07 Query Design（検索設計）

本章では、要件（FR-07 多条件検索）およびユースケース（UC-07）に基づき、部品/機材管理DBにおける代表的な検索（Query）を設計する。

- 目的：
  - 運用で頻出する「一覧・検索・履歴参照」を安定して実行できること
  - 将来Web化（API化）を見据えて、入力パラメータ・出力項目を明確にすること
- 参照：
  - スキーマ： `sql/01_schema.sql`
  - 代表トランザクション： `sql/04_transaction_cases.sql`
  - 自由検索（提出用）： `sql/05_free_queries.sql`
  - テスト： `tests/05_search.sql`

1. 前提（データモデル上のポイント）

- 現在状態（current）は `equipment` テーブルに保持する。
- 状態遷移・担当者変更などの履歴（history）は `equipment_events` に自動記録される。
- 貸出先/返却先は、運用上 `equipment` の現在状態だけでは表現しにくいため、
  - `event_type='loan'` の `equipment_events.loan_to_id`
  - `event_type='return'` の `equipment_events.return_to_id`
    に記録する（トリガで強制）。

2. 検索の分類

2.1 現在状態の検索（equipment中心）

- 例：プロジェクト別一覧、設置場所別、責任者別、状態別、種類別

2.2 履歴（events）を含む検索

- 例：期間内に loaned になった機材、直近の状態遷移、変更履歴の時系列

2.3 購入・識別子の検索

- 例：販売元別購入一覧、学内番号/資金元番号から機材を逆引き

3. 推奨インデックス（物理設計との対応）

本プロジェクトのDDLでは、以下のような検索向けインデックスが用意されている（ `sql/01_schema.sql` 参照）。

- `equipment`
  - `idx_equipment_status (status_code)`
  - `idx_equipment_type (equipment_type_code)`
  - `idx_equipment_project (project_id)`
  - `idx_equipment_location (location_id)`
  - `idx_equipment_manager (manager_id)`
  - `idx_equipment_user (user_id)`
- `equipment_events`
  - `idx_events_equipment_time (equipment_id, event_timestamp)`
  - `idx_events_actor_time (actor_id, event_timestamp)`
  - `idx_events_to_status_time (to_status_code, event_timestamp)`
- `purchases`
  - `idx_purchases_vendor (vendor_id)`
  - `idx_purchases_purchase_date (purchase_date)`

必要に応じて、将来の本格運用では以下も検討できる。

- `equipment (project_id, status_code)` の複合インデックス（プロジェクト×状態の頻出検索向け）
- `equipment (location_id, status_code)` の複合インデックス（場所×状態の頻出検索向け）

4. クエリカタログ（代表クエリ）

以下は、アプリ/API化を想定して「入力パラメータ」「出力」「SQL例」をセットで定義する。

Q-01 機材の多条件検索（現在状態）

- 対応：FR-07 / UC-07
- 入力（任意）：
  - `project_id`, `equipment_type_code`, `status_code`, `location_id`, `manager_id`, `user_id`
  - `name_like`（部分一致：名前）
  - `model_like`（部分一致：型番）
- 出力：機材の一覧（現在状態）

```sql
SELECT
  e.equipment_id,
  e.name,
  e.model,
  e.quantity,
  e.unit,
  e.equipment_type_code,
  e.status_code,
  e.project_id,
  e.location_id,
  e.manager_id,
  e.user_id
FROM equipment e
WHERE 1=1
  AND (:project_id IS NULL OR e.project_id = :project_id)
  AND (:equipment_type_code IS NULL OR e.equipment_type_code = :equipment_type_code)
  AND (:status_code IS NULL OR e.status_code = :status_code)
  AND (:location_id IS NULL OR e.location_id = :location_id)
  AND (:manager_id IS NULL OR e.manager_id = :manager_id)
  AND (:user_id IS NULL OR e.user_id = :user_id)
  AND (:name_like IS NULL OR e.name LIKE CONCAT('%', :name_like, '%'))
  AND (:model_like IS NULL OR e.model LIKE CONCAT('%', :model_like, '%'))
ORDER BY e.equipment_id DESC;
```

> 注：MySQL CLIでの検証では `:param` ではなく値を直書きする。提出用の自由検索は `sql/05_free_queries.sql` に具体値版を置く。

Q-02 特定プロジェクトの機材一覧（基本）

- 対応：課題4（プロジェクト部品一覧）
- 入力： `project_id`
- 出力：プロジェクト内の機材一覧

```
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 = :project_id
ORDER BY e.equipment_id DESC;
```

Q-03 期間内に「loaned になった」機材（履歴ベース）

- 対応：課題6（複合条件検索）
- 入力： `project_id` （任意）, `from_ts` , `to_ts`
- 出力：期間内に loaned へ遷移した機材

```
SELECT DISTINCT
  e.equipment_id,
  e.name
FROM equipment e
JOIN equipment_events ev
  ON ev.equipment_id = e.equipment_id
WHERE (:project_id IS NULL OR e.project_id = :project_id)
  AND ev.to_status_code = 'loaned'
  AND ev.event_timestamp >= :from_ts
  AND ev.event_timestamp <  :to_ts
ORDER BY e.equipment_id DESC;
```

Q-04 現在「貸出中」の一覧 + 最新貸出先（eventsから取得）

- 対応：FR-04（貸出/返却）× FR-07（検索）
- 入力（任意）： `project_id` , `location_id`
- 出力：貸出中機材 + 直近の貸出先（people）

```sql
SELECT
  e.equipment_id,
  e.name,
  e.status_code,
  last_loan.event_timestamp AS loaned_at,
  p_to.full_name AS loan_to_name,
  p_to.email     AS loan_to_email
FROM equipment e
JOIN (
  SELECT
    ev1.equipment_id,
    ev1.loan_to_id,
    ev1.event_timestamp
  FROM equipment_events ev1
  JOIN (
    SELECT equipment_id, MAX(event_timestamp) AS max_ts
    FROM equipment_events
    WHERE event_type = 'loan'
    GROUP BY equipment_id
  ) t
    ON t.equipment_id = ev1.equipment_id
   AND t.max_ts = ev1.event_timestamp
  WHERE ev1.event_type = 'loan'
) last_loan
  ON last_loan.equipment_id = e.equipment_id
LEFT JOIN people p_to
  ON p_to.id = last_loan.loan_to_id
WHERE e.status_code = 'loaned'
  AND (:project_id IS NULL OR e.project_id = :project_id)
  AND (:location_id IS NULL OR e.location_id = :location_id)
ORDER BY last_loan.event_timestamp DESC;
```

Q-05 学内番号 / 資金元番号で機材を逆引き

- 対応：FR-01（備品・資産の番号管理）
- 入力： `university_id` または `funding_id`
- 出力：該当機材

```
SELECT
  e.equipment_id,
  e.name,
  e.status_code,
  ids.university_id,
  ids.funding_id
FROM equipment_identifiers ids
JOIN equipment e
  ON e.equipment_id = ids.equipment_id
WHERE (:university_id IS NOT NULL AND ids.university_id = :university_id)
   OR (:funding_id IS NOT NULL AND ids.funding_id = :funding_id);
```

Q-06 販売元（vendor）別の購入一覧（期間）

- 対応：購入情報の検索
- 入力： `vendor_id`, `from_date`, `to_date`
- 出力：購入一覧

```
SELECT
  pu.id AS purchase_id,
  pu.purchase_date,
  pu.price,
  pu.note,
  v.name AS vendor_name
FROM purchases pu
JOIN vendors v
  ON v.id = pu.vendor_id
WHERE pu.vendor_id = :vendor_id
  AND pu.purchase_date >= :from_date
  AND pu.purchase_date <  :to_date
ORDER BY pu.purchase_date DESC;
```

Q-07 機材1台の状態遷移タイムライン（監査ログ）

- 対応：NFR（監査/追跡）
- 入力： `equipment_id`

- 出力：時系列のイベント

```
SELECT
  ev.event_timestamp,
  ev.event_type,
  ev.from_status_code,
  ev.to_status_code,
  ev.from_user_id,
  ev.to_user_id,
  ev.from_manager_id,
  ev.to_manager_id,
  ev.loan_to_id,
  ev.return_to_id,
  ev.actor_id
FROM equipment_events ev
WHERE ev.equipment_id = :equipment_id
ORDER BY ev.event_timestamp;
```

Q-08 ダッシュボード用集計（状態別件数）

- 対応：運用可視化（任意）
- 入力（任意）： `project_id`
- 出力：状態別件数

```
SELECT
  e.status_code,
  COUNT(*) AS cnt
FROM equipment e
WHERE (:project_id IS NULL OR e.project_id = :project_id)
GROUP BY e.status_code
ORDER BY cnt DESC;
```

5. 提出用（自由検索）への落とし込み

- `sql/05_free_queries.sql` には、上記クエリのうち課題に該当するものを「具体値版」として記載する。

- `tests/05_search.sql` では、最低限、
  - プロジェクト別一覧（課題4）
  - 期間×状態（課題6）
  - 貸出中一覧（Q-04）
    を検証対象に含める。

6. 注意点（トリガ・終端状態との関係）

- `discarded / returned_to_funder` は終端状態のため、該当機材は以後 UPDATE できない。
  - 再現性のため、 `sql/04_transaction_cases.sql` は毎回 T-00 で新規作成し、そのIDを以後のTで使用する。
- 貸出先/返却先は events に保存されるため、
  - 「現在の貸出先」を求める場合は `equipment_events` の最新 loan を参照する（Q-04）。

6.1 トランザクションとROLLBACK（実運用 vs 手作業）

本プロジェクトのSQL例（例： `sql/04_transaction_cases.sql` ）では、学習目的のために `START TRANSACTION / COMMIT` を明示している。
一方で「途中で失敗したらどうなるか（ROLLBACKは必要か）」は、実運用と手作業で扱いが変わるため注意する。

A) 実運用（アプリ/API経由）の一般的な扱い

- 多くの場合、トランザクション境界（開始・コミット・ロールバック）はアプリ側が管理する。
  - 成功したら `COMMIT`
  - 例外やエラーが起きたら `ROLLBACK` （または接続終了により未コミット分を破棄）
- そのため、SQLを1本ずつ手で実行する運用ではなく「アプリの処理単位」で整合性を担保するのが一般的。

B) 手作業（MySQL CLIでの検証）の注意点

MySQLは「トランザクション中に1文が失敗したら全体を自動でROLLBACK」するとは限らない。
`START TRANSACTION;` の途中でエラーが起きても、

- 失敗した文は反映されない

- ただし **それ以前に成功した更新はトランザクション内に残る**
- セッションを継続するなら、意図しない“途中状態”を避けるために **自分で `ROLLBACK;` を実行する**

手作業での推奨パターン：

```
START TRANSACTION;
-- 途中のSQL...
-- エラーが出たら：
ROLLBACK;
-- 問題なければ：
COMMIT;
```

補足： `mysql < ファイル.sql` のようにバッチ実行していて、エラー後にプロセス/接続が終了した場合は、未COMMITの変更が破棄されて結果的にROLLBACKされたように見えることがある。
ただし「同じセッションで作業を続ける」場合は、明示的な `ROLLBACK` が安全。

7. 次に作成するSQLファイル

- `sql/05_free_queries.sql` ：提出用の自由検索クエリ（具体値版）
- `tests/05_search.sql` ：検索系テストの自動確認