Tracking Failed Records in Batch Apex

NISHANT PATIL in

When processing large data sets in Salesforce using Batch Apex, it's essential to track which records were processed successfully and which ones failed. The **Database.SaveResult** and **Database.BatchableContext** interfaces help you do this by providing feedback on record operations during the execute and finish methods of a batch job.

NISHANT PATIL

By using the **Database.Stateful** interface, we can retain state between the different batch execution steps, making it possible to store information about failed records.

Why use Database.Stateful ?

Normally, Apex batch jobs are stateless, meaning they don't retain variable values between executions. By implementing the Database.Stateful interface, you can maintain state, such as tracking failed records across batch executions.

NISHANT PATIL in

Approach:

- **Use Partial Success in DML Operations:**
    - Using Database.insert, Database.update, or Database.delete with allOrNone=false ensures that the successful records are processed while allowing you to handle the failed ones separately.
- **Capture Failed Record IDs**:
    - You can loop through the results of the DML operation to capture and log any records that fail.

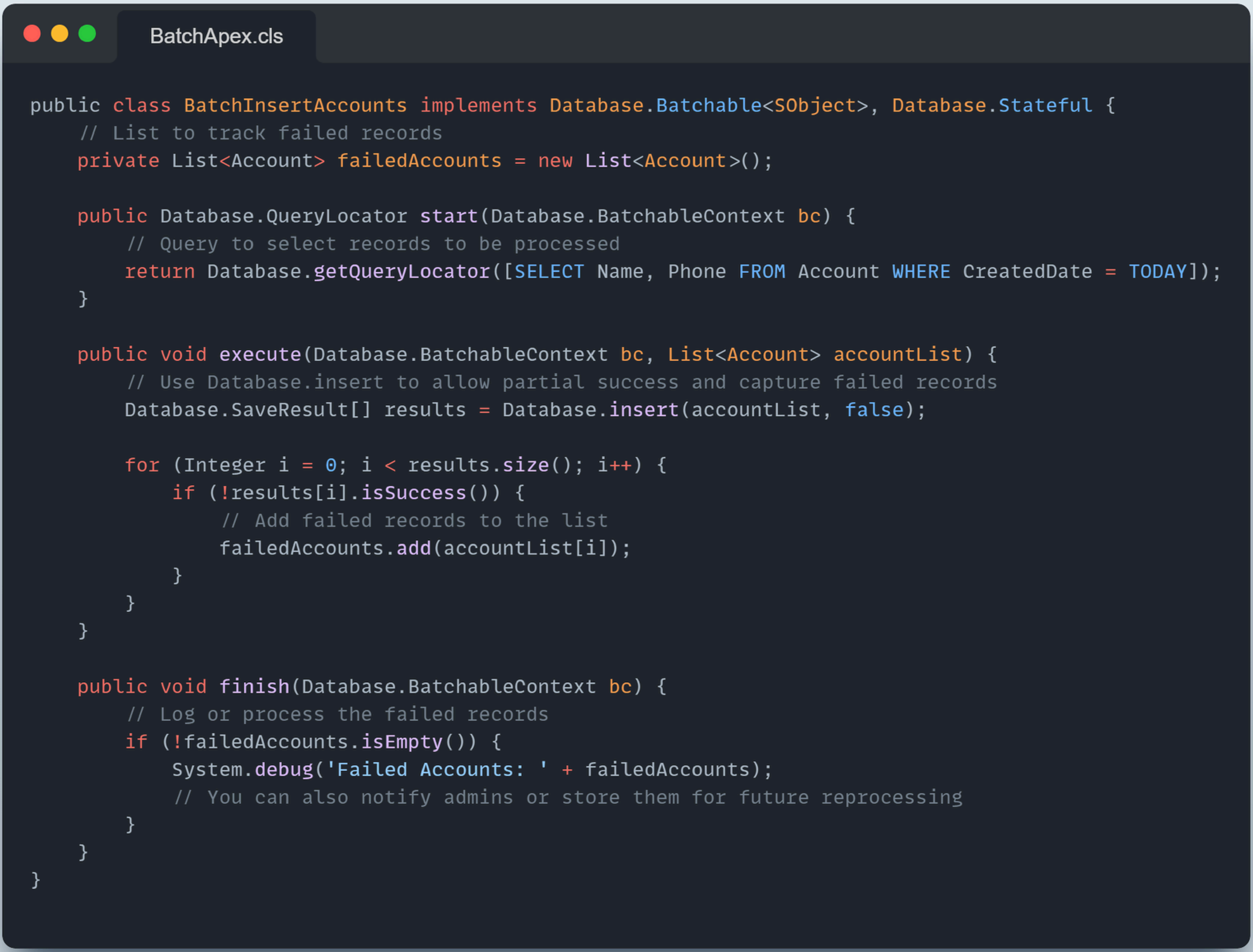
NISHANT PATIL

Approach:

- **Log the Failed Records**:
    - The failed records' IDs and error messages can be logged for further investigation. You can store these in a custom object or simply log them using System.debug().

NISHANT PATIL in

EXAMPLE



BatchApex.cls

```
public class BatchInsertAccounts implements Database.Batchable<SObject>, Database.Stateful {
    // List to track failed records
    private List<Account> failedAccounts = new List<Account>();

    public Database.QueryLocator start(Database.BatchableContext bc) {
        // Query to select records to be processed
        return Database.getQueryLocator([SELECT Name, Phone FROM Account WHERE CreatedDate = TODAY]);
    }

    public void execute(Database.BatchableContext bc, List<Account> accountList) {
        // Use Database.insert to allow partial success and capture failed records
        Database.SaveResult[] results = Database.insert(accountList, false);

        for (Integer i = 0; i < results.size(); i++) {
            if (!results[i].isSuccess()) {
                // Add failed records to the list
                failedAccounts.add(accountList[i]);
            }
        }
    }

    public void finish(Database.BatchableContext bc) {
        // Log or process the failed records
        if (!failedAccounts.isEmpty()) {
            System.debug('Failed Accounts: ' + failedAccounts);
            // You can also notify admins or store them for future reprocessing
        }
    }
}
```

NISHANT PATIL

EXPLANATION

- **Database.Stateful**: Ensures that the failedAccounts list retains its values between each batch execution.
- **Database.insert**(records, false): Inserts records while allowing partial successes. Records that fail don’t stop the entire operation.

NISHANT PATIL

EXPLANATION

- **SaveResult[]**: The result of the insert operation, where you can check each record’s success or failure.
- **finish method**: At the end of the batch, the failed records are logged and can be handled accordingly (e.g., reprocessed or reported).

KEY POINTS

- Use **Database.Stateful** to track the state of variables across batch executions.
- Use **Database** methods like insert or update with false to allow partial success.
- Handle failed records in the finish method for further action or notification.

NISHANT PATIL

LIKE
SHARE
IT

FOLLOW FOR MORE SUCH CONTENTS

NISHANT PATIL in