**AMIS Database Schema**

1. Product table

CREATE TABLE products (

product\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

description TEXT,

category\_id INT,

unit VARCHAR(50), -- e.g., kg, piece, liter

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

1. Inventory table

CREATE TABLE inventory (

inventory\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_id INT NOT NULL,

stock\_quantity DECIMAL(10,2) DEFAULT 0, -- Total stock available

average\_price DECIMAL(10,2) DEFAULT 0, -- Weighted average price

last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (product\_id) REFERENCES products(product\_id) ON DELETE CASCADE

);

1. Purchase table

CREATE TABLE purchases (

purchase\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_id INT NOT NULL,

supplier\_id INT,

quantity DECIMAL(10,2) NOT NULL,

unit\_price DECIMAL(10,2) NOT NULL, -- Cost per unit at purchase time

total\_cost DECIMAL(10,2) GENERATED ALWAYS AS (quantity \* unit\_price) STORED,

min\_stock\_threshold DECIMAL(10,2) DEFAULT 5,

purchase\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (product\_id) REFERENCES products(product\_id) ON DELETE CASCADE

);

UPDATE inventory

SET

average\_price = ((stock\_quantity \* average\_price) + (NEW.quantity \* NEW.unit\_price))

/ (stock\_quantity + NEW.quantity),

stock\_quantity = stock\_quantity + NEW.quantity

WHERE product\_id = NEW.product\_id;

1. Sales table

CREATE TABLE sales (

sale\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_id INT NOT NULL,

quantity\_sold DECIMAL(10,2) NOT NULL,

sale\_price DECIMAL(10,2) NOT NULL, -- Selling price per unit

total\_revenue DECIMAL(10,2) GENERATED ALWAYS AS (quantity\_sold \* sale\_price) STORED,

sale\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (product\_id) REFERENCES products(product\_id) ON DELETE CASCADE

);

UPDATE inventory

SET stock\_quantity = stock\_quantity - NEW.quantity\_sold

WHERE product\_id = NEW.product\_id;

1. Suppliers table

CREATE TABLE suppliers (

supplier\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

contact\_info TEXT

);

1. Purchase orders table

CREATE TABLE purchase\_orders (

po\_id INT PRIMARY KEY AUTO\_INCREMENT,

supplier\_id INT NOT NULL,

order\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status ENUM('Pending', 'Received', 'Cancelled') DEFAULT 'Pending',

total\_cost DECIMAL(10,2) DEFAULT 0, -- Calculated from purchase\_order\_items

received\_date TIMESTAMP NULL,

FOREIGN KEY (supplier\_id) REFERENCES suppliers(supplier\_id) ON DELETE CASCADE

);

1. Purchase order items table

CREATE TABLE purchase\_order\_items (

po\_item\_id INT PRIMARY KEY AUTO\_INCREMENT,

po\_id INT NOT NULL,

product\_id INT NOT NULL,

quantity DECIMAL(10,2) NOT NULL,

unit\_price DECIMAL(10,2) NOT NULL,

total\_price DECIMAL(10,2) GENERATED ALWAYS AS (quantity \* unit\_price) STORED,

FOREIGN KEY (po\_id) REFERENCES purchase\_orders(po\_id) ON DELETE CASCADE,

FOREIGN KEY (product\_id) REFERENCES products(product\_id) ON DELETE CASCADE

);

DELIMITER $$

CREATE TRIGGER update\_inventory\_on\_po\_receive

AFTER UPDATE ON purchase\_orders

FOR EACH ROW

BEGIN

-- Only update inventory if status is changed to 'Received'

IF NEW.status = 'Received' THEN

-- Loop through each product in the PO

INSERT INTO purchases (product\_id, supplier\_id, quantity, unit\_price, purchase\_date)

SELECT poi.product\_id, OLD.supplier\_id, poi.quantity, poi.unit\_price, NOW()

FROM purchase\_order\_items poi WHERE poi.po\_id = OLD.po\_id;

-- Update inventory stock and average price

UPDATE inventory i

JOIN (

SELECT poi.product\_id,

SUM(poi.quantity) AS total\_qty,

SUM(poi.quantity \* poi.unit\_price) / SUM(poi.quantity) AS new\_avg\_price

FROM purchase\_order\_items poi

WHERE poi.po\_id = OLD.po\_id

GROUP BY poi.product\_id

) temp ON i.product\_id = temp.product\_id

SET i.stock\_quantity = i.stock\_quantity + temp.total\_qty,

i.average\_price = ((i.stock\_quantity \* i.average\_price) + (temp.total\_qty \* temp.new\_avg\_price))

/ (i.stock\_quantity + temp.total\_qty);

END IF;

END$$

DELIMITER ;

*When the order is received, update the PO status:*

UPDATE purchase\_orders

SET status = 'Received', received\_date = NOW()

WHERE po\_id = 1;

1. Carts table

CREATE TABLE carts (

cart\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT NOT NULL, -- Buyer ID

status ENUM('Active', 'Checked Out', 'Abandoned') DEFAULT 'Active',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

1. Cart items table

CREATE TABLE cart\_items (

cart\_item\_id INT PRIMARY KEY AUTO\_INCREMENT,

cart\_id INT NOT NULL,

product\_id INT NOT NULL,

quantity DECIMAL(10,2) NOT NULL CHECK (quantity > 0),

price\_at\_add DECIMAL(10,2) NOT NULL, -- Price at the time of adding to cart

FOREIGN KEY (cart\_id) REFERENCES carts(cart\_id) ON DELETE CASCADE,

FOREIGN KEY (product\_id) REFERENCES products(product\_id) ON DELETE CASCADE

);

1. Orders table

CREATE TABLE orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT NOT NULL,

cart\_id INT NOT NULL,

status ENUM('Pending Payment', 'Processing', 'Shipped', 'Delivered', 'Cancelled') DEFAULT 'Pending Payment',

total\_amount DECIMAL(10,2) DEFAULT 0,

order\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (cart\_id) REFERENCES carts(cart\_id) ON DELETE CASCADE

);

1. Order items table

CREATE TABLE order\_items (

order\_item\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT NOT NULL,

product\_id INT NOT NULL,

quantity DECIMAL(10,2) NOT NULL,

price\_at\_sale DECIMAL(10,2) NOT NULL,

total\_price DECIMAL(10,2) GENERATED ALWAYS AS (quantity \* price\_at\_sale) STORED,

FOREIGN KEY (order\_id) REFERENCES orders(order\_id) ON DELETE CASCADE,

FOREIGN KEY (product\_id) REFERENCES products(product\_id) ON DELETE CASCADE

);

*Checkout Process (Convert Cart to Order)*

-- Step 1: Create an Order

INSERT INTO orders (user\_id, cart\_id, total\_amount)

SELECT c.user\_id, c.cart\_id, SUM(ci.quantity \* ci.price\_at\_add)

FROM carts c

JOIN cart\_items ci ON c.cart\_id = ci.cart\_id

WHERE c.cart\_id = 1

GROUP BY c.cart\_id;

-- Step 2: Get the Order ID

SET @order\_id = LAST\_INSERT\_ID();

-- Step 3: Move Cart Items to Order Items

INSERT INTO order\_items (order\_id, product\_id, quantity, price\_at\_sale)

SELECT @order\_id, ci.product\_id, ci.quantity, ci.price\_at\_add

FROM cart\_items ci

WHERE ci.cart\_id = 1;

-- Step 4: Deduct Inventory Stock

UPDATE inventory i

JOIN cart\_items ci ON i.product\_id = ci.product\_id

SET i.stock\_quantity = i.stock\_quantity - ci.quantity

WHERE ci.cart\_id = 1;

-- Step 5: Mark Cart as "Checked Out"

UPDATE carts SET status = 'Checked Out' WHERE cart\_id = 1;

-- Step 6: Remove Items from Cart

DELETE FROM cart\_items WHERE cart\_id = 1;

*Stock Alert + Auto-Reorder for Low Stock*

DELIMITER $$

CREATE TRIGGER auto\_reorder\_on\_low\_stock

AFTER UPDATE ON inventory

FOR EACH ROW

BEGIN

IF NEW.stock\_quantity < NEW.min\_stock\_threshold THEN

-- Create a new Purchase Order for Restocking

INSERT INTO purchase\_orders (supplier\_id, status)

VALUES ((SELECT supplier\_id FROM suppliers ORDER BY RAND() LIMIT 1), 'Pending');

-- Get the New PO ID

SET @po\_id = LAST\_INSERT\_ID();

-- Add the Low Stock Product to the PO

INSERT INTO purchase\_order\_items (po\_id, product\_id, quantity, unit\_price)

VALUES (@po\_id, NEW.product\_id, 50, NEW.average\_price);

END IF;

END$$

DELIMITER ;