want to make a entry form for purchase, so need you to help me to generate database tables
what i want ?
- supplier account payment
- supplier account due

purchase invoice with total amount, discount and more common financial
let me know the initial tables

Below are the **initial database tables** you need for a purchase-entry system that includes:

- Supplier accounts
- Payments & dues
- Purchase invoices with totals, discount, taxes, etc.

These tables form a clean, scalable relational structure.

✅ Core Tables

1. suppliers

Stores supplier profile & opening balances.

Field	Type	Notes
supplier_id (PK)	INT	Auto increment
name	VARCHAR	Supplier name
contact_person	VARCHAR	Optional
phone	VARCHAR	Optional
email	VARCHAR	Optional
address	TEXT	Optional
opening_balance	DECIMAL(12,2)	Opening due (if any)

Field	Type	Notes
created_at	DATETIME	
updated_at	DATETIME	

2. purchase_invoices

Store purchase bill header / summary.

Field	Type	Notes
invoice_id (PK)	INT	Auto increment
supplier_id (FK)	INT	Link to suppliers
invoice_no	VARCHAR	Supplier bill number
invoice_date	DATE	Purchase date
subtotal	DECIMAL(12,2)	Before discount & tax
discount_amount	DECIMAL(12,2)	Flat or calculated
tax_amount	DECIMAL(12,2)	GST/VAT or custom
freight_amount	DECIMAL(12,2)	Optional
other_charges	DECIMAL(12,2)	Optional
total_amount	DECIMAL(12,2)	Final invoice total
paid_amount	DECIMAL(12,2)	Collected payment at entry
due_amount	DECIMAL(12,2)	Auto = total - paid
status	ENUM('Paid','Partial','Unpaid')	
remarks	TEXT	Optional
created_at	DATETIME	

Field	Type	Notes
updated_at	DATETIME	

---

3. purchase_invoice_items

Line items for the purchase invoice.

Field	Type	Notes
item_id (PK)	INT	Auto increment
invoice_id (FK)	INT	Link to purchase_invoices
product_id (FK)	INT	Reference products table
qty	DECIMAL(12,2)	
unit_price	DECIMAL(12,2)	
discount	DECIMAL(12,2)	Per item
tax	DECIMAL(12,2)	Per item
total	DECIMAL(12,2)	qty × unit_price - discount + tax

---

4. supplier_payments

Stores payment history for suppliers.

Field	Type	Notes
payment_id (PK)	INT	Auto increment
supplier_id (FK)	INT	
invoice_id (FK)	INT	Optional (NULL if advance payment)

Field	Type	Notes
payment_date	DATE	
amount	DECIMAL(12,2)	
payment_method	ENUM('Cash','Bank','UPI','Card','Cheque')	
reference_no	VARCHAR	Bank ref / cheque no
note	TEXT	

---

5. supplier_ledger (recommended)

A running ledger for all transactions (invoice, payment, credit note, debit note).

Field	Type	Notes
ledger_id (PK)	INT	Auto
supplier_id (FK)	INT	
transaction_type	ENUM('Invoice','Payment','Credit','Debit','Opening')	
transaction_id	INT	Related record ID
date	DATE	
debit	DECIMAL(12,2)	Invoice, debit note
credit	DECIMAL(12,2)	Payment, credit note
balance	DECIMAL(12,2)	Running balance

Why ledger?

✓ Easy to calculate supplier due

✓ Clean accounting system

✔️ More scalable if later you add returns, adjustments

---

6. products (if you want inventory integration)

Field	Type	Notes
product_id (PK)	INT	
name	VARCHAR	
unit	VARCHAR	pcs/kg/etc
purchase_price	DECIMAL(12,2)	
tax_rate	DECIMAL(12,2)	
stock_qty	DECIMAL(12,2)	optional

🧩 Relationship Diagram (Simplified)

yaml

Copy code

```yaml
suppliers
   |
   | 1 - M
   |
purchase_invoices ------ M - 1 ------ supplier_payments
       |
       | 1 - M
       |
purchase_invoice_items
```

And supplier_ledger links to all.

✔️ Your Basic System Will Support:

- Full supplier master
- Purchase invoice entry with discount, tax, freight
- Supplier payments (with or without invoice reference)
- Supplier account due calculation
- Ledger-style accounting

If you want, I can also provide:

✅ SQL scripts (MySQL / PostgreSQL / SQL Server)
✅ ERD diagram (image)
✅ Laravel / Node.js schema
✅ API endpoints design
Just tell me!

total amount of items => order_amount
total discount of items => discount_amount
total amount after discount => total_amount = order_amount - discount_amount
payment amount => paid_amount
due amount => due_amount = total_amount - paid_amount

want to handle more handy

if paid_amount is more than total_amount then rest of amount shoud be declare is it save as credit or returned
if less paid then make it due

if due was previous invoice, and new invoice paid is current + due

some times just pay for advance without any invoice
some times just pay for due without any invoice

and many of real world complexity, need to handle

Below is the **correct database design** that **fully supports all your real-world supplier accounting logic**, including:

- Order amount, discounts, totals
- Over-payment → credit
- Under-payment → due
- Paying old dues + new invoices together
- Advance payments without invoice
- Paying due without invoice
- Clean financial accounting
- Simple to query due balance anytime

This is how **professional ERPs** handle it.

✅ Core Principle

Instead of storing due/credit inside the invoice, you must use a **Supplier Ledger**.

Everything (invoice, payment, credit, debit, advance) goes to one ledger table, and the balance is calculated by summing debits and credits.

✔ This handles **every complexity**
✔ No duplication
✔ No surprises later
✔ Straight accounting-style system

✅ Final Tables (Optimized for All Cases)

1. suppliers

Same as before.

---

2. purchase_invoices

Keep only invoice values — NOT due, NOT credit.

Field	Type
invoice_id	INT (PK)
supplier_id	INT
order_amount	DECIMAL
discount_amount	DECIMAL
total_amount	DECIMAL
invoice_date	DATE
note	TEXT
created_at	DATETIME

👉 **No need to store paid or due here** — ledger handles it.

---

3. purchase_invoice_items

Same as previous.

---

4. supplier_payments

Record each payment event.

Field	Type
payment_id	INT (PK)
supplier_id	INT
invoice_id	INT NULL
payment_date	DATE

Field	Type
amount	DECIMAL
payment_method	ENUM
reference_no	VARCHAR
note	TEXT

---

5. supplier_ledger (MOST IMPORTANT)

This is where all accounting logic becomes simple.

Field	Type	Purpose
ledger_id	INT (PK)	
supplier_id	INT	
transaction_type	ENUM('Invoice','Payment','Credit','Debit','Advance','Adjustment')	
transaction_id	INT NULL	link to invoice or payment
date	DATE	
debit	DECIMAL	Money supplier OWES (invoice)
credit	DECIMAL	Money YOU paid to supplier
balance	DECIMAL	(running balance is optional)
note	TEXT	

How to enter ledger values:

Invoice:

- debit = invoice total_amount
- credit = 0

Payment (normal):

- debit = 0
- credit = amount

Advance payment:

- debit = 0
- credit = amount
- invoice_id = NULL

Paying previous due + new invoice:

- Just add a Payment entry
- System matches automatically because ledger reduces overall balance

Over-payment (credit balance):
If:

nginx Copy code

```nginx
paid_amount > total_amount
```

Then ledger naturally goes negative, meaning **you owe the supplier money**, i.e. supplier credit.

You don't need a special table — ledger handles it.

Under-payment:
Ledger stays positive → supplier due balance.

🧮 HOW BALANCE IS ALWAYS CALCULATED

Supplier balance formula:

ini Copy code

```ini
supplier_due = SUM(debit) - SUM(credit)
```

- **Positive balance** → supplier due
- **Negative balance** → supplier credit
- **Zero** → clear

That's it.

No more complex logic.

✔️ REAL-WORLD SCENARIOS AND HOW THE LEDGER HANDLES THEM

1️⃣ Invoice total = 1000, paid = 700

Ledger:

- Invoice → debit 1000
- Payment → credit 700

Balance = 300 due

2️⃣ Paid = 1200, invoice total = 1000

Ledger:

- Invoice → debit 1000
- Payment → credit 1200

Balance = -200 (credit available)

3️⃣ Supplier had previous due 500, now new invoice 800, pays = 1000

Ledger:

- Previous due = debit 500
- New invoice = debit 800
- Payment = credit 1000

Balance = 300 due

4️⃣ Supplier gives advance 2000 before any invoice

Ledger:

- Payment → credit 2000 (advance)

Balance = -2000

This advance gets adjusted automatically when invoices arrive.

5️⃣ Paying only old due without current invoice

Just insert a Payment entry:

- credit amount

  This reduces due directly.

🧩 DO WE NEED ANY EXTRA TABLES?

Only **one more optional table**:

6. payment_allocations (optional)

Use if you want to link a single payment to multiple invoices explicitly.

Field	**Type**
id	INT
payment_id	INT
invoice_id	INT
allocated_amount	DECIMAL

ERP-style systems use this when exact mapping is necessary.

🎉 You now have a COMPLETE PURCHASE + SUPPLIER ACCOUNT SYSTEM

It supports:

✔️ Invoices (with total, discount, etc.)

✔️ Partial payments

✔️ Overpayments → credit

✔ Underpayments → due
✔ Advance payments
✔ Paying due without invoice
✔ Paying invoice + due together
✔ Auto-balance using simple math

If you want, I can also provide:

✅ MySQL CREATE TABLE scripts
✅ ERD diagram (image)
✅ Laravel migration files
✅ Example queries (calculate due, insert invoice, payment logic, etc.)

Just tell me!

your formula with supplier_ledger is a very good way, now want to mark purchase_invoices individually which are paid or not
supplier_payments is different records table is good way
in complexity-
inv 1 total - 1000, paid - 500
inv 2 total - 500, paid - 1500
inv 3 total - 300, paid - 2500
and so on

Great — you want **each purchase invoice to know its own payment status** (Paid / Partial / Unpaid), *while still using the ledger for total account balance*.
This can be done cleanly with a **payment allocation table**, which works alongside ledger.

This is how real accounting software handles it.
Below is the perfect solution.

✅ Final Structure (Industry-Grade)

✔️ `purchase_invoices`

Keeps each invoice’s individual balance:

Add fields:

- `paid_amount`
- `due_amount`
- `status` (Paid / Partial / Unpaid)

You **calculate these automatically** after each payment operation.

Field	Type
invoice_id	INT PK
supplier_id	INT
total_amount	DECIMAL
paid_amount	DECIMAL
due_amount	DECIMAL
status	ENUM('Paid','Partial','Unpaid')
invoice_date	DATE

✔️ `supplier_payments`

Every payment event (cash, bank, UPI, etc.).

Field	Type
payment_id	INT PK
supplier_id	INT
payment_date	DATE

Field	Type
amount	DECIMAL
payment_method	ENUM
note	TEXT

No linking to any invoice here.
Because one payment can affect *multiple* invoices.

⭐ THE KEY: `payment_allocations`

This table connects payments to invoices in ANY way you need.

Example:

- Payment of 500 goes to Invoice #1
- Remaining 150 goes to Invoice #3
- Payment of 1500 goes to Invoice #2
- Extra 300 becomes credit

This table supports all those messy cases.

Field	Type	Meaning
allocation_id	INT PK	
payment_id	INT	which payment
invoice_id	INT	which invoice
allocated_amount	DECIMAL	how much from this payment went to this invoice

---

✔️ How Your Complex Example Works

Your scenario:

Invoice	Total	Paid
Inv 1	1000	500
Inv 2	500	1500
Inv 3	300	2500

But payments can be allocated like:

Invoice 1

Total = 1000

Paid = 500

Due = 500

Status = Partial

Invoice 2

Total = 500

Paid = 1500

Means: 500 goes to this invoice, 1000 becomes **supplier credit**

So:

- invoice status = **Paid**
- ledger credit = 1000

Invoice 3

Total = 300

Paid = 2500

Means:

- Invoice paid = 300
- Extra = 2200 → supplier credit

Status = Paid

⭐ How `payment_allocations` stores this

For Invoice 1

payment_id	invoice_id	allocated_amount
101	1	500

For Invoice 2

payment_id	invoice_id	allocated_amount
102	2	500
102	NULL	1000

For Invoice 3

payment_id	invoice_id	allocated_amount
103	3	300
103	NULL	2200

⭐ How to mark invoice status automatically

After inserting payment allocations:

1. Calculate paid amount:

ini Copy code

```ini
paid_amount = SUM(allocated_amount WHERE invoice_id = ?)
```

2. Update invoice:

lua Copy code

```lua
due_amount = total_amount - paid_amount
```

```
IF paid_amount = 0 → status = 'Unpaid'
IF paid_amount > 0 AND paid_amount < total_amount → status = 'Partial'
IF paid_amount >= total_amount → status = 'Paid'
```

⭐ How supplier credit / advance is stored

Whenever a payment has leftover that does NOT map to any invoice:

Store in ledger:

- credit = leftover amount
- transaction_type = "Advance"

This becomes a **supplier credit** and will auto-adjust on the next invoice.

⭐ Why this structure works perfectly

✔ Each invoice knows its payment status

✔ You support:

- Overpayment
- Underpayment
- Advance payments
- Pay old due + new invoice
- Paying without invoice
- Credit balances

  ✔ No messy logic, clean separation

  ✔ Used in SAP, QuickBooks, Zoho Books, Tally (same structure)

If you want next:

- Full MySQL table definitions
- Auto allocation algorithm
- Example queries (get due, credit, invoice status)

- ER diagram

  Just tell me!

share a clean table format

Below is a **clean, production-ready table format** for your full purchase + supplier accounting system.

This design is **simple, scalable, ERP-grade**, and supports:

✔️ Invoice totals, discount, due
✔️ Partial payment
✔️ Overpayment → credit
✔️ Advance payment
✔️ Pay old dues
✔️ Pay without invoice
✔️ Auto invoice status
✔️ Full ledger accounting

✅ 1. suppliers

sql

Copy code

```sql
CREATE TABLE suppliers (
    supplier_id      INT AUTO_INCREMENT PRIMARY KEY,
    name             VARCHAR(255) NOT NULL,
    phone            VARCHAR(50),
    email            VARCHAR(255),
    address          TEXT,
    opening_balance  DECIMAL(12,2) DEFAULT 0,
    created_at       DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at       DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
);
```

✅ 2. purchase_invoices

Keep invoice-level details and track paid & due status.

sql

Copy code

```
CREATE TABLE purchase_invoices (
    invoice_id      INT AUTO_INCREMENT PRIMARY KEY,
    supplier_id     INT NOT NULL,

    order_amount    DECIMAL(12,2) NOT NULL,   -- total of item amounts
    discount_amount DECIMAL(12,2) DEFAULT 0,
    total_amount    DECIMAL(12,2) NOT NULL,   -- order_amount - discount_amount

    paid_amount     DECIMAL(12,2) DEFAULT 0,  -- auto updated
    due_amount      DECIMAL(12,2) DEFAULT 0,  -- total_amount - paid_amount

    status          ENUM('Unpaid', 'Partial', 'Paid') DEFAULT 'Unpaid',

    invoice_no      VARCHAR(100),
    invoice_date    DATE NOT NULL,
    note            TEXT,

    created_at      DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at      DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAM

    FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id)
);
```

✅ 3. purchase_invoice_items

sql

Copy code

```sql
CREATE TABLE purchase_invoice_items (
    item_id        INT AUTO_INCREMENT PRIMARY KEY,
    invoice_id     INT NOT NULL,
    product_name   VARCHAR(255) NOT NULL,
    qty            DECIMAL(12,2) NOT NULL,
    unit_price     DECIMAL(12,2) NOT NULL,
    discount       DECIMAL(12,2) DEFAULT 0,
    tax            DECIMAL(12,2) DEFAULT 0,
    total          DECIMAL(12,2) NOT NULL,

    FOREIGN KEY (invoice_id) REFERENCES purchase_invoices(invoice_id)
);
```

✅ 4. supplier_payments

One record per payment (cash, bank, digital, anything).

sql

Copy code

```sql
CREATE TABLE supplier_payments (
    payment_id      INT AUTO_INCREMENT PRIMARY KEY,
    supplier_id     INT NOT NULL,
    payment_date    DATE NOT NULL,
    amount          DECIMAL(12,2) NOT NULL,
    payment_method  ENUM('Cash','Bank','UPI','Card','Cheque','Other') NOT NULL,
    reference_no    VARCHAR(255),
    note            TEXT,

    created_at      DATETIME DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id)
);
```

⭐ 5. payment_allocations (super important)

Connects payments to invoices (many-to-many).

sql

Copy code

```sql
CREATE TABLE payment_allocations (
    allocation_id     INT AUTO_INCREMENT PRIMARY KEY,
    payment_id        INT NOT NULL,
    invoice_id        INT NULL,              -- NULL = advance or credit
    allocated_amount  DECIMAL(12,2) NOT NULL,

    FOREIGN KEY (payment_id) REFERENCES supplier_payments(payment_id),
    FOREIGN KEY (invoice_id) REFERENCES purchase_invoices(invoice_id)
);
```

⭐ 6. supplier_ledger (ERP style)

Everything flows into this ledger automatically:

sql

Copy code

```sql
CREATE TABLE supplier_ledger (
    ledger_id        INT AUTO_INCREMENT PRIMARY KEY,
    supplier_id      INT NOT NULL,
    transaction_type ENUM('Invoice','Payment','Credit','Debit','Advance','Adjustm
    transaction_id   INT NULL,

    date             DATE NOT NULL,
    debit            DECIMAL(12,2) DEFAULT 0,    -- invoice, debit note
    credit           DECIMAL(12,2) DEFAULT 0,   -- payments, credit note

    note             TEXT,

    created_at       DATETIME DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id)
);
```

🧩 RELATION SUMMARY

nginx Copy code

```nginx
suppliers
    ├── purchase_invoices
    │       └── purchase_invoice_items
    ├── supplier_payments
    │       └── payment_allocations
    └── supplier_ledger
```

🎉 This table structure supports:

✔️ Calculate invoice due

✔️ Calculate supplier total due

✔️ Invoice-level paid/unpaid/partial

✔️ Overpayment → store as supplier credit

✔️ Advance payments without invoice

✔️ Paying old dues automatically

✔️ Ledger-based accounting

✔️ No data duplication

✔️ Scalable & clean

If you want next:

📌 Sample SQL routines for:

- Auto payment allocation
- Auto updating invoice status
- Calculating supplier due balance
- Query: unpaid invoices list
- Ledger view per supplier

Just tell me!

Auto payment allocation
Auto updating invoice status
Calculating supplier due balance
Query: unpaid invoices list

Below are **clean and production-ready SQL procedures / queries** for:

1️⃣ Auto Payment Allocation
2️⃣ Auto Updating Invoice Status
3️⃣ Calculating Supplier Total Due Balance
4️⃣ Listing Unpaid Invoices

These work perfectly with the table structure we created earlier.

✅ 1. Auto Payment Allocation Procedure

This procedure:

✔️ Takes a supplier payment
✔️ Automatically allocates it to **oldest unpaid invoices first**
✔️ Stores allocations into `payment_allocations`
✔️ Updates invoice paid/due status
✔️ Inserts leftover as supplier credit

> **This is FIFO allocation (industry standard).**

🔧 MySQL Stored Procedure

PROC: allocate_payment(payment_id)

sql

Copy code

```
DELIMITER $$

CREATE PROCEDURE allocate_payment(IN p_payment_id INT)
BEGIN
    DECLARE v_supplier_id INT;
    DECLARE v_payment_amount DECIMAL(12,2);
    DECLARE v_leftover DECIMAL(12,2);
    DECLARE v_invoice_id INT;
    DECLARE v_invoice_due DECIMAL(12,2);

    -- Get supplier + amount
    SELECT supplier_id, amount
    INTO v_supplier_id, v_payment_amount
    FROM supplier_payments
    WHERE payment_id = p_payment_id;

    SET v_leftover = v_payment_amount;

    -- Cursor invoices: oldest first
    DECLARE invoice_cursor CURSOR FOR
        SELECT invoice_id, due_amount
        FROM purchase_invoices
        WHERE supplier_id = v_supplier_id
          AND due_amount > 0
        ORDER BY invoice_date ASC, invoice_id ASC;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_invoice_id = NULL;

    OPEN invoice_cursor;

    invoice_loop: LOOP
        FETCH invoice_cursor INTO v_invoice_id, v_invoice_due;

        IF v_invoice_id IS NULL THEN
            LEAVE invoice_loop;
        END IF;

        IF v_leftover <= 0 THEN
            LEAVE invoice_loop;
        END IF;
        -- allocate min(due, leftover)
        SET @alloc = LEAST(v_leftover, v_invoice_due);

        -- Insert allocation
        INSERT INTO payment_allocations (payment_id, invoice_id, allocated_amount
        VALUES (p_payment_id, v_invoice_id, @alloc);

        -- Update invoice record
```

keep a note, its for SQLite

Understood — **SQLite has no triggers for multi-table joins**, no enums, no stored procedures.
So I will give you **SQLite-compatible SQL**, with logic you can run in your backend code (PHP/Laravel/Node/etc.).

Below are **clean and safe algorithms** + **SQLite queries** for:

1️⃣ Auto payment allocation
2️⃣ Auto updating invoice status
3️⃣ Calculating total supplier due balance
4️⃣ Querying unpaid invoice list

All SQLite friendly.

✅ 1. AUTO PAYMENT ALLOCATION (SQLite-Friendly Algorithm)

> You will run this logic in your backend code (because SQLite does not support stored procedures).

🔥 Steps to allocate a new payment:

Input:

`payment_id` , `supplier_id` , `payment_amount`

Step A — Get all unpaid/partial invoices (oldest first)

```sql
SELECT invoice_id, total_amount, paid_amount, due_amount
FROM purchase_invoices
WHERE supplier_id = ?
AND due_amount > 0
ORDER BY invoice_date ASC, invoice_id ASC;
```

Step B — Loop in code:

For each invoice:

```yaml
if payment_amount <= 0: stop

If payment_amount >= due_amount:
    allocated = due_amount
    payment_amount -= due_amount
Else:
    allocated = payment_amount
    payment_amount = 0
```

Insert allocation:

```sql
INSERT INTO payment_allocations (payment_id, invoice_id, allocated_amount)
VALUES (?, ?, ?);
```

Update paid_amount & due_amount:

sql

```
UPDATE purchase_invoices
SET paid_amount = paid_amount + ?,
    due_amount  = total_amount - (paid_amount + ?)
WHERE invoice_id = ?;
```

When payment left over (`payment_amount > 0`):

- Insert into `payment_allocations` with `invoice_id = NULL` (credit)
- Add ledger entry for credit

Example:

sql

Copy code

```
INSERT INTO payment_allocations (payment_id, invoice_id, allocated_amount)
VALUES (?, NULL, ?);       -- leftover credit
```

✅ 2. AUTO UPDATE INVOICE STATUS (SQLite)

After updating paid_amount/due_amount, update status:

sql

Copy code

```
UPDATE purchase_invoices
SET status =
    CASE
        WHEN paid_amount = 0 THEN 'Unpaid'
        WHEN paid_amount >= total_amount THEN 'Paid'
        ELSE 'Partial'
    END
WHERE invoice_id = ?;
```

✅ 3. CALCULATE TOTAL SUPPLIER DUE BALANCE (SQLite)

Using ledger approach:

Supplier Due Formula:

ini

```
due = SUM(debit) - SUM(credit)
```

SQLite query:

```sql
SELECT
    IFNULL(SUM(debit), 0) - IFNULL(SUM(credit), 0) AS supplier_due
FROM supplier_ledger
WHERE supplier_id = ?;
```

Result:

- `> 0` → Supplier due amount
- `< 0` → You have credit with supplier
- `0` → Clean account

✅ 4. QUERY: UNPAID INVOICES LIST

Invoices with due_amount > 0

```sql
SELECT invoice_id, invoice_date, total_amount, paid_amount, due_amount
FROM purchase_invoices
WHERE supplier_id = ?
AND due_amount > 0
ORDER BY invoice_date ASC;
```

If you want **all suppliers' unpaid invoices**:

```sql
SELECT supplier_id, invoice_id, invoice_date, total_amount, paid_amount, due_amou
FROM purchase_invoices
WHERE due_amount > 0
ORDER BY supplier_id ASC, invoice_date ASC;
```

⭐ FULL WORKFLOW (SQLite) — REAL WORLD

Here is the correct sequence when a new payment is received:

◆ Step 1: Insert into supplier_payments

sql Copy code

```sql
INSERT INTO supplier_payments (supplier_id, payment_date, amount, payment_method,
VALUES (?, ?, ?, ?, ?);
```

Retrieve `payment_id`.

◆ Step 2: Auto allocate payment (backend loop)

Use the algorithm shown above.

◆ Step 3: Insert ledger entry (for entire payment)

sql Copy code

```sql
INSERT INTO supplier_ledger (supplier_id, transaction_type, transaction_id, date,
VALUES (?, 'Payment', ?, ?, 0, ?, ?);
```

Leftover credit also becomes a ledger entry:

sql Copy code

```sql
INSERT INTO supplier_ledger (supplier_id, transaction_type, date, debit, credit,
VALUES (?, 'Advance', ?, 0, ?, 'Remaining payment credit');
```

◆ Step 4: Update invoice status

(Shown above)

◆ Step 5: Supplier balance check

sql

Copy code

```sql
SELECT IFNULL(SUM(debit), 0) - IFNULL(SUM(credit), 0) AS supplier_due
FROM supplier_ledger
WHERE supplier_id = ?;
```