06 Transaction Design（トランザクション設計）

目次

本章では、研究室の部品/機材管理DBにおける主要業務（配備・使用者変更・貸出/返却・故障/修理・廃棄・資金元返却）を、**一貫性を保つためのトランザクション単位**として定義する。

本リポジトリでは **方式A** を採用する：

- `equipment` を INSERT/UPDATE すると、トリガにより `equipment_events` が自動生成される（方針2：登録ログも残す）
- INSERT/UPDATE 前に必ずセッション変数 `@actor_id` と `@event_type` を設定する
- `equipment_events` へ直接 INSERT しない（履歴の一元性を保つ）

1. 前提（対象テーブルと設計方針）

1.1 主な対象テーブル

- `equipment` ：機材/部品の現在状態（現在の割当・場所・責任者・プロジェクト・状態）
- `equipment_events` ：状態遷移・管理情報変更の履歴（トリガで自動記録）
- `people` / `projects` / `locations` / `vendors` / `purchases` ：参照マスタ
- `equipment_type_reference` / `equipment_status_reference` ：参照テーブル（code 参照）

1.2 状態（status_code）

本システムの代表状態（seed で投入）

- `in_stock` ：在庫（利用可能）
- `in_service` ：稼働中（設置され稼働）
- `assigned` ：使用中（学内個人に割当）
- `loaned` ：貸出中（学外/外部含む）
- `broken` ：故障中
- `repairing` ：修理中
- `discarded` ：廃棄済み（原則戻さない）
- `returned_to_funder` ：資金元へ返却（原則運用終了）

2. トランザクション設計の基本ルール

2.1 セッション変数（方式Aの前提）

`equipment` の INSERT/UPDATE を行う前に、必ず以下を設定する。

```
SET @actor_id  = <people.id>;       -- 操作者（必須）
SET @event_type = '<event_type>';   -- 更新意図（必須）
```

未設定のまま INSERT/UPDATE すると、 `BEFORE INSERT/UPDATE` トリガによりエラーとなる。

2.2 更新は「BEGIN→COMMIT」を基本とする

- 単一UPDATEでも、業務操作は原則トランザクションとして扱う
- 途中で失敗した場合は `ROLLBACK` し、部分更新を残さない

```
START TRANSACTION;
-- ...（INSERT/UPDATE / SELECT 確認）...
COMMIT;
```

2.3 ロックと分離（InnoDB）

- `equipment` の更新は同一行に対して排他ロックを取得する
- 競合を避けるため、更新前に対象レコードを `SELECT ... FOR UPDATE` でロックしてから更新する手順を推奨する

```
START TRANSACTION;
SELECT * FROM equipment WHERE equipment_id = ? FOR UPDATE;
-- UPDATE ...
COMMIT;
```

3. 代表トランザクション（業務別）

各トランザクションは将来のWeb運用を見据え、(1)入力パラメータ（APIが受け取るべき値）と、(2)成功条件（DB上で起きるべき更新と履歴追加）を明示する。

3.1 Requirements / Use case との対応関係

本章の T-00〜T-08 は、 `docs/01_requirements.md` に定義した機能要件（FR）および代表ユースケース（UC）を、DB更新の最小単位（トランザクション）として具体化したものである。

Transaction（本章）	対応する機能要件（FR）	対応するユースケース（UC）	主に満たす非機能要件（NFR）
T-00 新規登録（register_purchase）	FR-01 新規購入部品/機材の登録	UC-01 新規登録（購入・台帳登録）	NFR-01（整合性），NFR-02（監査性），NFR-03（トランザクション）

Transaction（本章）	対応する機能要件（FR）	対応するユースケース（UC）	主に満たす非機能要件（NFR）
T-01 配備（deploy）	FR-02 配備（設置・稼働開始）	UC-02 配備（設置・稼働開始）	NFR-01（整合性），NFR-02（監査性），NFR-03（トランザクション）
T-02 使用者変更（change_user）	FR-03 使用者・責任者の変更	UC-03 使用者/責任者変更	NFR-01, NFR-02, NFR-03
T-03 貸出（loan）	FR-04 貸出・返却	UC-04 貸出/返却	NFR-01, NFR-02, NFR-03
T-04 返却（return）	FR-04 貸出・返却	UC-04 貸出/返却	NFR-01, NFR-02, NFR-03
T-05 故障登録（report_broken）	FR-05 故障・修理	UC-05 故障/修理	NFR-01, NFR-02, NFR-03
T-06 修理開始/完了（start/finish_repair）	FR-05 故障・修理	UC-05 故障/修理	NFR-01, NFR-02, NFR-03
T-07 廃棄（discard）	FR-06 廃棄・資金元返却	UC-06 廃棄/資金元返却	NFR-01, NFR-02, NFR-03
T-08 資金元返却（return_to_funder）	FR-06 廃棄・資金元返却	UC-06 廃棄/資金元返却	NFR-01, NFR-02, NFR-03

補足：本章では更新系操作（T-00〜T-08）の実行者は `admin / member` を想定し、認可（NFR-05）はアプリ層または運用ルールで担保する（DBでは `@actor_id` の必須化までを強制する）。

> ここに記載した SQL は、実行例として `sql/04_transaction_cases.sql` にも反映する。

T-00 新規登録（購入・台帳登録）

目的：購入情報と機材情報を登録し、登録時点の監査ログ（event）を必ず残す。

入力パラメータ（API想定）

- `actor_id`（操作者ID = `people.id`）
- `event_type`（例：`register_purchase`）
- `vendor_id`（既存業者ID）または `vendor`（新規業者情報）

- `order_date`, `delivery_date`, `purchase_date`, `price`, `note`（購入情報）
- `name`, `model`, `quantity`, `unit`, `equipment_type_code`, `status_code`（機材情報：初期は原則 `in_stock`）
- `project_id`, `location_id`, `manager_id`, `user_id`（管理情報： `user_id` は NULL 可）
- （備品・資産の場合） `university_id`, `funding_id`（識別子）

成功条件（DB上の期待結果）

- `purchases` が1行追加される（`vendor_id` を参照）
- `equipment` が1行追加される（`purchase_id` を参照）
- （必要な場合） `equipment_identifiers` が1行追加される
- **登録時点で `equipment_events` に1行追加される**（トリガにより自動）
  - `event_type='register_purchase'`（= `@event_type`）
  - `from_status_code/to_status_code` は同値（登録ログのため）
  - `actor_id` と `event_timestamp` が記録される

方式A（トリガ）に関する補足

- 方針2として、 `equipment` の **INSERT** に対してもトリガを定義し、登録時点のイベントを必ず記録する。
- そのため INSERT 前にも `SET @actor_id`, `SET @event_type` が必須である。

```
START TRANSACTION;

SET @actor_id  = (SELECT id FROM people WHERE user_name='admin01');
SET @event_type = 'register_purchase';

-- 1) purchases
INSERT INTO purchases (vendor_id, order_date, delivery_date, purchase_date, price, note
VALUES (
  (SELECT id FROM vendors WHERE name='サンプル商事'),
  '2025-12-01', '2025-12-05', '2025-12-05',
  120000.00,
  '授業用サンプル購入'
);

-- 2) equipment
INSERT INTO equipment (
  name, model, quantity, unit,
  equipment_type_code, status_code,
  purchase_id,
  project_id, location_id, manager_id, user_id
) VALUES (
  'オシロスコープ', 'DS1054Z', 1, '台',
  'equipment', 'in_stock',
  LAST_INSERT_ID(),
  (SELECT id FROM projects  WHERE project_no='PJ-2025-001'),
  (SELECT id FROM locations WHERE name='倉庫（別館1F）'),
  (SELECT id FROM people    WHERE user_name='admin01'),
  NULL
);

-- 3) identifiers（必要な場合のみ。例：資産/備品）
-- INSERT INTO equipment_identifiers (equipment_id, university_id, funding_id)
-- VALUES (LAST_INSERT_ID(), 'UNI-0001', 'FUND-0001');

COMMIT;
```

> 注： `equipment` の INSERT 後、トリガにより `equipment_events` が自動で1行追加される（登録ログ）。

T-01 配備（project/location/manager を確定し、必要に応じて稼働にする）

目的：購入/登録済みの機材をプロジェクトへ配備し、設置場所と責任者を確定する。

入力パラメータ（API想定）

- `equipment_id`（対象機材ID）
- `actor_id`（操作者ID＝`people.id`）
- `project_id`（配備先プロジェクトID＝`projects.id`）
- `location_id`（設置場所ID＝`locations.id`）
- `manager_id`（責任者ID＝`people.id`）
- `to_status_code`（例：`in_service`。配備で稼働開始する場合）

成功条件（DB上の期待結果）

- `equipment` が更新される（少なくとも `project_id`, `location_id`, `manager_id`。必要に応じて `status_code`）
- `equipment_events` に1行追加される
  - `event_type='deploy'`
  - `from_status_code/to_status_code`, `from_manager_id/to_manager_id`, `actor_id`, `event_timestamp` が記録される

```
START TRANSACTION;

SET @actor_id  = (SELECT id FROM people WHERE user_name='admin01');
SET @event_type = 'deploy';

SELECT * FROM equipment WHERE equipment_id = 1 FOR UPDATE;

UPDATE equipment
SET project_id   = (SELECT id FROM projects  WHERE project_no='PJ-2025-001'),
    location_id  = (SELECT id FROM locations WHERE name='研究室A（本館3F）'),
    manager_id   = (SELECT id FROM people    WHERE user_name='admin01'),
    status_code  = 'in_service'
WHERE equipment_id = 1;

COMMIT;
```

T-02 使用者変更（A → B の割当変更）

目的：学内の使用者（ `user_id` ）を変更し、履歴を残す。

入力パラメータ（API想定）

- `equipment_id`
- `actor_id`
- `to_user_id` （新しい使用者ID = `people.id` ）
- `to_status_code` （通常 `assigned` ）

成功条件（DB上の期待結果）

- `equipment.user_id` が `to_user_id` に更新される
- `equipment.status_code` が必要に応じて更新される（例： `assigned` ）
- `equipment_events` に1行追加される
  - `event_type='change_user'`
  - `from_user_id/to_user_id` が記録される

```
START TRANSACTION;

SET @actor_id  = (SELECT id FROM people WHERE user_name='admin01');
SET @event_type = 'change_user';

SELECT * FROM equipment WHERE equipment_id = 2 FOR UPDATE;

UPDATE equipment
SET user_id = (SELECT id FROM people WHERE user_name='viewer01'),
    status_code = 'assigned'
WHERE equipment_id = 2;

COMMIT;
```

T-03 貸出（loaned へ）

目的：貸出登録に相当する状態遷移を行い、履歴を残す。

入力パラメータ（API想定）

- `equipment_id`

- actor_id
- to_status_code （ loaned ）
- （将来拡張） loan_to_id （貸出先ID。外部/学外の貸出先を表す）
- （将来拡張） return_due_date （返却期限）

成功条件（DB上の期待結果）

- equipment.status_code が loaned に更新される
- equipment_events に1行追加される
  - event_type='loan'
  - from_status_code/to_status_code と actor_id が記録される
- 補足：現行トリガ実装では loan_to_id や期限情報は自動記録しない（厳密化するなら別テーブル or トリガ拡張）

```
START TRANSACTION;

SET @actor_id  = (SELECT id FROM people WHERE user_name='member01');
SET @event_type = 'loan';

SELECT * FROM equipment WHERE equipment_id = 2 FOR UPDATE;

UPDATE equipment
SET status_code = 'loaned'
WHERE equipment_id = 2;

COMMIT;
```

> 注：現行スキーマでは equipment_events.loan_to_id 等はトリガで自動投入していない。
> 貸出先情報を厳密に残す場合は、(a) 別テーブル loans を設ける、または (b) トリガを拡張してセッション変数で渡す、等の拡張を検討する。

T-04 返却（loaned → in_stock / in_service）

目的：返却に相当する状態遷移を行い、戻り先（場所）も反映する。

入力パラメータ（API想定）

- equipment_id
- actor_id

- to_status_code （例： in_stock または in_service ）
- to_location_id （返却先の場所ID）
- （将来拡張） return_to_id （返却受付担当や返却先エンティティを表すID）

成功条件（DB上の期待結果）

- equipment.status_code が返却後の状態に更新される
- equipment.location_id が返却先に更新される
- equipment_events に1行追加される
  - event_type='return'
  - from_status_code/to_status_code , actor_id が記録される

```
START TRANSACTION;

SET @actor_id  = (SELECT id FROM people WHERE user_name='member01');
SET @event_type = 'return';

SELECT * FROM equipment WHERE equipment_id = 2 FOR UPDATE;

UPDATE equipment
SET status_code = 'in_stock',
    location_id = (SELECT id FROM locations WHERE name='倉庫（別館1F）')
WHERE equipment_id = 2;

COMMIT;
```

T-05 故障登録（broken）

目的：使用不可（故障）を記録する。

入力パラメータ（API想定）

- equipment_id
- actor_id
- to_status_code （ broken ）
- （将来拡張） note （故障内容・状況メモ）

成功条件（DB上の期待結果）

- equipment.status_code が broken に更新される

- equipment_events に1行追加される
  - event_type='report_broken'
  - from_status_code/to_status_code , actor_id が記録される

```
START TRANSACTION;

SET @actor_id  = (SELECT id FROM people WHERE user_name='member01');
SET @event_type = 'report_broken';

SELECT * FROM equipment WHERE equipment_id = 2 FOR UPDATE;

UPDATE equipment
SET status_code = 'broken'
WHERE equipment_id = 2;

COMMIT;
```

T-06 修理開始/完了（repairing → in_service）

目的：修理中への遷移と、修理完了後の復帰を記録する。

入力パラメータ（API想定）

- equipment_id
- actor_id
- to_status_code （修理開始： repairing 、修理完了： in_service など）
- （将来拡張） repair_vendor_id （修理先業者ID）
- （将来拡張） repair_ticket_no （修理受付番号）

成功条件（DB上の期待結果）

- 修理開始： equipment.status_code='repairing' に更新され、 equipment_events が1件追加（ event_type='start_repair' ）
- 修理完了： equipment.status_code が復帰状態に更新され、 equipment_events が1件追加（ event_type='finish_repair' ）

```sql
-- 修理開始
START TRANSACTION;
SET @actor_id  = (SELECT id FROM people WHERE user_name='admin01');
SET @event_type = 'start_repair';
SELECT * FROM equipment WHERE equipment_id = 2 FOR UPDATE;
UPDATE equipment SET status_code = 'repairing' WHERE equipment_id = 2;
COMMIT;

-- 修理完了
START TRANSACTION;
SET @actor_id  = (SELECT id FROM people WHERE user_name='admin01');
SET @event_type = 'finish_repair';
SELECT * FROM equipment WHERE equipment_id = 2 FOR UPDATE;
UPDATE equipment SET status_code = 'in_service' WHERE equipment_id = 2;
COMMIT;
```

T-07 廃棄（discarded）

目的：廃棄を記録し、状態を終端にする。

入力パラメータ（API想定）

- `equipment_id`
- `actor_id`
- `to_status_code`（`discarded`）
- （将来拡張）`discarded_at`（廃棄日）
- （将来拡張）`reason`（廃棄理由）

成功条件（DB上の期待結果）

- `equipment.status_code` が `discarded` に更新される
- `equipment_events` に1行追加される
  - `event_type='discard'`
- 運用上の期待：`discarded` は終端であり、原則として状態を戻さない

```
START TRANSACTION;

SET @actor_id  = (SELECT id FROM people WHERE user_name='admin01');
SET @event_type = 'discard';

SELECT * FROM equipment WHERE equipment_id = 1 FOR UPDATE;

UPDATE equipment
SET status_code = 'discarded'
WHERE equipment_id = 1;

COMMIT;
```

> 運用ルール： `discarded` は原則として戻さない（復帰操作を禁止する場合は、追加のトリガ/制約で実装可能）。

T-08 資金元返却（returned_to_funder）

目的：資金元への返却を記録し、運用終了状態にする。

入力パラメータ（API想定）

- `equipment_id`
- `actor_id`
- `to_status_code`（`returned_to_funder`）
- （将来拡張） `funder_name` / `funder_return_ref`（資金元返却の管理情報）

成功条件（DB上の期待結果）

- `equipment.status_code` が `returned_to_funder` に更新される
- `equipment_events` に1行追加される
  - `event_type='return_to_funder'`
- 運用上の期待： `returned_to_funder` は終端であり、原則として状態を戻さない

```sql
START TRANSACTION;

SET @actor_id  = (SELECT id FROM people WHERE user_name='admin01');
SET @event_type = 'return_to_funder';

SELECT * FROM equipment WHERE equipment_id = 2 FOR UPDATE;

UPDATE equipment
SET status_code = 'returned_to_funder'
WHERE equipment_id = 2;

COMMIT;
```

> 補足：資産（ `equipment_type_code='asset'` ）のみ許可したい場合は、
> (a) アプリ側でチェックする、または (b) BEFORE UPDATE トリガで type を参照して拒否する、等で厳密化できる。

4. エラー/ROLLBACK 設計

4.1 典型エラーケース

- **E-01**： `@actor_id` 未設定で INSERT/UPDATE → トリガがエラー
- **E-02**： `@event_type` 未設定で INSERT/UPDATE → トリガがエラー
- **E-03**：存在しない `project_id` を設定 → 外部キー制約でエラー

4.2 ROLLBACK の確認（例）

```
START TRANSACTION;

-- あえて設定しない
-- SET @actor_id = ...
-- SET @event_type = ...

SELECT * FROM equipment WHERE equipment_id = 1 FOR UPDATE;

-- ここでトリガがエラーとなり、更新できない
UPDATE equipment SET status_code='loaned' WHERE equipment_id = 1;

ROLLBACK;
```

5. 検証観点（提出用）

- 各トランザクションについて、以下を **Before/After のスクリーンショット**で提出する
  - `equipment` の該当行（状態・user・manager・project・location）
  - `equipment_events` の追加行（event_type, from/to, actor, timestamp）

推奨クエリ：

```
-- 対象機材の現在状態
SELECT * FROM equipment WHERE equipment_id = ?;

-- 履歴（時系列）
SELECT *
FROM equipment_events
WHERE equipment_id = ?
ORDER BY event_timestamp;
```

6. 今後の拡張（任意）

現行は授業課題・最小実装として「INSERT/UPDATE → 自動履歴」を優先している。
実運用寄りに拡張する場合は、以下を検討する。

- 貸出情報の厳密化： `loans` （貸出テーブル）を追加し、貸出先/返却先/担当/期間を管理
- 終端状態の更新禁止： `discarded` / `returned_to_funder` からの復帰をトリガで拒否
- 権限制御： `people.role` に基づく操作制限をアプリ層（またはストアド）で実装
- 監査強化：event_type を参照テーブル化し、許可される遷移を定義（状態遷移表）