Hand-in · Moving to Group Submissions

Andrew IDs:

You run **Hand-in**. Students submit through its **web UI** or **`handin` CLI**; staff grade submissions, and an **LMS grade sync** sends grades to Canvas. Each submission has one `student_id`.

- All three use Hand-in's HTTP API. Anyone with a course account can also call it using an API token.
- 12 instances share one database; deploys replace two at a time over about ten minutes.

Your task is to add support for **group submissions**, shared by several students. Existing courses, clients, and past submissions must keep working.

1 · Who depends on the old representation?

A Friday release replaces `student_id` with `members` throughout the code, API (including `POST /submissions` requests), and database. The IDE updated every caller, types and tests pass, and a database migration is included.

`GET /submissions/812`

Before

```
{ "id": 812, "student_id": "ana" }
```

After

```
{ "id": 812, "members": ["ana"] }
```

Who breaks on Monday? Add other potential dependencies in the blank rows.

Consumer	Does it break? How?	What do we need to find out?
Web UI		
`handin` CLI		
LMS grade sync		

2 · Design the API transition

You control Hand-in's **backend, web UI, and LMS sync**. Students choose when to upgrade installed CLIs. Add groups while existing individual submissions keep working.

The old API accepts and returns `student_id`, the submission's **sole owner**. Old CLIs require it but ignore extra response fields. The new API must support `members`.

1. Every member of a group can find the submission and receives its grade
2. A submission is counted and graded once
3. Courses that have not enabled groups behave exactly as before

In each cell, write short JSON for the response (reads) or the request body (create), or the error or restriction that applies. **Keep the meaning of `student_id`.**

Operation	Old clients (not upgraded)	Updated clients
Read an individual submission (response)	`{ "id": 812, "student_id": "ana" }`	
Read a group submission (response)		
Create a submission (request body)		

How does Hand-in tell an old client's request from an updated client's?

How do we roll out? What do we check before enabling groups or retiring the old API?

3a · What goes wrong with this plan?

Groups are first enabled in **one pilot course**; a group submission has no `student_id`. The database must move from `submissions.student_id` (a required column) to a `submission_members` table.

- Hand-in stores about **two million** past submissions, so a backfill takes hours.
- Students keep submitting throughout, including during deployments. Hand-in cannot go offline.
- Deployments replace two of the 12 server instances at a time over about ten minutes; all share one database.

A colleague proposes this plan, to be done before next semester. Under each step, mark whether it is safe; where it is not, explain the risk or the missing precondition.

1. Add the `submission_members` table.

2. Backfill membership rows for every past submission.

3. Deploy new code that reads and writes only `submission_members`.

4. Enable group submissions for the pilot course.

5. Drop the `student_id` column.

3b · Reference: steps for a safer plan

Keep submissions and your Activity 2 API contracts working after every step.

- **A.** Enable group submissions for the pilot course.
- **B.** Backfill one member per submission that has none, in small, resumable batches; preserve newer rows.
- **C.** Drop the `student_id` column.
- **D.** Add a `submission_members` table, and allow `student_id` to be empty.
- **E.** Switch authority and reads to membership rows. Writers derive `student_id` for one-member submissions and leave it empty for groups.
- **F.** Release a CLI version and an LMS sync update that understand groups.
- **G.** Deploy code that writes both `student_id` and membership rows in one transaction; `student_id` stays authoritative.
- **H.** Stop writing `student_id`, and tell the ORM to ignore the column.
- **I.** Verify exactly one member matching `student_id` per submission, and that reads from either copy agree.
- **J.** Retire the old API contract (`student_id` in requests and responses).

Our own steps (if we add or split any):

- **K.**
- **L.**
- **M.**

Write your plan and checkpoint answers on the next page.

Glossary

- **Source of truth:** the authoritative copy, which determines the correct value; the other copy is maintained from it.
- **Authority flag:** G deploys code that supports either copy as the source of truth; E switches between them without redeploying.

3b · Our plan and checkpoints

Write your order using the letters from the previous page, including any steps you added as K–M.

Our plan: | | | | | | | | | | | | | |

For each checkpoint, assume the migration pauses there.

After…	Which copy is the source of truth?	Keep submitting? Conditions?	Can we still go back? How?
Every instance writes both forms			
The membership table becomes authoritative, before any group exists			
The first group submission			
The `student_id` column is dropped			

Our riskiest step:

What must be true before we take it, and how would we check?

4 · Recover from a failed step

Recovery	What it means
Roll back	Return to the previous version of the code or flag setting.
Roll forward	Deploy a corrective change while continuing from the current state.
Re-derive	Rebuild a derived copy from the source of truth.
Restore	Bring back lost data from a backup.
Compensate	Correct effects that have already left the system, such as data sent to another service, and tell the people affected.

Incident card number:

What would we check, and what evidence would show our recovery is safe?

Contain: how do we stop it getting worse while we investigate?

Recover: roll back, roll forward, re-derive, restore, compensate? More than one may apply.

Why not the alternatives?