assigns fees to multiple students in bulk
3. Each assignment creates ledger transaction (`fee_charge`)
4. Student ledger `total_billed` increases

Adhoc Fees (One-time Fees)

Table: `adhoc_fee_assignments`

Individual one-time fees assigned to specific students.

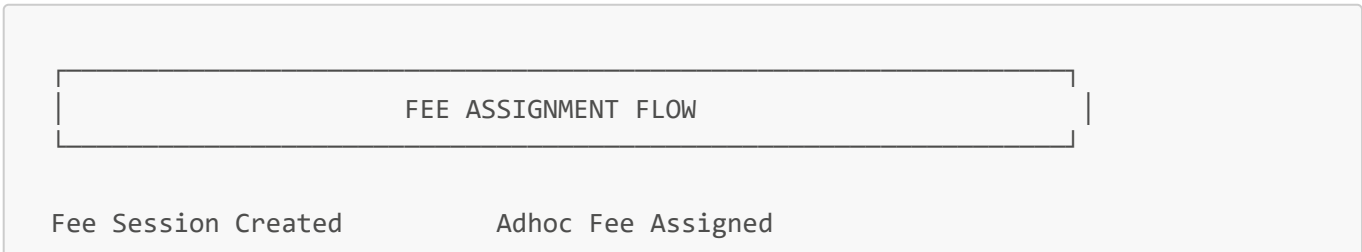
Examples:

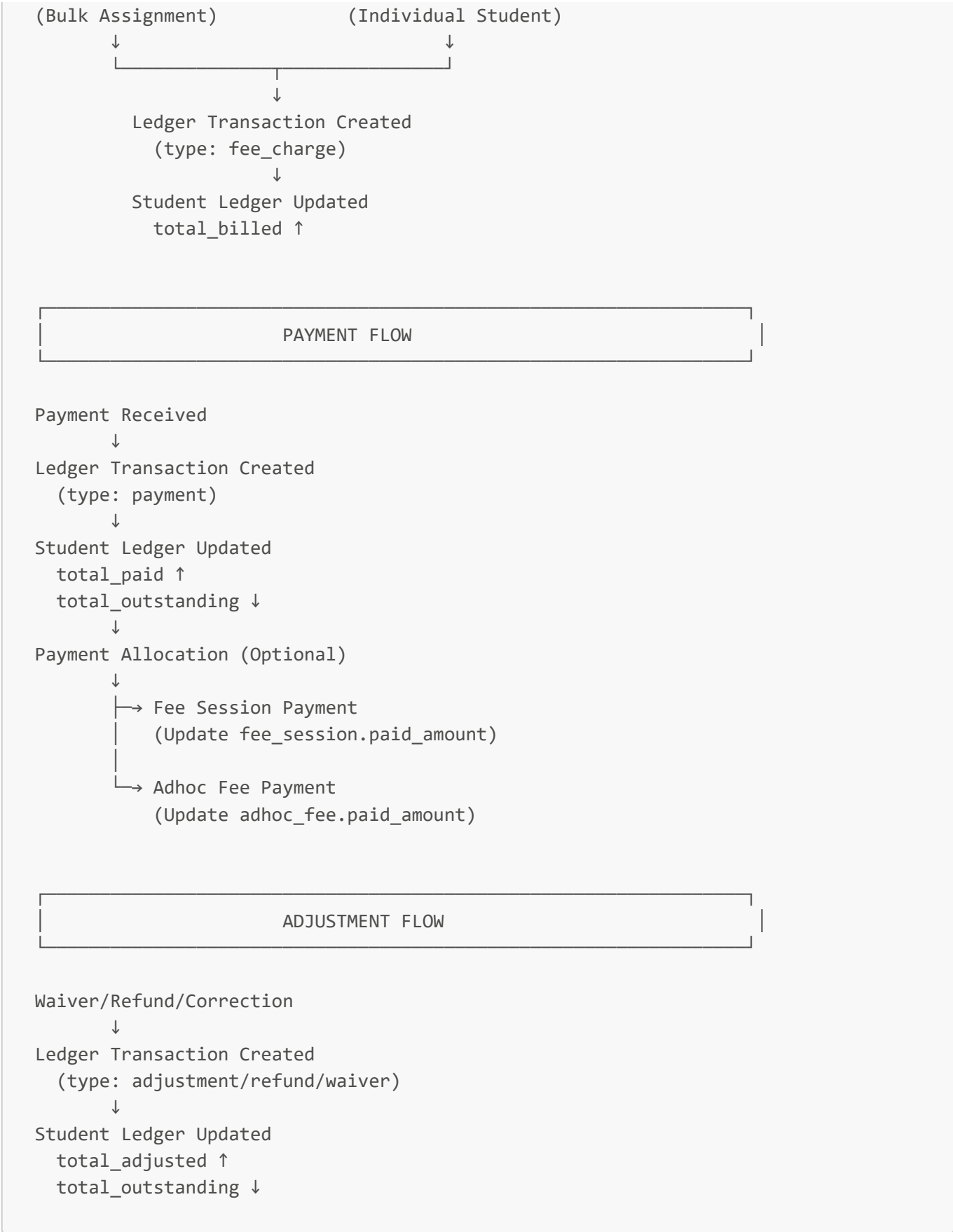
- Late fee fines
- Lost library book charges
- Special exam fees
- Event participation fees

Workflow:

1. Admin assigns adhoc fee to individual student
2. Assignment creates ledger transaction (`fee_charge`)
3. Student ledger `total_billed` increases

Transaction Flow Architecture





Database Schema

```
-- Master ledger for each student
CREATE TABLE student_ledgers (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```

    student_id INTEGER NOT NULL,
    academic_year_id INTEGER NOT NULL,
    total_billed DECIMAL(10, 2) DEFAULT 0,
    total_paid DECIMAL(10, 2) DEFAULT 0,
    total_outstanding DECIMAL(10, 2) DEFAULT 0,
    total_adjusted DECIMAL(10, 2) DEFAULT 0,
    last_payment_date DATETIME,
    is_defaulter BOOLEAN DEFAULT FALSE,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY(student_id) REFERENCES students(id),
    FOREIGN KEY(academic_year_id) REFERENCES academic_years(id),
    UNIQUE(student_id, academic_year_id)
);

-- Immutable transaction log
CREATE TABLE ledger_transactions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    ledger_id INTEGER NOT NULL,
    transaction_type VARCHAR(20) NOT NULL,
    amount DECIMAL(10, 2) NOT NULL,
    transaction_date DATETIME NOT NULL,
    description TEXT,
    reference_type VARCHAR(50),
    reference_id INTEGER,
    created_by INTEGER,
    is_reversed BOOLEAN DEFAULT FALSE,
    reversal_transaction_id INTEGER,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY(ledger_id) REFERENCES student_ledgers(id),
    FOREIGN KEY(created_by) REFERENCES users(id),
    FOREIGN KEY(reversal_transaction_id) REFERENCES ledger_transactions(id)
);

-- Payment to fee allocation tracking
CREATE TABLE payment_allocations (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    payment_id INTEGER NOT NULL,
    student_id INTEGER NOT NULL,
    fee_type VARCHAR(20) NOT NULL,
    fee_session_id INTEGER,
    adhoc_fee_id INTEGER,
    allocated_amount DECIMAL(10, 2) NOT NULL,
    fee_description VARCHAR(200),
    allocation_notes TEXT,
    allocated_by INTEGER,
    allocated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY(payment_id) REFERENCES payments(id) ON DELETE CASCADE,
    FOREIGN KEY(student_id) REFERENCES students(id) ON DELETE CASCADE,
    FOREIGN KEY(fee_session_id) REFERENCES fee_sessions(id),
    FOREIGN KEY(adhoc_fee_id) REFERENCES adhoc_fee_assignments(id),

```

```
FOREIGN KEY(allocated_by) REFERENCES users(id)
);
```

API Flow Examples

Example 1: Assign Fee Session

```
# 1. Create fee session
session = FeeSession(
    session_name="Q1 2025 Tuition Fees",
    fee_structure_id=1,
    total_amount=50000
)

# 2. Assign to students
for student_id in [101, 102, 103]:
    assignment = FeeSessionAssignment(
        fee_session_id=session.id,
        student_id=student_id,
        amount=50000,
        payment_status="pending"
    )

# 3. Create ledger transaction
create_ledger_transaction(
    student_id=student_id,
    transaction_type="fee_charge",
    amount=50000,
    reference_type="fee_session",
    reference_id=assignment.id
)

# 4. Update student ledger
update_ledger_summary(student_id, academic_year_id)
```

Example 2: Record Payment with Allocation

```
# 1. Create payment record
payment = Payment(
    student_id=101,
    amount=30000,
    payment_method="upi",
    payment_status="completed"
)

# 2. Create ledger transaction
create_ledger_transaction(
    student_id=101,
    transaction_type="payment",
```

```

        amount=30000,
        reference_type="payment",
        reference_id=payment.id
    )

# 3. Allocate payment to fees
allocations = [
    {"fee_session_id": 5, "amount": 20000},
    {"adhoc_fee_id": 12, "amount": 10000}
]

for allocation in allocations:
    PaymentAllocation.create(
        payment_id=payment.id,
        student_id=101,
        **allocation
    )

    # Update fee paid amount
    if allocation.get("fee_session_id"):
        update_fee_session_payment(allocation["fee_session_id"],
        allocation["amount"])
    elif allocation.get("adhoc_fee_id"):
        update_adhoc_fee_payment(allocation["adhoc_fee_id"], allocation["amount"])

# 4. Update student ledger
update_ledger_summary(101, academic_year_id)

```

Example 3: Issue Waiver

```

# 1. Create waiver transaction
create_ledger_transaction(
    student_id=101,
    transaction_type="waiver",
    amount=5000,
    description="Financial hardship waiver",
    created_by=admin_user_id
)

# 2. Update student ledger
update_ledger_summary(101, academic_year_id)
# Result: total_adjusted += 5000, total_outstanding -= 5000

```

Key Benefits

1. Financial Integrity

- Double-entry style accounting ensures balances are always accurate
- All changes tracked with full audit trail
- Supports reconciliation and error detection

2. Complete Auditability

- Every rupee accounted for with timestamp and user tracking
- Transaction reversal maintains complete history
- Compliance-ready for audits and inspections

3. Payment Traceability

- Know exactly which payment covered which fee
- Support for partial payments and overpayments
- Easy to generate payment allocation reports

4. Flexibility

- Handle both structured (sessions) and ad-hoc fees
- Support for manual adjustments and corrections
- Advance payments and credit balances supported

5. Reporting & Analytics

- Real-time outstanding fee reports
- Defaulter identification (>90 days overdue)
- Collection trends and payment history
- Revenue recognition and cash flow reports

6. Indian School Context

- Supports ₹ currency and GST compliance
- Multiple fee types (tuition, transport, hostel, exams)
- Quarterly/annual payment cycles
- Late fee calculations and waivers

Performance Considerations

Indexing Strategy

```
CREATE INDEX idx_ledger_student ON student_ledgers(student_id);
CREATE INDEX idx_ledger_academic_year ON student_ledgers(academic_year_id);
CREATE INDEX idx_transactions_ledger ON ledger_transactions(ledger_id);
CREATE INDEX idx_transactions_date ON ledger_transactions(transaction_date);
CREATE INDEX idx_allocations_payment ON payment_allocations(payment_id);
CREATE INDEX idx_allocations_student ON payment_allocations(student_id);
```

Caching

- Cache student ledger summaries for frequently accessed students
- Invalidate cache on any transaction that updates the ledger
- Cache defaulter lists with TTL

Batch Operations

- Process fee session assignments in bulk
- Update ledger summaries in batches
- Background jobs for large-scale operations

Future Enhancements

1. **Multi-Currency Support** - For international schools
2. **Payment Plans** - Installment tracking and reminders
3. **Auto-Reconciliation** - Bank statement matching
4. **SMS/Email Alerts** - Payment confirmations and reminders
5. **Parent Portal** - View ledger and make payments
6. **Advanced Analytics** - Predictive models for collection
7. **Integration** - Accounting software (Tally, QuickBooks)
8. **Blockchain** - Immutable receipt generation

Conclusion

The Ledger-Centered Architecture provides a robust, auditable, and flexible foundation for managing school fees in the Indian education context. By maintaining a single source of truth with complete transaction history, schools can ensure financial integrity while providing transparency to parents and stakeholders.

Last Updated: October 23, 2025 Version: 1.0