# Title: Bulk Import Experiment

## **Existing approach:**

We create each record serially from the CSV data.

Let's take a scenario of Custom Modules.

CSV has 10,000 rows to be imported. There are 2 custom fields associations for custom module. Thus, for every custom module entity to be created, there will be 3 inserts.

So for complete CSV import, there will be **30,000 insert queries**.

## **New Approach:**

We will use the gem -[ActiveRecord-import](https://github.com/zdennis/activerecord-import).

Gem requires an array of active records as input. It will create and execute a single insert query for all records in the array.

As compared to the existing scenario, we can complete the task with **3 insert queries.**

**Why not used till now in the product?**

Gem has some major limitations:

1. The only database operation involved in the gem is to execute the insert query.
2. Insert happens only for the provided model i.e. Associations of the model are not imported.
3. After insert, it won't fetch and return the records.
4. Since it does not fetch the records that are inserted, it does not support running the callbacks.

Since models and entire implementation needs to be tweaked, this approach was dropped and we went on with the traditional way of import.

## **Implementation:**

Recently Custom Modules was introduced in Freshsales. Being a new entity with minimal implementation, we had the opportunity to test the gem again.

**Assumptions and Decisions that are taken keeping performance in mind:**

1. We will use gem to create an insert query of 100 active records at a time.
2. The model class will not include any around callbacks. Since we will be triggering all callbacks manually, it will be hard to handle around callbacks.
3. Association decisions:
   1. Associations will be created up to one level. i.e. Association of Associations will not be handled.
   2. As of now, there is no requirement to run callbacks on the association. We will handle it when required.
   3. We are whitelisting the associations that need to be imported.
   4. In case if any belongs-to association is present and need to be imported, we should consider it as parent entity and import it first. As of now, this is not yet implemented.
   5. There is no delete operation performed. Even for associations, there should be only update-case. There should not be any recreate-case eg. contact\_sales\_accounts associations for contacts. If required, we will have to bulk delete. But as of now, no such use-case is present.

**The flow of execution:**

* From CSV, We will select a chunk of 100 rows.
* We will build active record objects and validate. We will skip invalid records at this stage.
* Execution inside a transaction block:
  + Run before callbacks on the parent entity.
  + Create and execute bulk insert query for 100 active records.
  + Fetch imported records.
  + Create a list of association per model for eg. an array of CustomField1 records and execute Bulk Insert for each model.
  + Set previous\_changes data from the stored changes to the fetched records. We will manually set it.
  + Run after callbacks
* Run after\_commit callback on each record. This will run out of the transaction block.

## **Challenges:**

Since the gem does not support anything other than Insert Query generation and Execution as per use case, we had to mock rails way manually.

1. We have to store the changes of each active record and have to assign it back manually to a rails instance variable to give back previous\_changes data.
2. We have to set the transaction state as create/update manually for each record so that implementation is not altered in the model.
3. We will have to trigger callbacks manually.
   1. The way rails work is, we can trigger callbacks in 2 ways:
      1. trigger before callbacks
      2. trigger before and after callbacks together
   2. Since there is no way to trigger after\_callbakcs alone, we have to add a check in before callbacks methods to skip it running the second time.
4. Rails implementation is very big and abstracted at each level. As per current usecase we were able to mock rails way of handling features like callbacks etc. We will have to put extra efforts if any new use cases are found.

## **Results:**

For a CSV of 10,000 rows for custom module:

|  |  |  |
| --- | --- | --- |
|  | Old Approach | New Approach |
| **No. of insert queries** | * 30,000 queries   + 1000 for custom\_module1 table   + 1000 for custom\_field\_table1   + 1000 for custom\_field\_table2 | * 300 queries(since the batch size is 100)   + 100 for custom\_module1 table   + 100 for custom\_field\_table1   + 100 for custom\_field\_table2 * Skipped extra **29,700 queries.** |
| **Time Taken** **(Local Machine)** | Calculated in rails console: 396 sec + (variable time during import flow) It might reach to 410 to 420 seconds easily. | 235 sec for import flow. **Import is close to 2X faster.**  As the number of associations increases, this will become much faster as compared to the old approach. |
| **Time Taken** **(Vijay Rails Stack)** |  | 199 sec for Import flow.  **Per second - 50 records are created.** |
| **Exception handling** | * Identify run time exceptions for each record and skip it alone * Resuming import can happen from the exact point it stopped. | * We will have to skip a batch of 100 fully if any runtime exception occurs. * Resuming import will happen batch-wise only. i.e. even if 99 records were inserted properly since batch was not completed, we will rollback and on resuming again 99 records will be processed. |