📘 試験データ「完全再ロード」テスト手順（破壊あり / Phase3運用）

目的

Subject / Exam / Question / 採点系（SEV/SE/Adjust）を **一度クリアして、ゼロから再生成**できることを確認する。

Phase3（Subjectに fsyear/term を持ち、Examは subject+version）に合わせた運用。

前提

- 対象：subjectNo=<科目コード>, fsyear=2025（必要に応じて読み替え）
- JSON：/Volumes/NBPlan/TTC/examtools/work/answers_.json
- YAML：exam2/data/studentVersion.yaml
- **重要**：この手順は「破壊あり」です。対象 subjectNo/fsyear を必ず確認してから実行してください。

変数（読み替え用）

- <SUBJECT_NO> = <科目コード>
- = 2025

✅ 手順B：完全再ロード（破壊あり）

---

0) 事前安全確認（絶対やる）

0-1. 対象 Subject が正しいか確認

```
python manage.py shell -c "from exam2.models import Subject;
s=Subject.objects.get(subjectNo='<科目コード>',fsyear=2025);
print(s.id,s.subjectNo,s.fsyear,s.term,s.nenji,s.name)"
```

0-2. 現在の件数（消える量の見積）

```
python manage.py shell -c "
from exam2.models import
Subject,Exam,Question,StudentExam,StudentExamVersion,ExamAdjust
s=Subject.objects.get(subjectNo='<科目コード>',fsyear=2025)
ex=Exam.objects.filter(subject=s)
print('Exam=',ex.count())
print('Question=',Question.objects.filter(exam__in=ex).count())
print('SEV=',StudentExamVersion.objects.filter(exam__in=ex).count())
print('SE=',StudentExam.objects.filter(exam__in=ex).count())
print('Adjust=',ExamAdjust.objects.filter(exam__in=ex).count())
"
```

---

1) clear（破壊）

1-1. clear_subject_data 実行

```
python manage.py clear_subject_data <科目コード>
```

このプログラムでsubject以外のテーブルの関連する科目コードのデータをすべて削除する。

対象テーブル:

- `StudentExam`: 学生の問題ごとの解答データ
- `ExamAdjust`: 学生ごとの最終的な得点調整データ

絞り込み条件:

- 指定した **科目コード（`subjectNo`）** かつ **年度（`fsyear`）** に一致するもののみが対象となります。

項目	状態
試験問題 (`Question`)	そのまま残ります。
学生の解答行 (`StudentExam`)	行自体は残りますが、点数が 0 になります。
試験の枠組み (`Exam`)	そのまま残ります。
科目マスター (`Subject`)	そのまま残ります。

1-2. clear 後の確認（Phase3の期待値）

Phase3運用では **Subjectは残し、Exam以下が0** になっているのが正しいです。

```
python manage.py shell -c "
from exam2.models import
Subject,Exam,Question,StudentExam,StudentExamVersion,ExamAdjust
s=Subject.objects.get(subjectNo='<科目コード>',fsyear=2025)
ex=Exam.objects.filter(subject=s)
print('Subject=',1)
print('Exam=',ex.count())
print('Question=',Question.objects.filter(exam__in=ex).count())
print('SEV=',StudentExamVersion.objects.filter(exam__in=ex).count())
print('SE=',StudentExam.objects.filter(exam__in=ex).count())
print('Adjust=',ExamAdjust.objects.filter(exam__in=ex).count())
"
```

✅ 期待値：

- Subject=1
- Exam=0
- Question=0
- SEV=0
- SE=0
- Adjust=0

> 補足：あなたの実行結果で「Subjectが1件残る」は **正常**です（Phase3の設計どおり）。

---

2) load_subject_base（Subject/Exam作成）

2-1. 実行

```
python manage.py load_subject_base <科目コード>
```

2-2. 検証（Examバージョンが作られているか）

```
python manage.py shell -c "
from exam2.models import Subject,Exam
s=Subject.objects.get(subjectNo='<科目コード>',fsyear=2025)
print('Subject OK:',s.id,s.term,s.nenji,s.name)
print('Exam
versions:',list(Exam.objects.filter(subject=s).order_by('version').values_list('v
ersion',flat=True)))
"
```

✅ 期待：['A','B'] などが出る

---

3) load_questions（Question再作成）

3-1. 実行（破壊テストなので毎回同じ結果にするため clear 推奨）

```
python manage.py load_questions <科目コード> --fsyear 2025 --clear-existing --fix-
qno
```

3-2. 検証（version別 Question 件数）

```
python manage.py shell -c "
from exam2.models import Subject,Exam,Question
s=Subject.objects.get(subjectNo='<科目コード>',fsyear=2025)
for e in Exam.objects.filter(subject=s).order_by('version'):
    print(e.version, 'Q=', Question.objects.filter(exam=e).count())
"
```

✅ 期待：A/Bで同程度の件数が出る

---

4) load_student_exam_version（割当）

4-1. dry-run（必須）

```
python manage.py load_student_exam_version <科目コード> --fsyear 2025 --dry-run
```

4-2. 本実行（既存を消して作り直し）

```
python manage.py load_student_exam_version <科目コード> --fsyear 2025 --clear-existing
```

4-3. 検証（version別割当数）

```
python manage.py shell -c "
from exam2.models import Subject,Exam,StudentExamVersion
s=Subject.objects.get(subjectNo='<科目コード>',fsyear=2025)
for e in Exam.objects.filter(subject=s).order_by('version'):
    print(e.version,'SEV=',StudentExamVersion.objects.filter(exam=e).count())
"
```

---

5) load_student_exam（採点行の生成：TF=0,hosei=0）

5-1. dry-run（件数見積）

```
python manage.py load_student_exam <科目コード> --fsyear 2025 --dry-run
```

5-2. 本実行

```
python manage.py load_student_exam <科目コード> --fsyear 2025 --batch-size 2000
```

5-3. 検証（総件数）

```
python manage.py shell -c "
from exam2.models import Subject,Exam,StudentExam
s=Subject.objects.get(subjectNo='<科目コード>',fsyear=2025)
ex=Exam.objects.filter(subject=s)
print('StudentExam total=',StudentExam.objects.filter(exam__in=ex).count())
"
```

---

6) load_exam_adjust（補正行の生成：adjust=0）

6-1. dry-run

```
python manage.py load_exam_adjust <科目コード> --fsyear 2025 --dry-run
```

6-2. 本実行

```
python manage.py load_exam_adjust <科目コード> --fsyear 2025 --batch-size 2000
```

6-3. 検証（総件数）

```
python manage.py shell -c "
from exam2.models import Subject,Exam,ExamAdjust
s=Subject.objects.get(subjectNo='<科目コード>',fsyear=2025)
ex=Exam.objects.filter(subject=s)
print('ExamAdjust total=',ExamAdjust.objects.filter(exam__in=ex).count())
"
```

✅ 最終チェック（UI/API観点）

- index で subject を選ぶ → 学生一覧が出る
- 学生を開く（採点画面） → タイトルに **version(A/B)** が表示される
- 行一括 ON/OFF → /api/student-exams/bulk_update/ が **500にならない**
- cancel → 元に戻る（bulk_update でDBも戻る）

⚠️ 重要メモ（clear_subject_data の仕様固定）

Phase3運用では、**Subjectは残して Exam以下を消す**のが安全です。

あなたのコマンド出力も「Subject自体は削除しません。Exam以下を削除します。」になっているので、その方針で固定してください。

🧯 トラブルシュート

1)

YAML に fsyear=2025 のブロックがありません

原因：YAML 側のキーが 2025: であっても、読み込み結果のキー型（int/str）が想定と違う可能性があります。

あなたの提示では YAML は：

```
2025:
  1:
    A:
      - 25367001
```

→ load_student_exam_version.py 側で str(fsyear) と int 両方対応しているなら通常は通ります。

それでも出る場合は「YAMLファイルの読み込んでいるパスが違う」可能性が高いです（エラーメッセージのパスと実ファイルを照合）。

2) clearしたのにデータが残る

- subjectNo だけで消していて、fsyear違い の Subject が混在している可能性
  → 必ず subjectNo,fsyear で特定して確認する（0-1のコマンド）

必要なら、このドキュメントを **チェック欄付き（□）**にして「実施記録を書き込める版」にもできます。