Homework Problem Set 8: 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:

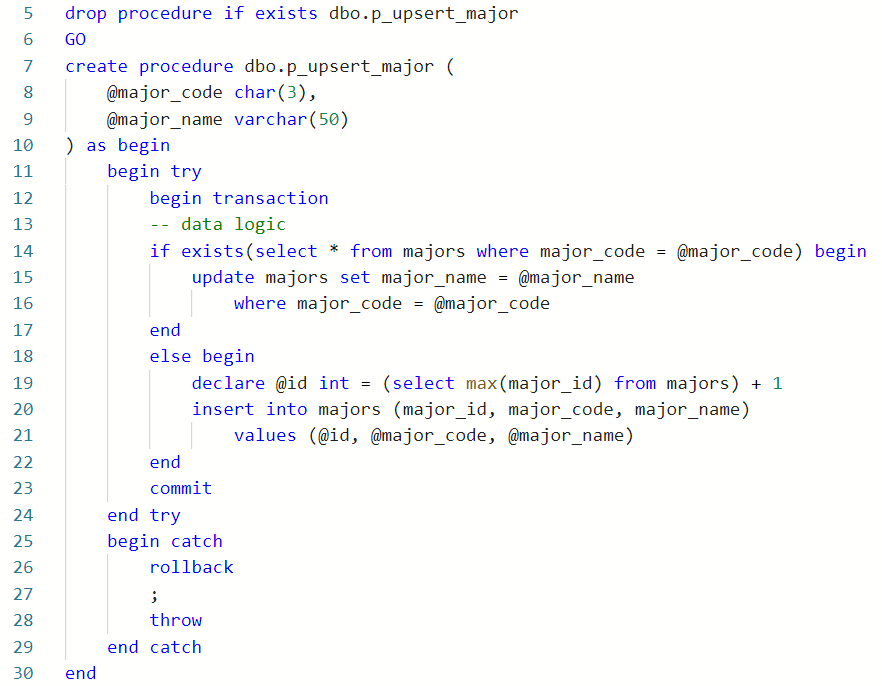
* Microsoft SQL Server DBMS.
* Provision the **TinyUB vBayB** and **Demo** databases using the database provisioner application <https://localhost:5000>.
* Azure Data Studio connected to SQL Server with an open query window.
* Please review the first lab if you require assistance with these tools.

# 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 it. Let’s rewrite this procedure to be transaction safe.

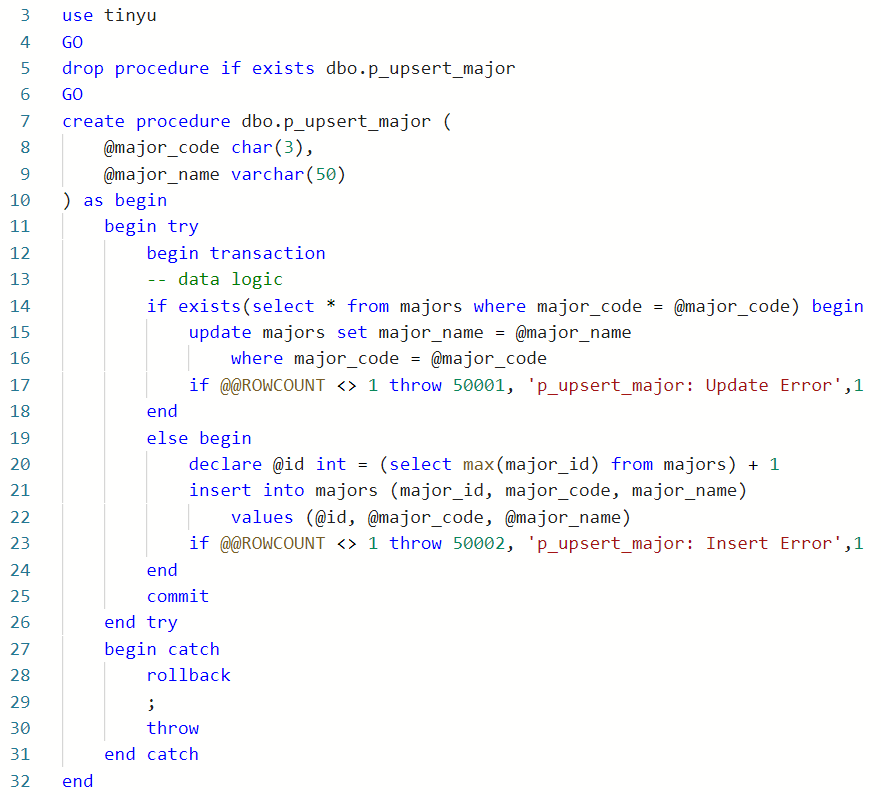
To be transaction safe, it must handle errors and exceptions to the data logic.

**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 one 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, one 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 screen shots provided, please follow the guidelines for submitting a screen shot.

Write the following as SQL programs. For each, include the SQL as a screen shot with the output of the SQL code.

1. Provide a screen shot of your code execution from the walkthrough where you modified **p\_upsert\_major** in the **TinyU** database to be transaction safe.

Graphical user interface, application

Description automatically generated

1. Provide a screen shot of examples of executing the **p\_upsert\_major** procedure to demonstrate it is transaction safe.

Graphical user interface, application, table

Description automatically generated

Table

Description automatically generated

1. Rewrite the **p\_place\_bid** stored procedure from the **vBay** database so that it is transaction safe. Provide a screen shot of the code and its execution.

Graphical user interface, text, application

Description automatically generated

1. 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.

Table

Description automatically generated

1. Rewrite the **p\_rate\_user** stored procedure from the **VBay** database so that it is transaction safe. Provide a screen shot of the code and its execution.

Graphical user interface, application

Description automatically generated

1. Execute the stored procedure in Step 5 to demonstrate the rollback works. You should give a six-star rating and then execute again where someone attempts to rate themselves. Produce a screen shot as evidence the rollback worked.

Graphical user interface, text, application, email

Description automatically generated

1. There is a conceptual data requirement that 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 do we enforce a specific minimum or maximum cardinality instead of just one or “many”?) Write data logic using an instead-of trigger to do this.

Graphical user interface, application

Description automatically generated

1. 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.

* Test – Failed response

Graphical user interface, text, application, email

Description automatically generated

* Success Update/Insert

Graphical user interface, application

Description automatically generated