08 Validation Plan（検証計画）

本章は、部品/機材管理データベース（MySQL）について、要件（FR）・ユースケース（UC）・トランザクション設計（T）・SQL（tests/・sql/）に基づき、再現可能な検証手順と証跡（スクリーンショット）取得方法を定義する。

目次

- 1. 検証の目的と範囲
- 2. 前提条件
- 3. 証跡（Evidence）方針
- 4. 実行手順（共通）
- 5. 検証観点一覧（FR/UC/T/SQL対応表）
- 6. 必須検証（課題セット 1〜6）
- 7. 追加検証（制約・負系・監査ログ）
- 8. 受入基準（Acceptance Criteria）
- 9. 既知の制約・補足

1. 検証の目的と範囲

目的

- 要件定義（docs/01_requirements.md）に記載した主要機能（登録・配備・使用者/責任者変更・貸出/返却・故障/修理・廃棄/資金元返却・検索）が、DDL/制約/トリガおよびSQL実行で正しく実現できることを確認する。
- 監査ログ（equipment_events）が、INSERT/UPDATE操作に対して自動生成され、操作者（actor）とイベント種別（event_type）、状態遷移（from/to）が追跡可能であることを確認する。
- DB制約（終端状態の固定、最低限の状態遷移制約、貸出/返却先の必須）が機能し、誤操作をDB側で検知・拒否できることを確認する。

範囲

- DBスキーマ（sql/01_schema.sql）
- 初期データ（sql/02_seed.sql）
- 代表トランザクション（sql/04_transaction_cases.sql：T-00〜T-08）
- テストSQL（tests/ 配下：制約・CRUD・貸出/返却・廃棄・検索・ロールバック）
- 証跡（evidence/screenshots/）

2. 前提条件

- MySQL 8.x（Docker Compose）で実行する。
- 本ドキュメントの検証手順は **docker exec**（compose ではなく）を使用し、実行には **root** アカウントを用いる。
- DB名は固定ではなく、環境ごとに任意に設定する。SQLファイル内に `USE sampledb;` のような指定は含めず、実行時にDB名を指定して使用する（後述の例を参照）。
- `equipment` の INSERT/UPDATE に対し、トリガで `equipment_events` が自動生成される。
- トリガ要件：
  - INSERT/UPDATE の前に `@actor_id` と `@event_type` を設定する。
  - `@event_type='loan'` の場合は `@loan_to_id` が必須。
  - `@event_type='return'` の場合は `@return_to_id` が必須。
  - `discarded / returned_to_funder` は終端状態として以後の更新を禁止する（DB制約で強制）。
- **注意**：MySQLのセッション変数（ `@actor_id` など）は接続単位で有効です。複数のコマンドを実行する際は同一接続内で設定してください。

3. 証跡（Evidence）方針

- すべての必須検証について、実行前後の結果を確認できる証跡（スクリーンショット）を保存する。
- 保存先： `evidence/screenshots/`
- ファイル名規約（例）：
  - `V01_schema_tables.png`
  - `V02_seed_counts.png`
  - `V11_register_before_after.png`
  - `V21_loan_before_after.png`
  - `V31_change_user_before_after.png`
  - `V41_project_list.png`
  - `V51_discard_before_after.png`
  - `V52_return_to_funder_before_after.png`
  - `V61_free_search.png`
  - `V71_negative_missing_actor.png`
  - `V72_negative_double_loan.png`
  - `V73_negative_terminal_update.png`

4. 実行手順（共通）

4.0 実行方法（本書の推奨：MySQL端末に貼り付けて手動実行）

本検証は **「SQLファイルの内容を MySQL クライアント（ `mysql>` ）に貼り付けて手動実行する」** 方法を主とする。

- `tests/` や `sql/` のファイルをエディタで開き、必要なブロックをコピーして `mysql>` に貼り付けて実行する。
- **セッション変数（例： `@actor_id` , `@event_type` ）は接続（セッション）単位** で有効なため、 `SET ...;` と `INSERT/UPDATE ...;` は **同じ `mysql>` 画面のまま連続して実行** する。
- 証跡（スクリーンショット）は、原則として **before → action → after** の順で取得する。

> 参考：自動実行（ `docker exec ... < file.sql` ）も可能だが、提出・スクショ重視の観点では、手動貼り付けの方が操作と結果を説明しやすい。

4.1 MySQLへ接続

MySQLコンテナの名前を確認します。

```
docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Ports}}"
```

`NAMES` 列から MySQL のコンテナ名（例： `dbclass-mysql-db-1` など）を選びます。

DB名は環境に合わせて `${DB_NAME}` （または `<DB名>` ）に置き換える。

（例：rootユーザでインタラクティブにログイン）

```
docker exec -it <コンテナ名> mysql -uroot -proot
```

ログイン後は `USE <DB名>;` を実行し、以降のSQLはこの `mysql>` 画面に貼り付けて実行する。

（外部SQLファイルを流し込む場合： `-i` を使用し、リダイレクトでファイルを渡す）

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < sql/01_schema.sql
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < sql/02_seed.sql
```

4.2 DBの初期化（推奨：毎回リセットして再現性を上げる）

0. DB作成（DB名は固定しない）

- テストは `sampledb` に固定せず、任意のDB名（例： `equipmentdb` ）を使うことができる。
- 環境変数 `DB_NAME` にDB名を設定する例と、単純なプレースホルダ `<DB名>` の例を示す。

```
export DB_NAME=equipmentdb
# または
DB_NAME=<DB名>
```

- Docker exec環境でDBを作成するコマンド例：

```
docker exec -i <コンテナ名> mysql -uroot -proot -e "CREATE DATABASE IF NOT EXISTS ${DB_NAME} CHARACTE
```

- （任意）完全リセットしたい場合は、DBを削除してから作成することもできる：

```
docker exec -i <コンテナ名> mysql -uroot -proot -e "DROP DATABASE IF EXISTS ${DB_NAME};"
```

1. スキーマ適用

- 手動（推奨）： `sql/01_schema.sql` を開き、内容を `mysql>` に貼り付けて実行する。
- 自動（任意）：

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < sql/01_schema.sql
```

2. 初期データ投入

- 手動（推奨）： `sql/02_seed.sql` を開き、内容を `mysql>` に貼り付けて実行する。
- 自動（任意）：

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < sql/02_seed.sql
```

3. テスト（tests/01〜07）を順に実行（推奨）

- 手動（推奨）：各 `tests/0X_.../*.sql` を開き、**before** → **action** → **after** の順で `mysql>` に貼り付けて実行する。
  - 例： `tests/01_constraints/before.sql` → `tests/01_constraints/action.sql` → `tests/01_constraints/after.sql`
- 自動（任意）：

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/03_loan_return/before.sql
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/03_loan_return/action.sql
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/03_loan_return/after.sql
```

他のテストも同様に before/action/after のパターンで実行可能。

（任意）代表トランザクション（sql/04_transaction_cases.sql）を実行

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < sql/04_transaction_cases.sql
```

4.3 最低限の動作確認（スキーマ/トリガ）

rootユーザで `4.1` の方法でログインし、必要に応じて対象DBを選択（例： `USE <DB名>;` ）してから、以下を実行し、証跡を保存する。

```
SHOW TABLES;
SHOW TRIGGERS;
```

5. 検証観点一覧（FR/UC/T/SQL対応表）

観点	対応FR/UC	参照トランザクション/SQL	期待結果（概要）
登録（購入→台帳登録）	FR-01 / UC-01	T-00 / tests/02_basic_crud/before	action
配備（project/location/manager設定）	FR-02 / UC-02	T-01 / tests/02_basic_crud/before	action
使用者変更（A→B）	FR-03 / UC-03	T-02 / tests/02_basic_crud/before	action
貸出（loan）	FR-04 / UC-04	T-03 / tests/03_loan_return/before	action
返却（return）	FR-04 / UC-04	T-04 / tests/03_loan_return/before	action
故障・修理	FR-05 / UC-05	T-05/T-06 / tests/02_basic_crud/before	action
廃棄	FR-06 / UC-06	T-07 / tests/04_discard/before	action
資金元返却	FR-06 / UC-06	T-08 / tests/05_return_to_funder/before	action
複合検索	FR-07 / UC-07	sql/05_free_queries.sql / tests/05_search.sql	条件検索が正しく動作
監査ログ	NFR（監査/	equipment_events + tests/01_constraints/before	action

観点	対応FR/UC	参照トランザクション/SQL	期待結果（概要）
	追跡）		

6. 必須検証（課題セット 1〜6）

ここは「提出用チェックリスト」の必須検証に対応する。各課題は **tests/0X_*.sql** を主軸に説明し、補助的にSQL例を示す。

課題1：部品/機材登録（tests/02_basic_crud/before.sql 等）

- 対象：tests/02_basic_crud/before.sql、tests/02_basic_crud/action.sql、tests/02_basic_crud/after.sql（登録・CRUD検証）、参考：T-00（register_purchase）
- 手順：
  i. 対象の `tests/02_basic_crud/before.sql` → `tests/02_basic_crud/action.sql` → `tests/02_basic_crud/after.sql` の順に開き、 `mysql>` に貼り付けて実行する。
  ii. 出力をスクリーンショットに保存する。
  iii. （任意：自動実行）

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/02_basic_crud/action.sql
```

- 補助確認SQL（任意）：

```
SELECT COUNT(*) FROM equipment;
SELECT * FROM equipment ORDER BY equipment_id DESC LIMIT 10;
SELECT * FROM equipment_events ORDER BY event_timestamp DESC LIMIT 20;
```

- 証跡： `V11_register_before_after.png` （tests実行前後の出力）

課題2：貸出前後（tests/03_loan_return/before.sql 等）

- 対象：tests/03_loan_return/before.sql、tests/03_loan_return/action.sql、tests/03_loan_return/after.sql、参考：T-03（loan）
- 手順：
  i. 対象の `tests/03_loan_return/before.sql` → `tests/03_loan_return/action.sql` → `tests/03_loan_return/after.sql` の順に開き、 `mysql>` に貼り付けて実行する。
  ii. 出力をスクリーンショットに保存する。
  iii. （任意：自動実行）

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/03_loan_return/action.sql
```

- 補助確認SQL（任意）：

```
SELECT equipment_id, status_code FROM equipment ORDER BY equipment_id DESC LIMIT 10;
SELECT * FROM equipment_events ORDER BY event_timestamp DESC LIMIT 20;
```

- 証跡： `V21_loan_before_after.png` （tests実行前後の出力）

課題3：使用者／責任者変更前後（tests/02_basic_crud/before.sql 等）

- 対象：tests/02_basic_crud/before.sql、tests/02_basic_crud/action.sql、tests/02_basic_crud/after.sql（使用者・責任者変更部分）、参考：T-02（change_user）
- 手順：
  i. 対象の `tests/02_basic_crud/before.sql` → `tests/02_basic_crud/action.sql` → `tests/02_basic_crud/after.sql` の順に開き、 `mysql>` に貼り付けて実行する。
  ii. 出力をスクリーンショットに保存する。
  iii. （任意：自動実行）

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/02_basic_crud/action.sql
```

- 補助確認SQL（任意）：

```
SELECT equipment_id, user_id, manager_id FROM equipment ORDER BY equipment_id DESC LIMIT 10;
SELECT * FROM equipment_events ORDER BY event_timestamp DESC LIMIT 20;
```

- 証跡： `V31_change_user_before_after.png` （tests実行前後の出力）

課題4：特定プロジェクトの部品/機材一覧（tests/05_search.sql）

- 対象：tests/05_search.sql、参考：FR-07（複合検索）
- 手順：
  i. 対象の `tests/05_search.sql` ファイルを開き、 `mysql>` に貼り付けて実行する。
  ii. 出力をスクリーンショットに保存する。
  iii. （任意：自動実行）

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/05_search.sql
```

- 補助確認SQL（任意）：

```
SELECT e.equipment_id, e.name, e.status_code, e.location_id, e.manager_id, e.user_id
FROM equipment e
WHERE e.project_id IS NOT NULL
ORDER BY e.equipment_id DESC
LIMIT 20;
```

- 証跡： `V41_project_list.png` （tests実行前後の出力）

課題5：廃棄処理（tests/04_discard/before.sql 等）

- 対象：tests/04_discard/before.sql、tests/04_discard/action.sql、tests/04_discard/after.sql、参考：T-07（discard）
- 手順：
  i. 対象の `tests/04_discard/before.sql` → `tests/04_discard/action.sql` → `tests/04_discard/after.sql` の順に開き、 `mysql>` に貼り付けて実行する。
  ii. 出力をスクリーンショットに保存する。
  iii. （任意：自動実行）

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/04_discard/action.sql
```

- 補助確認SQL（任意）：

```
SELECT equipment_id, status_code FROM equipment ORDER BY equipment_id DESC LIMIT 10;
SELECT * FROM equipment_events ORDER BY event_timestamp DESC LIMIT 20;
```

- 証跡：
  - `V51_discard_before_after.png` （tests実行前後の出力）
  - `V73_negative_terminal_update.png` （終端後更新拒否の負系テスト出力）

課題6：複合条件検索（自由）（tests/05_search.sql）

- 対象：tests/05_search.sql、参考：FR-07（複合検索）
- 手順：
  i. 対象の `tests/05_search.sql` ファイルを開き、 `mysql>` に貼り付けて実行する。
  ii. 出力をスクリーンショットに保存する。
  iii. （任意：自動実行）

```
docker exec -i <コンテナ名> mysql -uroot -proot ${DB_NAME} < tests/05_search.sql
```

- 補助確認SQL（任意）：

```
SELECT DISTINCT e.equipment_id, e.name
FROM equipment e
JOIN equipment_events ev ON ev.equipment_id = e.equipment_id
WHERE ev.to_status_code = 'loaned'
  AND ev.event_timestamp >= (NOW() - INTERVAL 30 DAY)
ORDER BY e.equipment_id DESC
LIMIT 20;
```

- 証跡： `V61_free_search.png` （tests実行前後の出力）

7. 追加検証（制約・負系・監査ログ）

7.1 負系：@actor_id 未設定（tests/01_constraints/action.sql）

- 実行： `tests/01_constraints/action.sql` に含まれる負系テストを実行し、操作者必須制約が機能することを確認する。
- 証跡： `V71_negative_missing_actor.png` （tests実行時のエラー出力）

補助：手動で試す場合

```
-- 注意：セッション変数は同一接続内で設定する必要があります。
SET @actor_id := NULL;
SET @event_type := 'deploy';
UPDATE equipment SET location_id = @loc_lab WHERE equipment_id = @eid_sa;
```

7.2 負系：二重貸出の禁止（tests/01_constraints/action.sql）

- 実行： `tests/01_constraints/action.sql` に含まれる二重貸出禁止の負系テストを実行し、エラーが発生することを確認する。
- 証跡： `V72_negative_double_loan.png` （tests実行時のエラー出力）

補助：手動で試す場合

```
SET @actor_id := @uid_member;
SET @event_type := 'loan';
SET @loan_to_id := @uid_viewer;
UPDATE equipment SET status_code = 'loaned' WHERE equipment_id = @eid_sa;
```

7.3 負系：貸出中の廃棄/資金元返却禁止（tests/07_transaction_rollback/action.sql）

- 実行： `tests/07_transaction_rollback/action.sql` に含まれる貸出中の廃棄/返却禁止の負系テストを実行し、エラーが発生することを確認する。
- 証跡： `V73_negative_terminal_update.png` （tests実行時のエラー出力）
- このテストは、トランザクション内の貸出更新とそれに伴う `equipment_events` レコードが `ROLLBACK` により取り消されることも検証する。

補助：手動で試す場合

```
SET @actor_id := @uid_admin;
SET @event_type := 'discard';
UPDATE equipment SET status_code = 'discarded' WHERE equipment_id = @eid_sa;
```

7.4 監査ログの整合（tests/01_constraints/action.sql）

- 実行： `tests/01_constraints/action.sql` で監査ログの整合性を検証する。
- 手動実行の場合は、最後に `tests/01_constraints/after.sql` の確認SQLも貼り付けて、eventsの記録（actor/event_type/from-to）をスクショする。
- 証跡：監査ログに `actor / event_type / from/to` が適切に記録されていることを示す出力をスクリーンショットで保存。

8. 受入基準（Acceptance Criteria）

- 必須検証（課題セット 1〜6）を実行し、すべて期待結果を満たす。
- `SHOW TABLES;` と `SHOW TRIGGERS;` でスキーマとトリガが確認できる。
- equipment の INSERT/UPDATE で `equipment_events` が自動生成され、 `@actor_id/@event_type` が必須として機能する。
- loan/return において `loan_to_id/return_to_id` が events に保存される。
- 終端状態（discarded/returned_to_funder）になった機材は以後更新できない。
- 主要な負系（actor未設定、二重貸出、貸出中の廃棄/返却）がDBで拒否される。
- すべての証跡が `evidence/screenshots/` に揃っている。

9. 既知の制約・補足

- 本プロジェクトは「授業課題の解答例」として、DB内で監査ログ・最低限の制約を強めに実装している。
- 将来Web化する場合は、アプリ側で認証・認可（role）を担い、DB側は監査ログ・整合性・禁止遷移を担保する設計が有力。
- テストを何度でも再実行できるよう、 `sql/04_transaction_cases.sql` は T-00 で毎回新しい機材を作成し、そのIDを後続トランザクションで使用する。