code snippet of the importSeasons method (This method uploads dummy data for the 2013 Season year including one Race, one Driver and one Driver's result as a Contestant in a Race) is provided which doesn't use a Unit of Work pattern (as a pattern example) and which consists of 54 lines of code in total. Notice that four insert operations are used in this piece of code and also Database. The setSavePoint method is used which is also counted in the total number of database statements. These lines of code are highlighted in bold, and the highlight is mine.

```
public static void importSeasons(String jsonData) {
 System.Savepoint serviceSavePoint = Database.setSavePoint();
 try{
  // Parse JSON data
  SeasonService.SeasonsData seasonsData = (SeasonService.SeasonsData) JSON.deserializeStrict(jsonData, SeasonService.SeasonsData.class);
 // Insert Drivers
 Map<String, Driver__c> driversById = new Map<String, Driver__c>();
 for(SeasonService.DriverData driverData : seasonsData.drivers)
  driversById.put(driverData.driverId, new Driver__c( Name = driverData.name,
    DriverId__c = driverData.driverId,
    Nationality__c = driverData.nationality,
    TwitterHandle__c = driverData.twitterHandle)
  );
  insert driversById.values();
  // Insert Seasons
 Map<String, Season__c> seasonsByYear = new Map<String, Season__c>();
 for(SeasonService.SeasonData seasonData : seasonsData.seasons)
  seasonsByYear.put(seasonData.year, new Season__c(
  Name = seasonData.year, Year__c = seasonData.year));
 insert seasonsByYear.values();
 // Insert Races
Map<String, Race__c>racesByYearAndRound = new Map<String, Race__c>();
for(SeasonService.SeasonData seasonData : seasonsData.seasons)
for(SeasonService.RaceData raceData : seasonData.races)
racesByYearAndRound.put(seasonData.Year + raceData.round,
new Race__c(
Season__c = seasonsByYear.get(seasonData.year).Id, Name = raceData.name));
insert racesByYearAndRound.values();
// Insert Contestants
List<Contestant__c> contestants = new List<Contestant__c>();
for(SeasonService.SeasonData seasonData : seasonsData.seasons)
    for(SeasonService.RaceData raceData : seasonData.races)
for(SeasonService.ContestantData contestantData: raceData.contestants)
    contestants.add(
new Contestant__c(
Race__c = racesByYearAndRound.get(seasonData.Year + raceData.round).Id,
Driver__c = driversById.get( contestantData.driverId).Id,
ChampionshipPoints__c = contestantData.championshipPoints,
DNF__c = contestantData.dnf, Qualification1LapTime__c =
contestantData.qualification1LapTime, Qualification2LapTime__c =
contestantData.qualification2LapTime, Qualification3LapTime__c =
contestantData.qualification3LapTime
));
insert contestants;
} catch (Exception e) {
// Rollback any data written before the exception
Database.rollback(serviceSavePoint); // Pass the exception on
throw e;
}
```

The version of he importSeasons methodt, which uses a Unit of Work pattern, contains 48 lines of code.

```
public static void importSeasons(String jsonData) {
 System.Savepoint serviceSavePoint = Database.setSavePoint();
 try{
  // Parse JSON data
  SeasonService.SeasonsData seasonsData = (SeasonService.SeasonsData) JSON.deserializeStrict(jsonData, SeasonService.SeasonsData.class);
 // Insert Drivers
 Map<String, Driver__c> driversById = new Map<String, Driver__c>();
 for(SeasonService.DriverData driverData : seasonsData.drivers)
  driversById.put(driverData.driverId, new Driver__c( Name = driverData.name,
    DriverId__c = driverData.driverId,
    Nationality__c = driverData.nationality,
    TwitterHandle__c = driverData.twitterHandle)
  );
  insert driversById.values();
  // Insert Seasons
 Map<String, Season__c> seasonsByYear = new Map<String, Season__c>();
 for(SeasonService.SeasonData seasonData : seasonsData.seasons)
  seasonsByYear.put(seasonData.year, new Season__c(
  Name = seasonData.year, Year__c = seasonData.year));
 insert seasonsByYear.values();
 // Insert Races
Map<String, Race__c>racesByYearAndRound = new Map<String, Race__c>();
for(SeasonService.SeasonData seasonData : seasonsData.seasons)
for(SeasonService.RaceData raceData : seasonData.races)
```

```
racesByYearAndRound.put(seasonData.Year + raceData.round,
new Race__c(
Season__c = seasonsByYear.get(seasonData.year).Id, Name = raceData.name));
insert racesByYearAndRound.values();
// Insert Contestants
List<Contestant__c> contestants = new List<Contestant__c>();
for(SeasonService.SeasonData seasonData : seasonsData.seasons)
    for(SeasonService.RaceData raceData : seasonData.races)
for(SeasonService.ContestantData contestantData: raceData.contestants)
    contestants.add(
new Contestant__c(
Race__c = racesByYearAndRound.get(seasonData.Year + raceData.round).Id,
Driver__c = driversById.get( contestantData.driverId).Id,
ChampionshipPoints__c = contestantData.championshipPoints,
DNF__c = contestantData.dnf, Qualification1LapTime__c =
contestantData.qualification1LapTime, Qualification2LapTime__c =
contestantData.qualification2LapTime, Qualification3LapTime__c =
contestantData.qualification3LapTime
));
insert contestants;
} catch (Exception e) {
// Rollback any data written before the exception
Database.rollback(serviceSavePoint); // Pass the exception on
throw e;
}
```

Let’s consider a default Unit of Work implementation to see if it is still used underneath the same four insert operations and Database. The setSavePoint method, which still yields a total of five database statements.

```
public class SimpleDML implements IDML{
        public void dmlInsert(List<SObject> objList){
            insert objList;
        }
        ….
}

public void commitWork() {
        // notify we're starting the commit work
        onCommitWorkStarting();

        // Wrap the work in its own transaction
        Savepoint sp = Database.setSavePoint();
        Boolean wasSuccessful = false;
        try
        {
            // notify we're starting the DML operations
            onDMLStarting();
            // Insert by type
            for(Schema.SObjectType sObjectType : m_sObjectTypes)
            {
                m_relationships.get(sObjectType.getDescribe().getName()).resolve();
                m_dml.dmlInsert(m_newListByType.get(sObjectType.getDescribe().getName()));
            }
            ….      }
        catch (Exception e)
        {
            // Rollback
            Database.rollback(sp);
            // Throw exception on to caller
            throw e;
        }
        finally
        {
            // notify we're done with commit work
            onCommitWorkFinished(wasSuccessful);
        }
    }
```

I agree with the author, in that it might be beneficial in some cases to follow a Unit of Work pattern and to handle database transactions in a governed way, but sometimes if we look at the task through a different angle, we can drastically simplify implementation. For the example provided in the book, I would recommend that slight modifications be made to the database model and to simplify the code drastically.

First of all to test this solution we need to download the code and customization provided by example code downloads.

After we download the code sample, we need to deploy it to some organization.

The classic way is to use ANT but now we can use SFDX instead and execute some commands like this:

```
sfdx force:mdapi:deploy -u ffcpt5 -d ../src
```

to deploy source code to some organization authenticated in or created previously.

Let’s take the JSON example data provided in the book and upload it to some Static Resource and let’s call it “json”.

```
{"drivers":[
  { "name": "Lewis Hamilton",
    "nationality": "British",
    "driverId": "44",
    "twitterHandle": "lewistwitter"
  }],
  "seasons": [
  { "year": "2013",
    "races": [
     {
       "round": 1,
       "name": "Spain",
       "contestants": [
       {
         "driverId": "44",
         "championshipPoints": 44,
         "dnf": false,
         "qualification1LapTime": 123,
         "qualification2LapTime": 124,
         "qualification3LapTime": 125
       }]
     }]
  }]
}
```

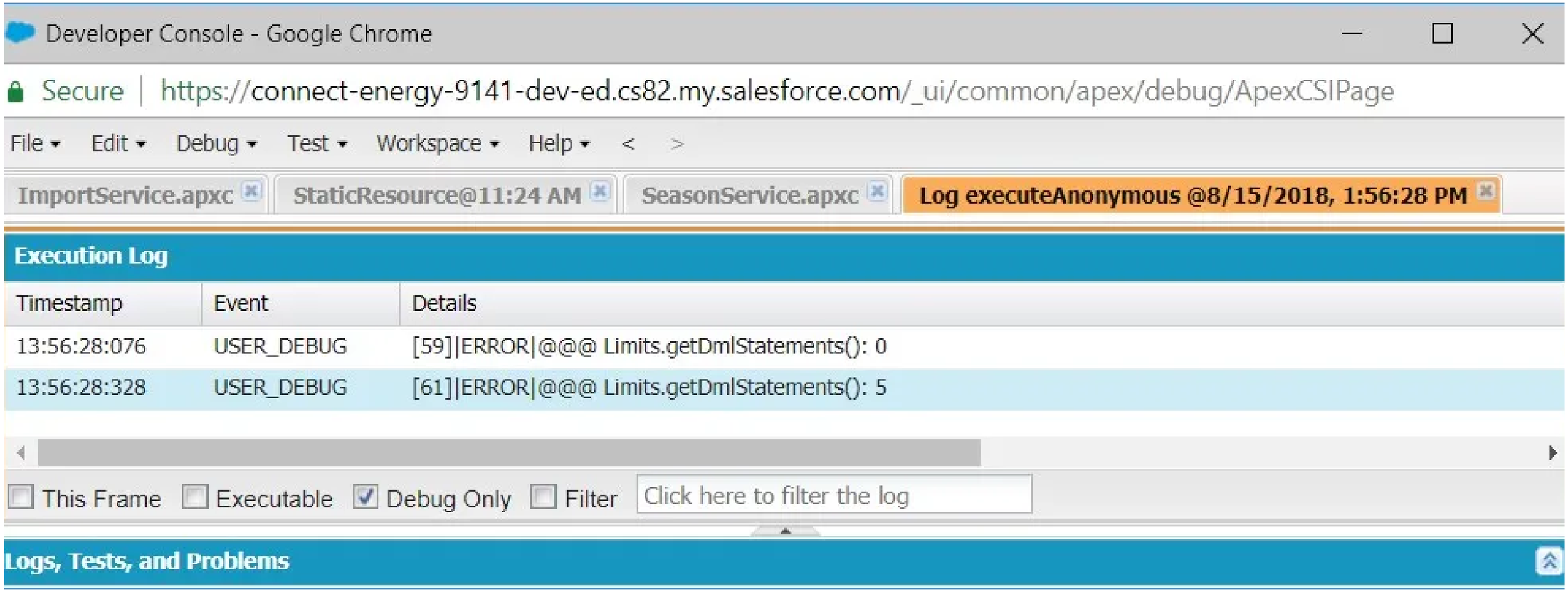
Also let’s create a Apex Class ImportService and copy the previous version of the importSeasons method into it.

Now let’s execute the following snippet of code to determine the number of DML statements used.

```
StaticResource x = [ SELECT Name, Body FROM StaticResource WHERE Name = 'json'];
        System.debug(LoggingLevel.ERROR, '@@@ Limits.getDmlStatements(): ' + Limits.getDmlStatements() );
        ImportService.importSeasons(x.Body.toString());
        System.debug(LoggingLevel.ERROR, '@@@ Limits.getDmlStatements(): ' + Limits.getDmlStatements() );
Reading the debug logs gives us the number of DML statements used, it is 5.
@@@ Limits.getDmlStatements(): 5
```

Reading the debug logs gives us the number of DML statements used, it is 5.

```
@@@ Limits.getDmlStatements(): 5
```



→ Get 100% Code Coverage for Salesforce Custom Metadata Based Decisions

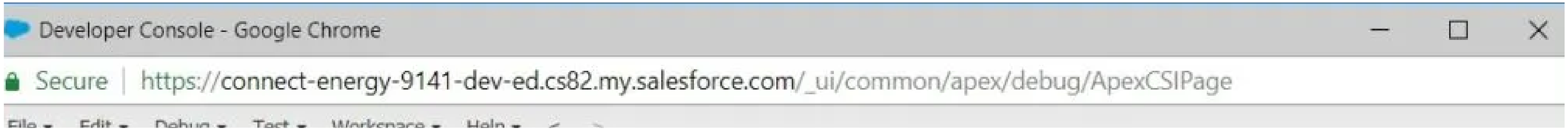
Now we need to delete inserted data and after that let's try another version.

After cleaning up, let's execute this code:

```
    StaticResource x = [ SELECT Name, Body FROM StaticResource WHERE Name = 'json'];
           SeasonService.importSeasons(x.Body.toString());
           System.debug(LoggingLevel.ERROR, '@@@ Limits.getDmlStatements(): ' + Limits.getDmlStatements() );
```

Again, the amount of DML statements is 5.

```
@@@ Limits.getDmlStatements(): 5
```



File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

cuteAnonymous @8/15/2018, 1:56:28 PM | Log executeAnonymous @8/15/2018, 1:58:50 PM | **Log executeAnonymous @8/15/2018, 1:59:13 PM**

Execution Log

Timestamp	Event	Details
13:59:13:000	USER_DEBUG	"round": 1,
13:59:13:000	USER_DEBUG	"name": "Spain",
13:59:13:000	USER_DEBUG	"contestants": [
13:59:13:000	USER_DEBUG	{
13:59:13:000	USER_DEBUG	"driverId": "44",
13:59:13:000	USER_DEBUG	"championshipPoints": 44,
13:59:13:000	USER_DEBUG	"dnf": false,
13:59:13:000	USER_DEBUG	"qualification1LapTime": 123,
13:59:13:000	USER_DEBUG	"qualification2LapTime": 124,
13:59:13:000	USER_DEBUG	"qualification3LapTime": 125
13:59:13:000	USER_DEBUG	}]
13:59:13:000	USER_DEBUG	}]
13:59:13:000	USER_DEBUG	}
13:59:13:000	USER_DEBUG	
13:59:13:000	USER_DEBUG	
13:59:13:350	USER_DEBUG	[68]|ERROR|@@@ Limits.getDmlStatements(): 5

☐ This Frame ☐ Executable ☑ Debug Only ☐ Filter Click here to filter the log

Logs, Tests, and Problems

The amount of DML statements is one of the Governor Limits. In Salesforce, Apex execution is allowed to execute 150 DML Statements in a single transaction.

If we want to see the number of governor limits used in a transaction we don’t have to add System.debug statements, we can just scroll down the debug log to the end and find the section about limit usage.

Developer Console - Google Chrome

Secure | https://connect-energy-9141-dev-ed.cs82.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

cuteAnonymous @8/15/2018, 1:56:28 PM | Log executeAnonymous @8/15/2018, 1:58:50 PM | **Log executeAnonymous @8/15/2018, 1:59:13 PM**

Execution Log

Timestamp	Event	Details
13:59:13:350	USER_DEBUG	[68]|ERROR|@@@ Limits.getDmlStatements(): 5
13:59:13:350	METHOD_EXIT	[13]|01p3E000000yKs0|ImportService.checkSeasonService()
13:59:13:350	SYSTEM_MODE...	false
13:59:13:403	CUMULATIVE_L...	
13:59:13:403	LIMIT_USAGE_...	(default)|
13:59:13:000	LIMIT_USAGE_...	Number of SOQL queries: 1 out of 100
13:59:13:000	LIMIT_USAGE_...	Number of query rows: 1 out of 50000
13:59:13:000	LIMIT_USAGE_...	Number of SOSL queries: 0 out of 20
13:59:13:000	LIMIT_USAGE_...	Number of DML statements: 5 out of 150
13:59:13:000	LIMIT_USAGE_...	Number of DML rows: 5 out of 10000
13:59:13:000	LIMIT_USAGE_...	Maximum CPU time: 0 out of 10000
13:59:13:000	LIMIT_USAGE_...	Maximum heap size: 0 out of 6000000
13:59:13:000	LIMIT_USAGE_...	Number of callouts: 0 out of 100
13:59:13:000	LIMIT_USAGE_...	Number of Email Invocations: 0 out of 10
13:59:13:000	LIMIT_USAGE_...	Number of future calls: 0 out of 50
13:59:13:000	LIMIT_USAGE_...	Number of queueable jobs added to the queue: 0 out of 50
13:59:13:000	LIMIT_USAGE_...	Number of Mobile Apex push calls: 0 out of 10

☐ This Frame ☐ Executable ☐ Debug Only ☐ Filter Click here to filter the log

Logs, Tests, and Problems

```
LIMIT_USAGE_FOR_NS Number of DML statements: 5 out of 150
```

In the highlighted line it is shown that the number of DML statements used is 5 and the limit is 150.

In the snippet code, we perform a set of database save points and four insert operations of four different objects. Assume that we have 150 different objects to insert, in such a case this approach would exceed governor limit.

Is there a way to refactor this into a more efficient way to achieve desired business logic?

Actually the book hints on the solution, mentioning that external field references might help to simplify the manual data import process, but this topic is not expanded on further.

In fact, external field references might help to simplify the code as well. In the provided data model, two of three parent objects already have an external id unique field. Let’s introduce another one on the Race object. Let’s call it Unique Key. It will be a Text field with External Id and Unique attributes.

Race Custom Field

Unique Key

Back to Race

Help for this Page

Validation Rules [0]

Custom Field Definition Detail

Edit | Set Field-Level Security | View Field Accessibility

Field Information

Field Label	Unique Key	**Object Name**	Race
Field Name	Unique_Key	**Data Type**	Text
API Name	Unique_Key__c		
Description			
Help Text			
Created By	User User, 8/14/2018 8:28 AM	**Modified By**	User User, 8/14/2018 8:28 AM

General Options

Required	☐
Unique	✓
Case Sensitive	☐
External ID	✓
Default Value	

Text Options

Length	6

→ Logging exceptions in Salesforce

Let’s reformat the JSON data so that it can be automatically consumed by Salesforce Apex without the need of redundant Data Transfer Objects. The new JSON Data will look like following:

```
[
  {
    "attributes": {
      "type": "Driver__c"
    },
    "Name": "Lewis Hamilton",
    "DriverId__c": "44",
    "Nationality__c": "British",
    "TwitterHandle__c": "lewistwitter"
  },
  {
    "attributes": {
      "type": "Season__c"
    },
    "Name": "2013",
    "Year__c": "2013"
  },
  {
    "attributes": {
      "type": "Race__c"
    },
    "Name": "Spain",
    "Unique_Key__c": "2013-1",
    "Season__r": {
      "Year__c": "2013"
    }
  },
  {
    "attributes": {
      "type": "Contestant__c"
    },
    "Race__r": {
      "Unique_Key__c": "2013-1"
    },
    "Driver__r": {
      "TwitterHandle__c": "lewistwitter"
    },
    "RacePosition__c": null,
    "ChampionshipPoints__c": 44,
    "DNF__c": false,
    "Qualification1LapTime__c": 123,
    "Qualification2LapTime__c": 124,
    "Qualification3LapTime__c": 125
  }
]
```

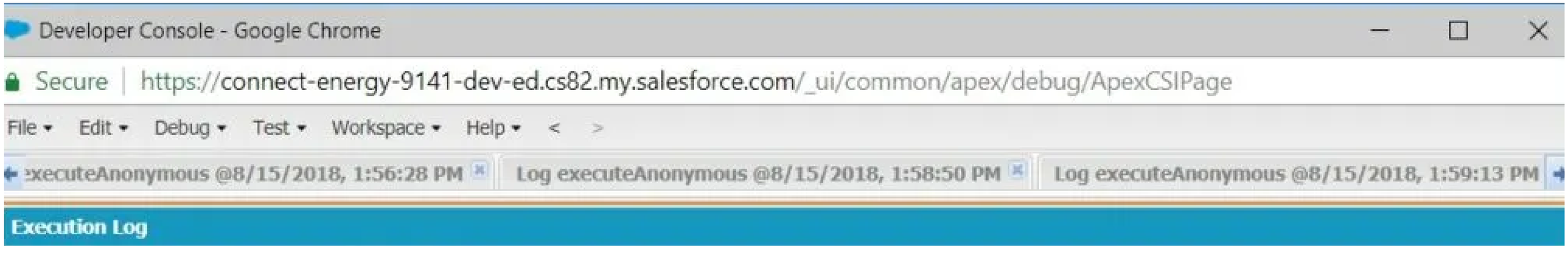

Let’s save it for convenience in another Static Resource and call it “optimizedJSON”.

Now let’s execute the following code:

```
StaticResource x = [ SELECT Name, Body FROM StaticResource WHERE Name = 'optimizedJSON'];
List<SObject> records = (List<SObject>) JSON.deserialize(x.Body.toString(), List<SObject>
```

We can see that in this approach, only one DML statement is used.

```
@@ Limits.getDmlStatements(): 1
```



Timestamp	Event	Details
14:25:21:308	USER_DEBUG	[76]|ERROR|@@@ Limits.getDmlStatements(): 1
14:25:21:308	METHOD_EXIT	[14]|01p3E000000yKs0|ImportService.checkOptimized()
14:25:21:308	SYSTEM_MODE...	false
14:25:21:347	CUMULATIVE_L...	
14:25:21:347	LIMIT_USAGE_...	(default)|
14:25:21:000	LIMIT_USAGE_...	Number of SOQL queries: 1 out of 100
14:25:21:000	LIMIT_USAGE_...	Number of query rows: 1 out of 50000
14:25:21:000	LIMIT_USAGE_...	Number of SOSL queries: 0 out of 20
14:25:21:000	LIMIT_USAGE_...	Number of DML statements: 1 out of 150
14:25:21:000	LIMIT_USAGE_...	Number of DML rows: 4 out of 10000
14:25:21:000	LIMIT_USAGE_...	Maximum CPU time: 0 out of 10000
14:25:21:000	LIMIT_USAGE_...	Maximum heap size: 0 out of 6000000
14:25:21:000	LIMIT_USAGE_...	Number of callouts: 0 out of 100
14:25:21:000	LIMIT_USAGE_...	Number of Email Invocations: 0 out of 10
14:25:21:000	LIMIT_USAGE_...	Number of future calls: 0 out of 50
14:25:21:000	LIMIT_USAGE_...	Number of queueable jobs added to the queue: 0 out of 50
14:25:21:000	LIMIT_USAGE_...	Number of Mobile Apex push calls: 0 out of 10

This Frame Executable Debug Only Filter Click here to filter the log

Logs, Tests, and Problems

→ Migrate from Aura to Lightning Web Components to Increase Performance

While the same business requirements are met and implemented, the same amount of data is imported, including Season, Race, Driver and Contestant record.

→ Salesforce Marketing Cloud March 2020 Release Notes

Also, the code is more concise and simpler to understand and maintain. We don’t have here transaction savepoint management since we have only one DML statement. The optimized version of code has two lines of code compared to 54 lines in the without unit of work pattern implemented, or compared to 48 lines in the code snippet having a unit of work pattern implemented including 350 lines of code in flib_Application class, 402 lines of code in fflib_SObjectUnitOfWork class and 54 lines of code in Application class.

We can summarize our results in the following table:

Efficiency Matrix	Without Unit Of Work	With Unit of Work	Optimized
DML Statements Usage	5	5	1
Apex Code Lines	54	854+	2
JSON Data File Lines	31	31	45

So we have shown here that it is possible to simplify and reduce the code and save total number of issued DML statements in the cost of complexification of auxiliary structures like an additional field in the data model and more complex JSON data format.