Lab: Unit 11 – Transactions

# Overview

In this lab we will explore how to write and test transaction safe code.

## Learning Objectives

Upon completion of the lab, you should be able to:

* Write your own data logic in a transaction.
* Test your code for transaction safety.
* Write instead-of triggers.

## What you will need

To complete this lab, you will need the Learn Databases Environment up and running, specifically:

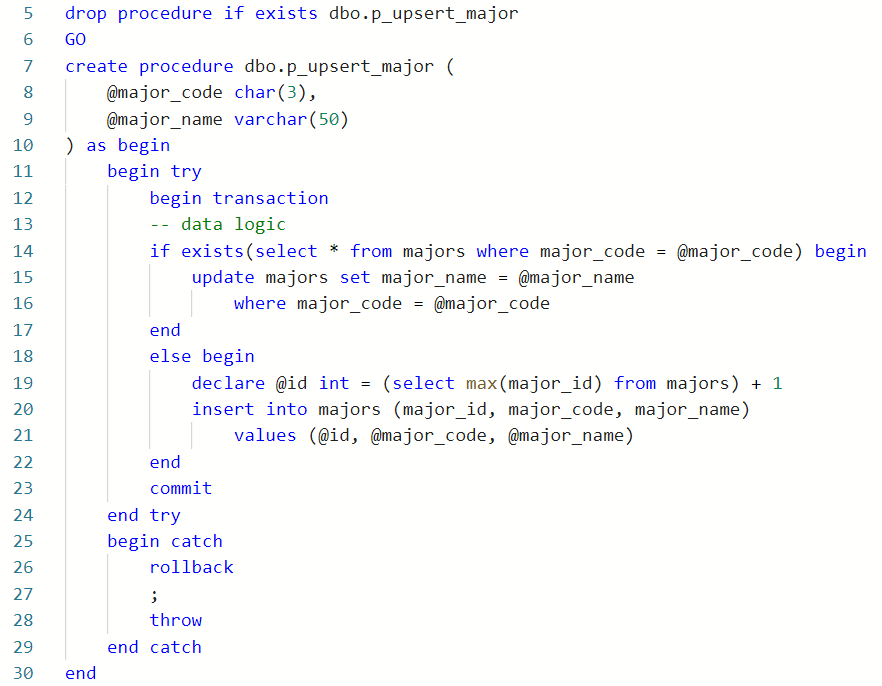
* Microsoft SQL Server DBMS,
* Provision the **TinyU, vBay** and **Demo** databases using the Database Provisioner application <https://localhost:5000>
* Azure Data Studio connected to SQL server with an open query window.
* Please review the first lab if you require assistance with these tools.

# Walkthrough

In a previous problem set, we created a stored procedure **p\_upsert\_major** which, would add a major if major\_code did not exist. When the major\_code exists, it would update. Let’s rewrite this procedure to be transaction safe.

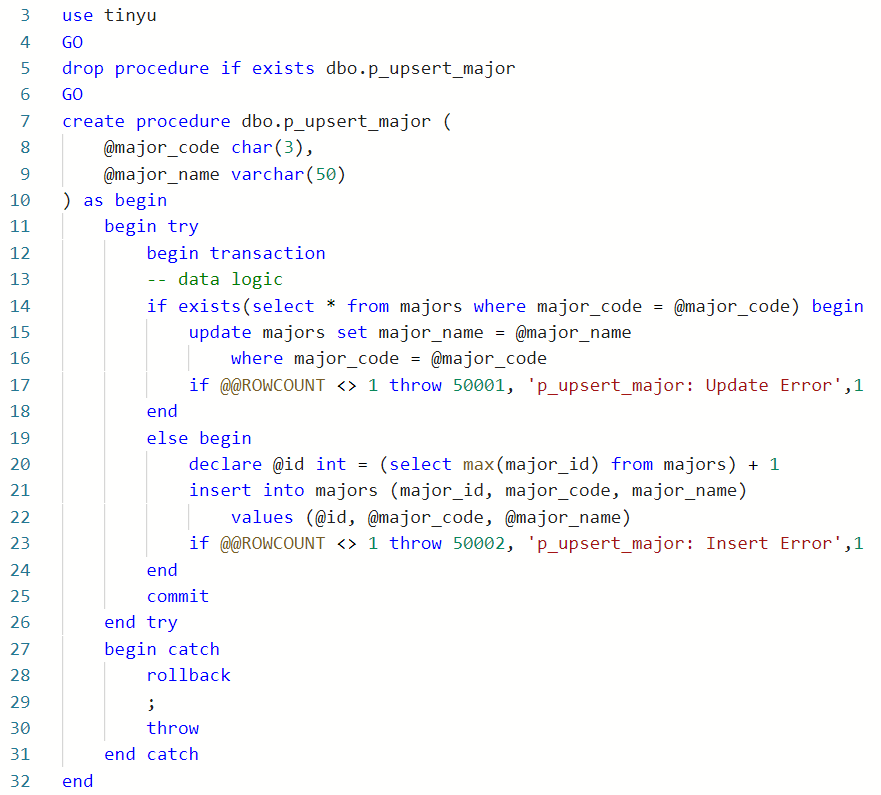
To be transaction-safe it must handle errors and exceptions to the data logic.

**To handle errors**, we introduce TRY/CATCH:



The original data logic are lines 14 through 22. This is what should be surrounded by the transaction and the try/catch.

**To handle custom data logic,** we must consider the expected output of the procedure. How many rows should it affect upon success? Are there required values? In this case we always expect 1 row to be affected by the upsert operation (either inserted or updated):



Lines 17 and 23 test the update and insert respectively to check whether the proper number of rows was affected, 1 in this case.

# Questions

Answer these questions using the problem set submission template. You will need to consult the logical model in the overview section for details. For any screenshots provided, please follow the guidelines for submitting a screenshot.

Write the following as SQL programs. For each, include the SQL as a screenshot with the output of the SQL Code.

1. Provide a screenshot of your code execution from the walkthrough were you modified **p\_upsert\_major** in the **TinyU** database to be transaction-safe.
2. Provide a screenshot of examples of executing the **p\_upsert\_major** procedure to demonstrate it is transaction safe.
3. Re-write the **p\_place\_bid** stored procedure from the **vBay** database so that it is transaction safe. Provide a screenshot of the code and its execution.
4. Execute your stored procedure in step 3 to demonstrate the procedure works. Make user 2, Bid $105 on item 36 and show the bid was placed with a SELECT.
5. Re-write the **p\_rate\_user** stored procedure from the **VBay** database so that it is transaction safe. Provide a screenshot of the code and its execution.
6. Execute the stored procedure in step 5 to demonstrate the rollback works. You should give a 6 star rating and then execute again where someone attempts to rate themselves. Produce as screen shot as evidence the rollback worked.
7. There is a conceptual data requirement which says that no **TinyU** major can have more than 15 students in it. (I know, this seems silly but think of the bigger problem – how to we enforce a specific minimum or maximum cardinality instead of just 1 or “many”?) Write data logic using an instead of trigger to do this.
8. Test step 7 by trying to add or update a student and change their major to ADS. The ADS major has 15 students already. Your code should drop/create the trigger and also test the success and failure of the trigger.