StoreFlow — A1.2 Data Model & SQL Specification (Part 4)

24. Orders & Fulfilment Domain – Overview

The Orders & Fulfilment Domain is the core transactional heart of StoreFlow.

Responsibilities:

• Represent every customer order, regardless of fulfilment type.

• Capture the full state of the order lifecycle.

• Freeze all pricing and item data at time of purchase.

• Track fulfilment progress (pickup or shipping).

• Provide a reliable audit trail for merchants, customers, and accounting.

Design goals:

• Immutable financial data once order is completed.

• Clear, finite state machine for status transitions.

• Support both pickup and shipping in a unified model.

• Stripe-ready but not Stripe-dependent in v1 (payments simulated as successful).

25. Orders Table – Full Specification

TABLE: orders

---

id BIGINT PK

public_id VARCHAR(32) UNIQUE -- customer-facing order code

merchant_id BIGINT FK → merchants.id

store_id BIGINT FK → stores.id

customer_id BIGINT FK → customers.id

fulfilment_type ENUM('pickup','shipping') NOT NULL

status ENUM(

'pending', -- created but not yet accepted

'accepted', -- merchant acknowledged

'in_progress', -- being prepared (pickup)

'ready', -- ready for pickup

'packing', -- packing for shipping

'shipped', -- handed to carrier

'delivered', -- customer received

'cancelled' -- cancelled by merchant or system

) NOT NULL DEFAULT 'pending'

payment_status ENUM('unpaid','paid','refunded') DEFAULT 'paid' -- v1 treat as paid

payment_method VARCHAR(50) NULL -- future: card/wallet/etc

payment_reference VARCHAR(191) NULL -- future Stripe payment_intent_id, etc

MONETARY FIELDS (CENTS):

items_total_cents INT NOT NULL

shipping_cost_cents INT NOT NULL DEFAULT 0

discount_cents INT NOT NULL DEFAULT 0

tax_cents INT NOT NULL DEFAULT 0 -- flexible for different tax regimes

total_cents INT NOT NULL -- items_total + shipping - discounts + tax

CUSTOMER SNAPSHOT:

customer_name VARCHAR(255)

customer_email VARCHAR(255)

customer_mobile VARCHAR(50)

PICKUP FIELDS:

pickup_time DATETIME NULL -- optional requested pickup window

pickup_notes TEXT NULL

SHIPPING SNAPSHOT:

shipping_method VARCHAR(255) NULL

shipping_status VARCHAR(50) NULL -- 'pending','packing','shipped','delivered','returned'

tracking_code VARCHAR(255) NULL

tracking_url VARCHAR(1024) NULL

shipping_name VARCHAR(255) NULL

shipping_line1 VARCHAR(255) NULL

shipping_line2 VARCHAR(255) NULL

shipping_city VARCHAR(255) NULL

shipping_state VARCHAR(255) NULL

shipping_postcode VARCHAR(20) NULL

shipping_country VARCHAR(2) NULL -- ISO country code

TIMESTAMPS:

placed_at DATETIME -- creation time

accepted_at DATETIME NULL

ready_at DATETIME NULL -- ready for pickup OR shipped

completed_at DATETIME NULL

cancelled_at DATETIME NULL

created_at TIMESTAMP

updated_at TIMESTAMP

Indexes:

• idx_orders_merchant_store (merchant_id, store_id)

• idx_orders_customer (customer_id)

• idx_orders_status (store_id, status)

• idx_orders_public (public_id)

• idx_orders_placed (store_id, placed_at)

Notes:

• placed_at is separate from created_at for clearer business semantics.

• payment fields are designed to plug in Stripe later without schema changes.

26. Order Status State Machine

The order status flow is implemented as a constrained state machine.

Shared initial states:

pending → accepted

Pickup-only states:

accepted → in_progress → ready → completed

Shipping-only states:

accepted → packing → shipped → delivered

Cancellation:

Any non-terminal state except completed/delivered can transition to cancelled.

Invalid transitions:

• completed → any other state (blocked)

• delivered → any other state (blocked)

• cancelled → any other state (blocked)

Enforcement:

• Service-layer methods (OrderService) enforce allowed transitions.

• Status change attempts are validated against allowed graph.

• Audit logs capture: previous_status, new_status, actor, timestamp.

27. Order Items – Full Specification

TABLE: order_items

---

id BIGINT PK

order_id BIGINT FK → orders.id

product_id BIGINT FK → products.id -- original product reference (for analytics)

name VARCHAR(255) -- frozen product name

sku VARCHAR(64) NULL -- optional SKU, future use

quantity INT NOT NULL

unit_price_cents INT NOT NULL -- price per unit at time of order

line_subtotal_cents INT NOT NULL -- unit_price_cents * quantity

tax_cents INT NOT NULL DEFAULT 0

total_cents INT NOT NULL -- line_subtotal + tax + modifiers

TIMESTAMPS:

created_at, updated_at

Indexes:

• idx_order_items_order (order_id)

• idx_order_items_product (product_id)

Immutability:

• Once order reaches 'paid', these values must never change.

• Later corrections use credit notes or new orders (not in-place edits).

28. Order Item Options – Full Specification

TABLE: order_item_options

---

id BIGINT PK

order_item_id BIGINT FK → order_items.id

option_id BIGINT FK → customization_options.id -- original option reference

name VARCHAR(255) -- frozen option name

quantity INT NOT NULL DEFAULT 1

price_delta_cents INT NOT NULL DEFAULT 0

line_delta_cents INT NOT NULL -- price_delta_cents * quantity

TIMESTAMPS:

created_at, updated_at

Indexes:

• idx_item_options_item (order_item_id)

Notes:

• name is stored to remain accurate even if the original option changes later.

• If price_delta_cents was negative (e.g., discount), line_delta_cents will reflect that.

29. Pricing & Immutability Rules

Pricing model:

• Item level: unit_price_cents is captured from product at order time.

• Modifiers: price_delta_cents from customization_options captured at order time.

• Order level:

items_total_cents = SUM(order_items.total_cents)

shipping_cost_cents = from shipping engine decision

discount_cents = discounts applied

tax_cents = computed based on jurisdiction (future)

total_cents = items_total_cents + shipping_cost_cents - discount_cents + tax_cents

Immutability:

• After order.status transitions to 'completed' or 'delivered':

- No price fields may be modified.

- Edits must result in refunds or new orders.

• Soft correction fields may be surfaced via adjustments table (future).

30. Pickup Fulfilment Model

Pickup is the simplest fulfilment path.

Pickup-specific fields:

• fulfilment_type = 'pickup'

• pickup_time optional

• pickup_notes optional

Typical Flow:

1. Order created: status = pending

2. Staff/manager review → set status = accepted

3. Staff begins preparing → status = in_progress

4. Ready at counter → status = ready

5. Customer collects, staff marks → status = completed

Timestamps updated:

• accepted_at when status first becomes accepted

• ready_at when first becomes ready

• completed_at when first becomes completed

Operational impact:

• Dashboard operations view shows pickup orders with clear badges.

• Possible filter: show only ready/pending for pickup screen at counter.

31. Shipping Fulfilment Model

Shipping is a richer workflow.

Shipping-specific fields:

• fulfilment_type = 'shipping'

• shipping_method, shipping_status

• tracking_code, tracking_url

• shipping_name + full address

Typical Flow:

1. Order created: status = pending, shipping_status = 'pending'

2. Merchant accepts → status = accepted

3. Merchant starts packing → status = packing, shipping_status = 'packing'

4. Merchant hands order to carrier:

- tracking_code and tracking_url set

- status = shipped

- shipping_status = 'shipped'

- ready_at set timestamp

5. Customer receives order:

- status = delivered

- shipping_status = 'delivered'

- completed_at timestamp

Edge conditions:

• Lost shipment: shipping_status may become 'returned' or 'lost' (future extension).

• Partial shipments not supported in MVP; future design may split orders.

32. Audit Logging Integration

Every significant order event writes an audit log row.

Audit log for orders includes:

• entity = 'order'

• entity_id = order.id

• action ∈ { 'created','status_changed','shipping_updated','cancelled' }

• meta_json contains:

- previous_status

- new_status

- previous_shipping_status

- new_shipping_status

- actor_user_id

- reason (if cancellation)

- timestamp

Use cases:

• Owner can review fulfilment reliability by staff.

• Disputes can be traced to specific actions and users.

• Security reviews can identify suspicious mass cancellations.

33. Example SQL Queries – Orders & Fulfilment

Query 1 — Get all active orders for a store:

```
SELECT * FROM orders
WHERE merchant_id = ?
AND store_id = ?
AND status IN ('pending','accepted','in_progress','ready','packing','shipped')
ORDER BY placed_at ASC;
```

Query 2 — Fetch order with items and options:

```
SELECT o.*, i.*, io.*
FROM orders o
LEFT JOIN order_items i ON i.order_id = o.id
LEFT JOIN order_item_options io ON io.order_item_id = i.id
WHERE o.id = ?;
```

Query 3 — Count completed orders for billing:

```
SELECT COUNT(*) FROM orders
WHERE merchant_id = ?
AND status IN ('completed','delivered')
AND payment_status = 'paid'
AND placed_at BETWEEN ? AND ?;
```

Query 4 — Update order status with safety:

```
UPDATE orders
SET status = 'ready', ready_at = NOW()
WHERE id = ?
AND status = 'in_progress';
```

Query 5 — Recent shipped orders for customer:

```
SELECT o.*
FROM orders o
WHERE o.customer_id = ?
```

```
AND o.status IN ('shipped','delivered')
ORDER BY o.placed_at DESC
LIMIT 20;
```