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:

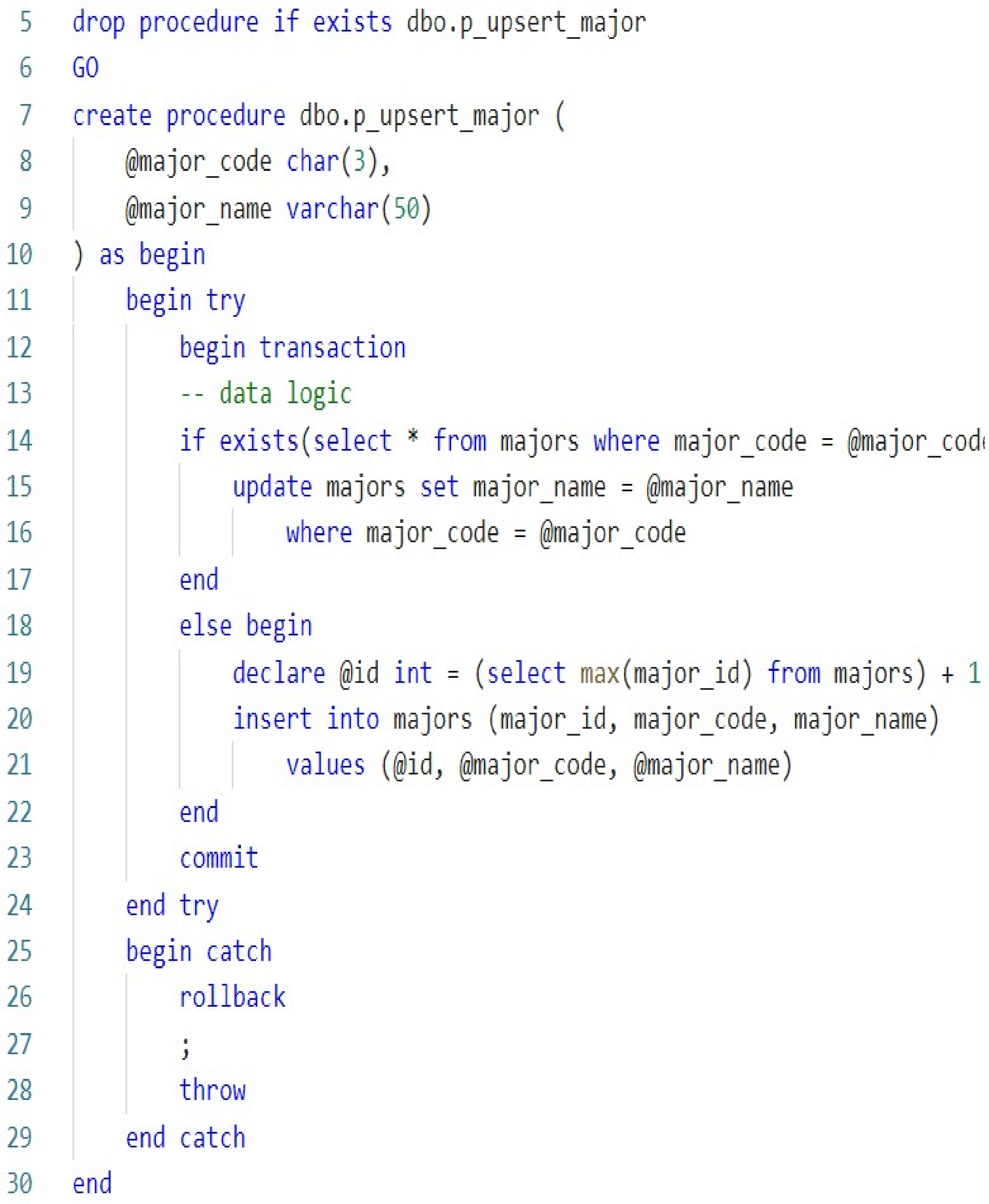
- Microsoft SQL Server DBMS.
- Provision the **TinyUB vBayB** and **Demo** databases using the database provisioner application https://localhost:5000.
- Azure Data Studio connected to SQL Server with an open query window.
- Please review the first lab if you require assistance with these tools.

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.

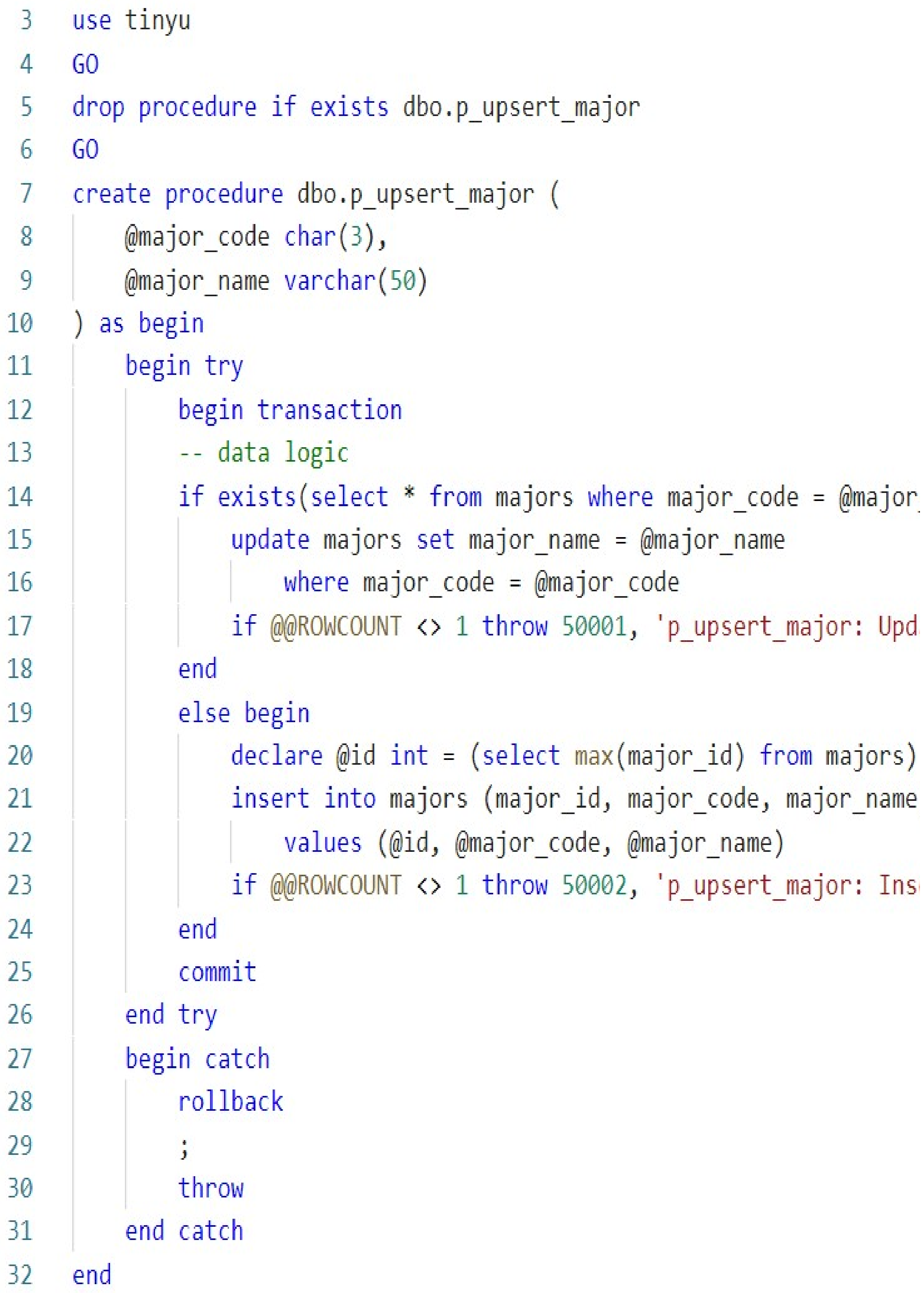
To handle errors, we introduce try/catch:



```
5   drop procedure if exists dbo.p_upsert_major
6   GO
7   create procedure dbo.p_upsert_major (
8       @major_code char(3),
9       @major_name varchar(50)
10  ) as begin
11      begin try
12          begin transaction
13          -- data logic
14          if exists(select * from majors where major_code = @major_cod
15              update majors set major_name = @major_name
16                  where major_code = @major_code
17          end
18          else begin
19              declare @id int = (select max(major_id) from majors) + 1
20              insert into majors (major_id, major_code, major_name)
21                  values (@id, @major_code, @major_name)
22          end
23          commit
24      end try
25      begin catch
26          rollback
27          ;
28          throw
29      end catch
30  end
```

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):



```
use tinyu
GO
drop procedure if exists dbo.p_upsert_major
GO
create procedure dbo.p_upsert_major (
    @major_code char(3),
    @major_name varchar(50)
) as begin
    begin try
        begin transaction
        -- data logic
        if exists(select * from majors where major_code = @major
            update majors set major_name = @major_name
                where major_code = @major_code
            if @@ROWCOUNT <> 1 throw 50001, 'p_upsert_major: Upd
        end
        else begin
            declare @id int = (select max(major_id) from majors)
            insert into majors (major_id, major_code, major_name
                values (@id, @major_code, @major_name)
            if @@ROWCOUNT <> 1 throw 50002, 'p_upsert_major: Ins
        end
        commit
    end try
    begin catch
        rollback
        ;
        throw
    end catch
end
```

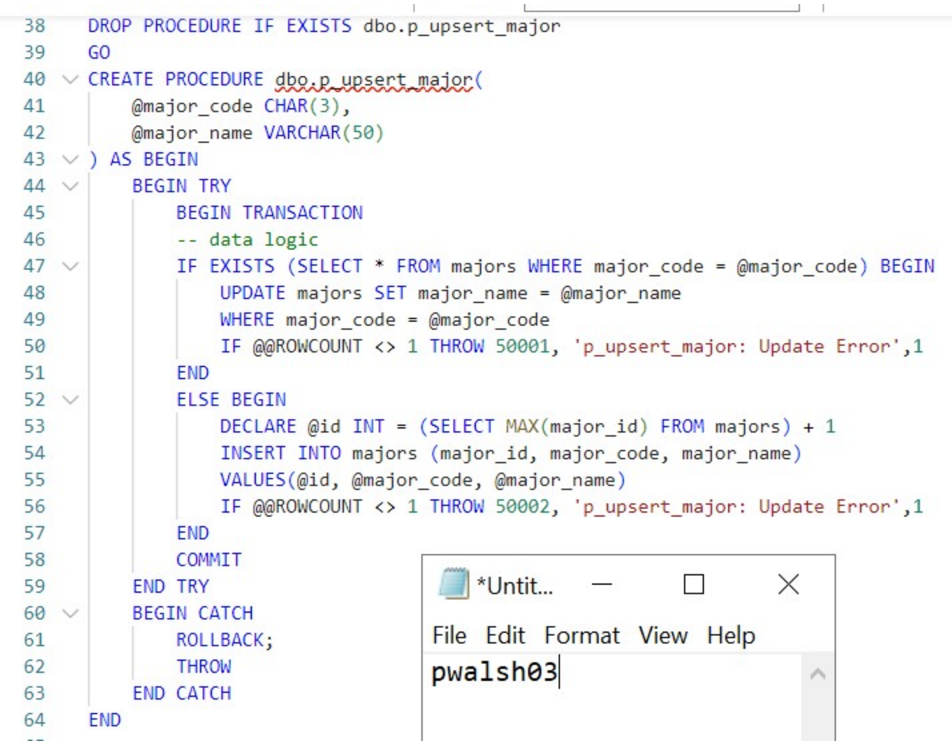
Lines 17 and 23 test the update and insert, respectively, to check whether the proper number of rows was affected, one in this case.

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.

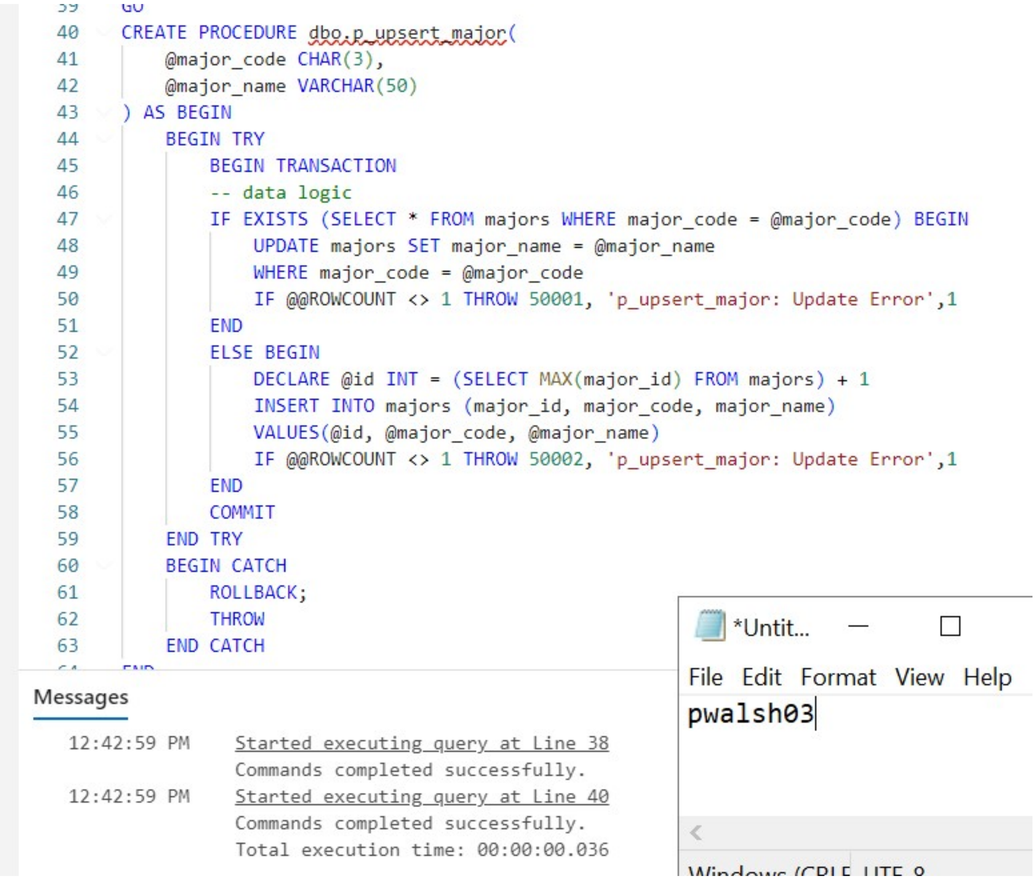
- Provide a screen shot of your code execution from the walkthrough where you modified **p_upsert_major** in the **TinyU** database to be transaction safe.



```sql
DROP PROCEDURE IF EXISTS dbo.p_upsert_major
GO
CREATE PROCEDURE dbo.p_upsert_major(
    @major_code CHAR(3),
    @major_name VARCHAR(50)
) AS BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
        -- data logic
        IF EXISTS (SELECT * FROM majors WHERE major_code = @major_code) BEGIN
            UPDATE majors SET major_name = @major_name
            WHERE major_code = @major_code
            IF @@ROWCOUNT <> 1 THROW 50001, 'p_upsert_major: Update Error',1
        END
        ELSE BEGIN
            DECLARE @id INT = (SELECT MAX(major_id) FROM majors) + 1
            INSERT INTO majors (major_id, major_code, major_name)
            VALUES(@id, @major_code, @major_name)
            IF @@ROWCOUNT <> 1 THROW 50002, 'p_upsert_major: Update Error',1
        END
        COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW
    END CATCH
END
```

pwalsh03

- Provide a screen shot of examples of executing the **p_upsert_major** procedure to demonstrate it is transaction safe.



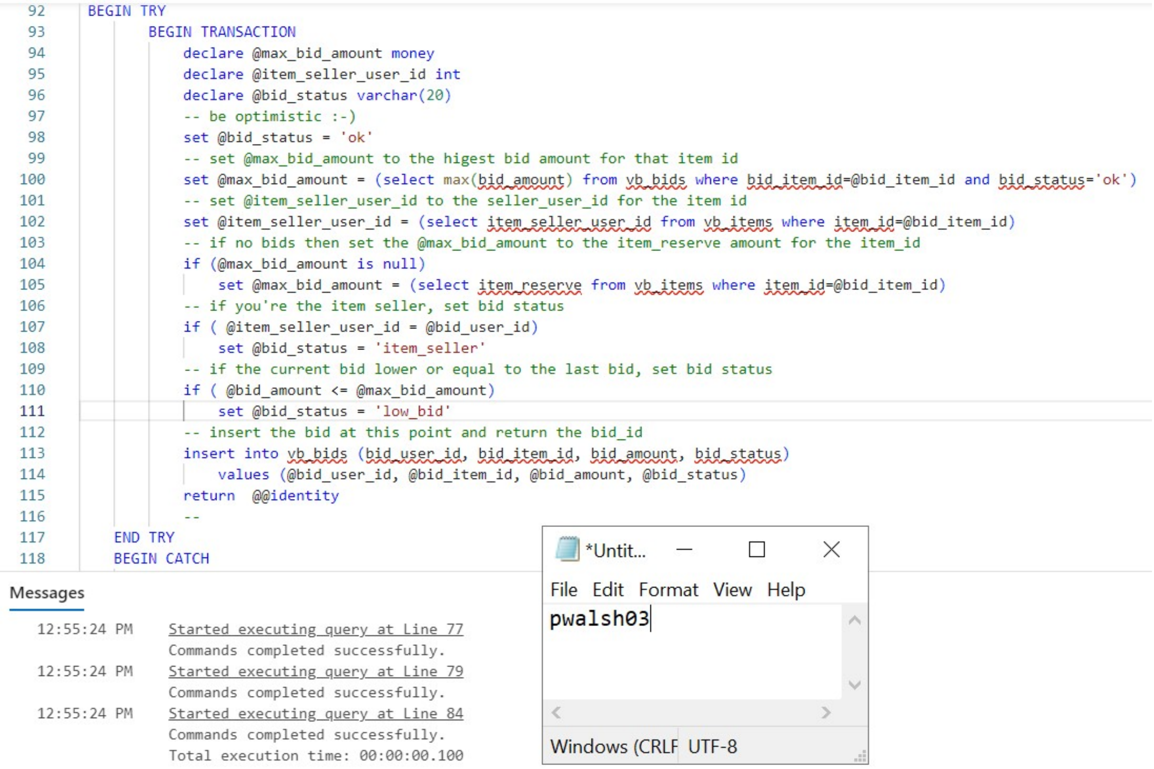
```
GO
CREATE PROCEDURE dbo.p_upsert_major(
    @major_code CHAR(3),
    @major_name VARCHAR(50)
) AS BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
        -- data logic
        IF EXISTS (SELECT * FROM majors WHERE major_code = @major_code) BEGIN
            UPDATE majors SET major_name = @major_name
            WHERE major_code = @major_code
            IF @@ROWCOUNT <> 1 THROW 50001, 'p_upsert_major: Update Error',1
        END
        ELSE BEGIN
            DECLARE @id INT = (SELECT MAX(major_id) FROM majors) + 1
            INSERT INTO majors (major_id, major_code, major_name)
            VALUES(@id, @major_code, @major_name)
            IF @@ROWCOUNT <> 1 THROW 50002, 'p_upsert_major: Update Error',1
        END
        COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW
    END CATCH
END
```

Messages

```
12:42:59 PM    Started executing query at Line 38
               Commands completed successfully.
12:42:59 PM    Started executing query at Line 40
               Commands completed successfully.
               Total execution time: 00:00:00.036
```

pwalsh03

- 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.



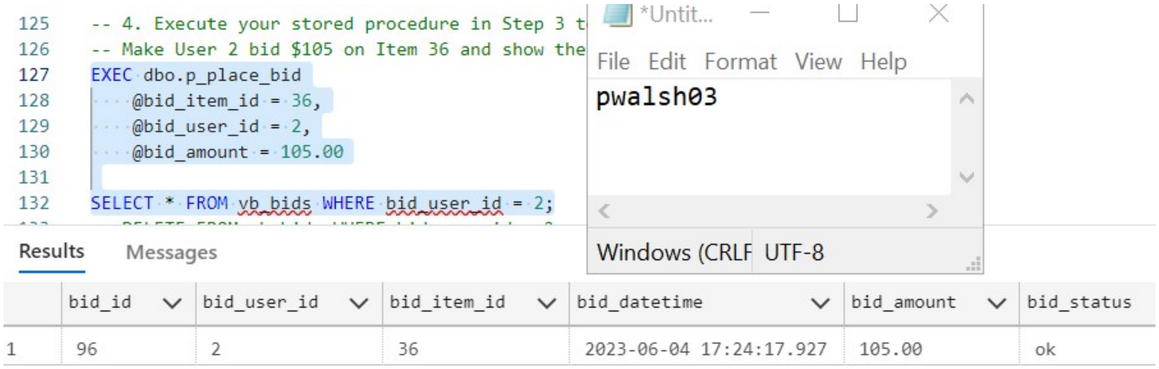
```
BEGIN TRY
       BEGIN TRANSACTION
           declare @max_bid_amount money
           declare @item_seller_user_id int
           declare @bid_status varchar(20)
           -- be optimistic :-)
           set @bid_status = 'ok'
           -- set @max_bid_amount to the higest bid amount for that item id
           set @max_bid_amount = (select max(bid_amount) from vb_bids where bid_item_id=@bid_item_id and bid_status='ok')
           -- set @item_seller_user_id to the seller_user_id for the item id
           set @item_seller_user_id = (select item_seller_user_id from vb_items where item_id=@bid_item_id)
           -- if no bids then set the @max_bid_amount to the item_reserve amount for the item_id
           if (@max_bid_amount is null)
               set @max_bid_amount = (select item_reserve from vb_items where item_id=@bid_item_id)
           -- if you're the item seller, set bid status
           if ( @item_seller_user_id = @bid_user_id)
               set @bid_status = 'item_seller'
           -- if the current bid lower or equal to the last bid, set bid status
           if ( @bid_amount <= @max_bid_amount)
               set @bid_status = 'low_bid'
           -- insert the bid at this point and return the bid_id
           insert into vb_bids (bid_user_id, bid_item_id, bid_amount, bid_status)
               values (@bid_user_id, @bid_item_id, @bid_amount, @bid_status)
           return  @@identity
           --
   END TRY
   BEGIN CATCH
```

Messages

```
12:55:24 PM    Started executing query at Line 77
               Commands completed successfully.
12:55:24 PM    Started executing query at Line 79
               Commands completed successfully.
12:55:24 PM    Started executing query at Line 84
               Commands completed successfully.
               Total execution time: 00:00:00.100
```

pwalsh03

- 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.



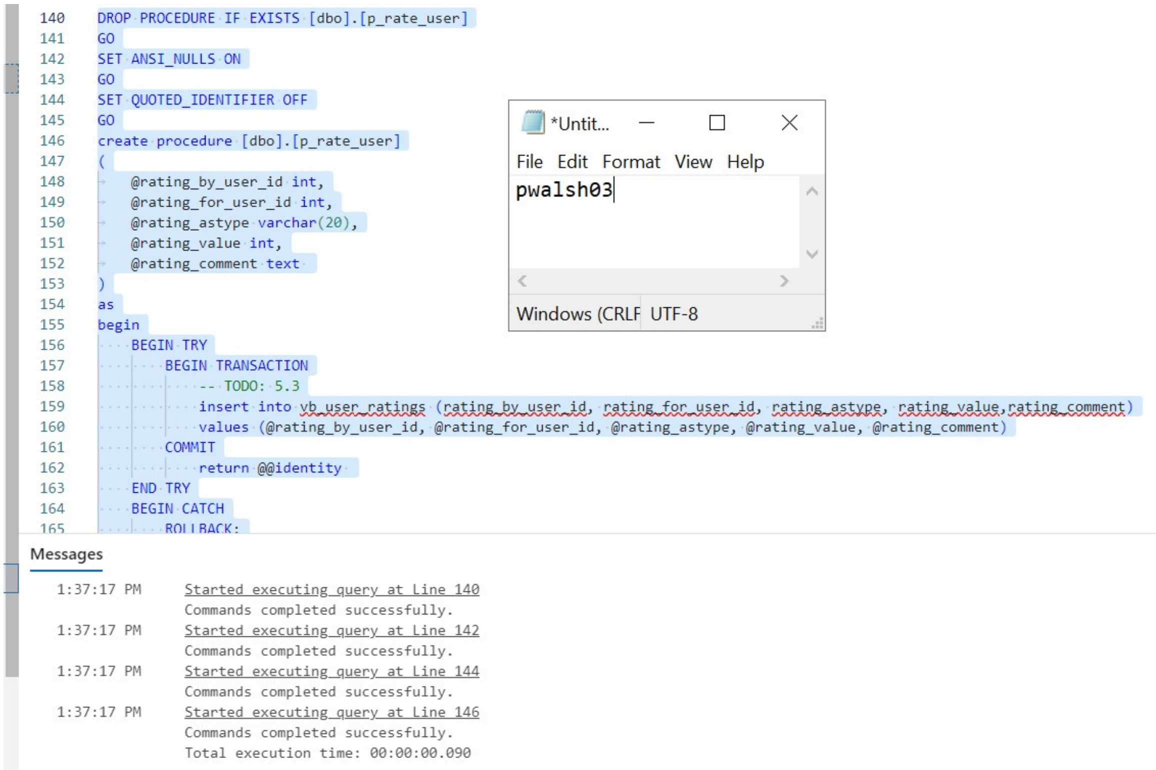
```
-- 4. Execute your stored procedure in Step 3 t
-- Make User 2 bid $105 on Item 36 and show the
EXEC dbo.p_place_bid
    @bid_item_id = 36,
    @bid_user_id = 2,
    @bid_amount = 105.00

SELECT * FROM vb_bids WHERE bid_user_id = 2;
```

pwalsh03

	bid_id	bid_user_id	bid_item_id	bid_datetime	bid_amount	bid_status
1	96	2	36	2023-06-04 17:24:17.927	105.00	ok

- 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.



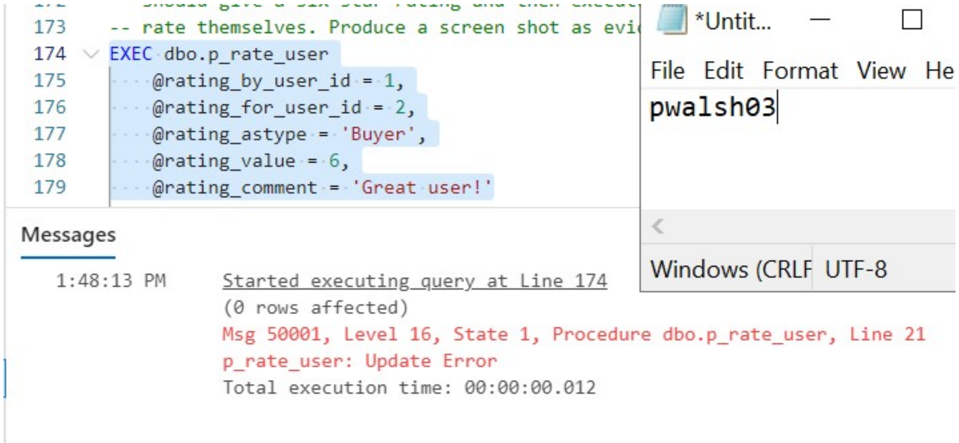
```
DROP PROCEDURE IF EXISTS [dbo].[p_rate_user]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER OFF
GO
create procedure [dbo].[p_rate_user]
(
    @rating_by_user_id int,
    @rating_for_user_id int,
    @rating_astype varchar(20),
    @rating_value int,
    @rating_comment text
)
as
begin
    BEGIN TRY
        BEGIN TRANSACTION
            -- TODO: 5.3
            insert into vb_user_ratings (rating_by_user_id, rating_for_user_id, rating_astype, rating_value,rating_comment)
            values (@rating_by_user_id, @rating_for_user_id, @rating_astype, @rating_value, @rating_comment)
        COMMIT
            return @@identity
    END TRY
    BEGIN CATCH
        ROLLBACK;
```

Messages

```
1:37:17 PM      Started executing query at Line 140
                Commands completed successfully.
1:37:17 PM      Started executing query at Line 142
                Commands completed successfully.
1:37:17 PM      Started executing query at Line 144
                Commands completed successfully.
1:37:17 PM      Started executing query at Line 146
                Commands completed successfully.
                Total execution time: 00:00:00.090
```

pwalsh03

- 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.

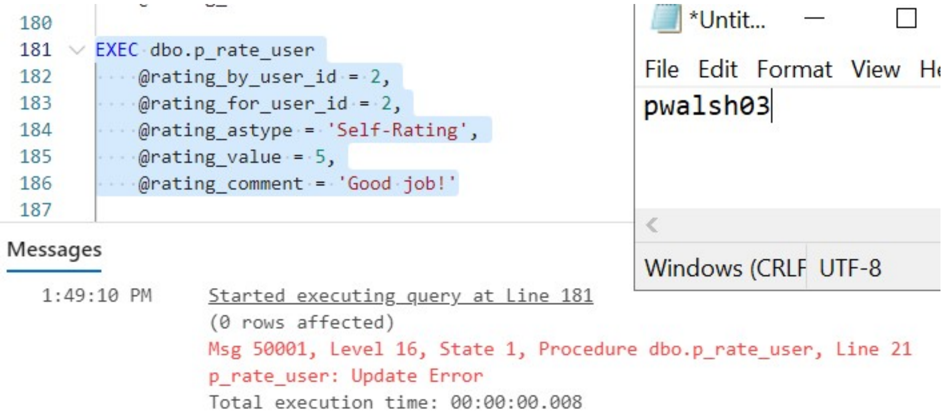


```
-- rate themselves. Produce a screen shot as evi
EXEC dbo.p_rate_user
    @rating_by_user_id = 1,
    @rating_for_user_id = 2,
    @rating_astype = 'Buyer',
    @rating_value = 6,
    @rating_comment = 'Great user!'
```

Messages

```
1:48:13 PM      Started executing query at Line 174
                (0 rows affected)
                Msg 50001, Level 16, State 1, Procedure dbo.p_rate_user, Line 21
                p_rate_user: Update Error
                Total execution time: 00:00:00.012
```

pwalsh03



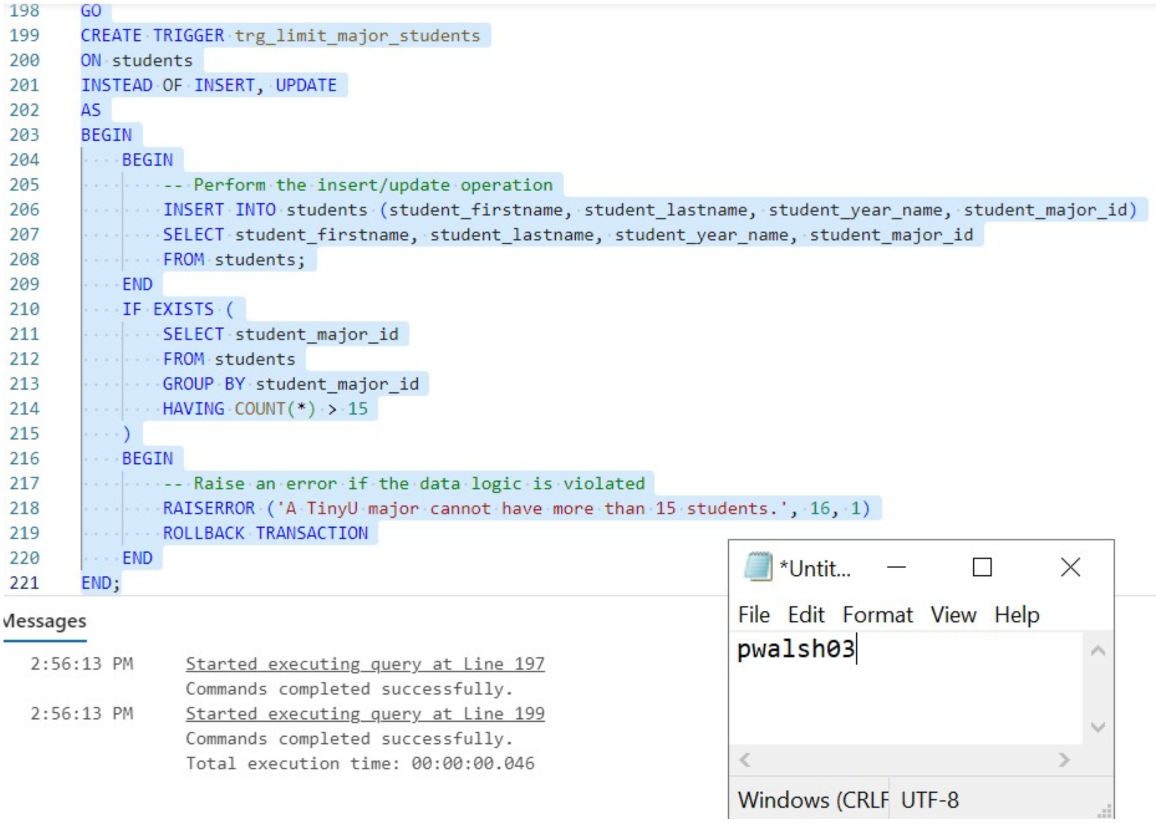
```
180
181  EXEC dbo.p_rate_user
182      @rating_by_user_id = 2,
183      @rating_for_user_id = 2,
184      @rating_astype = 'Self-Rating',
185      @rating_value = 5,
186      @rating_comment = 'Good job!'
187
```

Messages

```
1:49:10 PM     Started executing query at Line 181
               (0 rows affected)
               Msg 50001, Level 16, State 1, Procedure dbo.p_rate_user, Line 21
               p_rate_user: Update Error
               Total execution time: 00:00:00.008
```

pwalsh03

- 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.



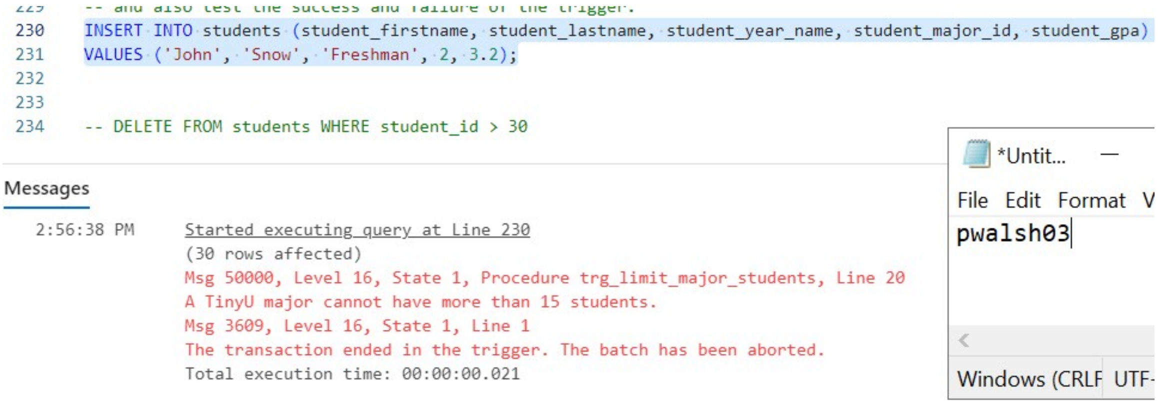
```
198  GO
199  CREATE TRIGGER trg_limit_major_students
200  ON students
201  INSTEAD OF INSERT, UPDATE
202  AS
203  BEGIN
204      BEGIN
205          -- Perform the insert/update operation
206          INSERT INTO students (student_firstname, student_lastname, student_year_name, student_major_id)
207          SELECT student_firstname, student_lastname, student_year_name, student_major_id
208          FROM students;
209      END
210      IF EXISTS (
211          SELECT student_major_id
212          FROM students
213          GROUP BY student_major_id
214          HAVING COUNT(*) > 15
215      )
216      BEGIN
217          -- Raise an error if the data logic is violated
218          RAISERROR ('A TinyU major cannot have more than 15 students.', 16, 1)
219          ROLLBACK TRANSACTION
220      END
221  END;
```

Messages

```
2:56:13 PM     Started executing query at Line 197
               Commands completed successfully.
2:56:13 PM     Started executing query at Line 199
               Commands completed successfully.
               Total execution time: 00:00:00.046
```

pwalsh03

- 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.



```
-- and also test the success and failure of the trigger.
INSERT INTO students (student_firstname, student_lastname, student_year_name, student_major_id, student_gpa)
VALUES ('John', 'Snow', 'Freshman', 2, 3.2);


-- DELETE FROM students WHERE student_id > 30
```

Messages

```
2:56:38 PM    Started executing query at Line 230
              (30 rows affected)
              Msg 50000, Level 16, State 1, Procedure trg_limit_major_students, Line 20
              A TinyU major cannot have more than 15 students.
              Msg 3609, Level 16, State 1, Line 1
              The transaction ended in the trigger. The batch has been aborted.
              Total execution time: 00:00:00.021
```

pwalsh03