Repro Drum Management – Distillation ↔ Inventory Integration

> Draft v0.1 – *living document for the* `feature/repro-drums` *branch*

1 · Overview

Distillation operations occasionally yield material that fails final specification (QC grade ≠ accepted analytical grade). Instead of disposal, this material is stored in **"re-pro" drums** (work-in-progress containers, up to 200 L each) for future re-processing.
This feature adds first-class support for creating, tracking, filling and scanning **repro drums** directly from the distillation record workflow.

2 · Goals

1. Record failed-specification volume during QRD -> Summary step.
2. Allow the operator to:
   • Select an existing *pending* repro drum of the same material, **or**
   • Generate a new repro drum (server action → DB insert → single-label PDF).
3. Maintain accurate drum-level stock via shared `inventory.drums` & `inventory.batches` while respecting different business semantics for `batch_type = 'repro'`.
4. Provide unique, sequential barcodes identical in format to *new* drums.
5. Safeguard against over-filling (>200 L) and enforce status transitions:
   `pending` → (≥ 200 L) scan → `in_stock`.

3 · Business Rules (BR)

BR-1 Only distillations with QC **failed** volumes can create / update repro drums.
BR-2 Experimental quantitative results are linked with each new distillation output volume added to a repro drum.
BR-3 Repro drums must share the same `material_id` as the originating distillation (`v_production_job_details.material_id`).
BR-4 While `status = 'pending'` the drum can accept additional failed volumes until `volume >= 200`.

BR-5 On each fill we log a `production.volume_transfer` row linking `operation_id` ➝ `drum_id` (type: `failed_to_repro`).
BR-6 When the operator scans a full repro drum the status flips to `in_stock` . From that moment the drum is frozen for any further `volume_transfer` rows.

4 • Database Impact

4.1 Existing Entities

```
inventory.drums           (drum_id PK, batch_id FK, serial_number, status, volume, capacity
inventory.batches         (batch_id PK, material_id FK, batch_type, qty_drums, …)
production.operations     (op_id PK, job_id FK, op_type, status, …)
production.volume_transfer (transfer_id PK, op_id FK, drum_id FK, volume, transfer_type,
```

4.2 Proposed Changes

1. **`inventory.batches`**
   - Allow `batch_type = 'repro'` .
   - `supplier_id` & `po_id` become NULLABLE (already true).
2. **`inventory.drums`**
   - No structural change – semantics vary by `batch_type` of parent batch.
3. **`production.volume_transfer`**
   - Add enum value `failed_to_repro` in `transfer_type` domain.
4. **Views**
   - Create `v_repro_drums_pending` for quick lookup of *fillable* repro drums:

   ```
   SELECT d.*
   FROM inventory.drums d
   JOIN inventory.batches b ON b.batch_id = d.batch_id
   WHERE b.batch_type = 'repro'
     AND d.status = 'pending';
   ```

5. **Functions**
   - `fn_next_repro_serial(material_id uuid)` – deterministic next serial number following existing drum format.

All migrations will be delivered via Supabase MCP migrations.

5 • Server-Side Logic

Concern	Action / RPC	Notes
Create repro drum	`createReproDrum(material_id, op_id?)`	1) Inserts batch (if none *pending* for material) 2) Inserts drum row with status `pending`, volume 0 3) Returns serial + PDF bytes.
Add failed volume	`logFailedVolume(op_id, drum_id, volume)`	Wraps insert into `volume_transfer` and updates `drums.volume`. Raises error if >200 L.
Auto-close	trigger `drums_volume_full`	On `UPDATE drums SET volume >= 200` → status `in_stock`.
Label generation	Re-use existing `label-generation.ts` pipeline with `batch_type = 'repro'`.	

6 • Frontend (Apps/Web)

1. **`QRDSummary` additions**
   • New section "Failed Volume Handling".
   • Radio: *Lost to process / Add to repro drum*.
   • If *Add to repro*:
   – Dropdown (Combobox) loading from `v_repro_drums_pending` by `material_id`.
   – "Create New Repro Drum" button → calls `createReproDrum` → downloads PDF → selects newly created drum.
   – Numeric input `volume (L)` pre-filled with failed volume (editable, ≤ failedVolume).
2. **Form state changes** propagate through `onChange` to parent `QRDForm` → saved via existing `updateQRDData`.

7 • API / Routes

```
POST  /api/repro-drums            → createReproDrum
POST  /api/repro-drums/:id/fill   → logFailedVolume
GET   /api/repro-drums/pending?material_id=…
```

All handlers use App Router `route.ts` files and Supabase Server Actions.

8 • Milestones

1. DB migrations & Supabase functions
2. Server actions + PDF generation reuse
3. `QRDSummary` UI / state management
4. End-to-end tests (new distillation → failed volume → repro drum fill)
5. Documentation & training material

9 • Acceptance Criteria

- [ ] Operator can record failed volume and assign to repro drum.
- [ ] New repro drum barcode is generated and downloadable as PDF.
- [ ] Drum volumes accumulate correctly and lock at ≥200 L.
- [ ] Distillation record stores linkage (`volume_transfer`).
- [ ] Inventory views reflect accurate pending vs in-stock repro drums.

10 • Risks & Considerations

- Mis-classification of volumes could inflate WIP stock – mitigate with validation & supervisor approval.
- Concurrency: two operators filling same drum – solved via row-level `SELECT … FOR UPDATE` in `logFailedVolume` .
- Label uniqueness – relies on `fn_next_repro_serial` ; must be strictly atomic.

*This document will evolve during implementation. Contributors: **@Conrad** & Engineering Team.*