Chapter 7 – Part 3

Stored Procedure, Function & Trigger

Contents

A. Sport Shop Problem

B. Solution

A. Sport Shop Problem

PST, a sport shop, has been in success of business lately. Therefore, it makes sense to establish a database to manage their products and selling bills.

To easily managing, they classify their products into different types which are included type code and type name. Each product type may have many products or not and a product must belong to a product type. The sport product is described by the following properties: product code, product name, quantity, and buying price.

Otherwise, when a bill is created the system stores the following information: bill id, customer name, date and the list of products included product id, quantity and selling price. A bill must have at least one sold-product (bill item).

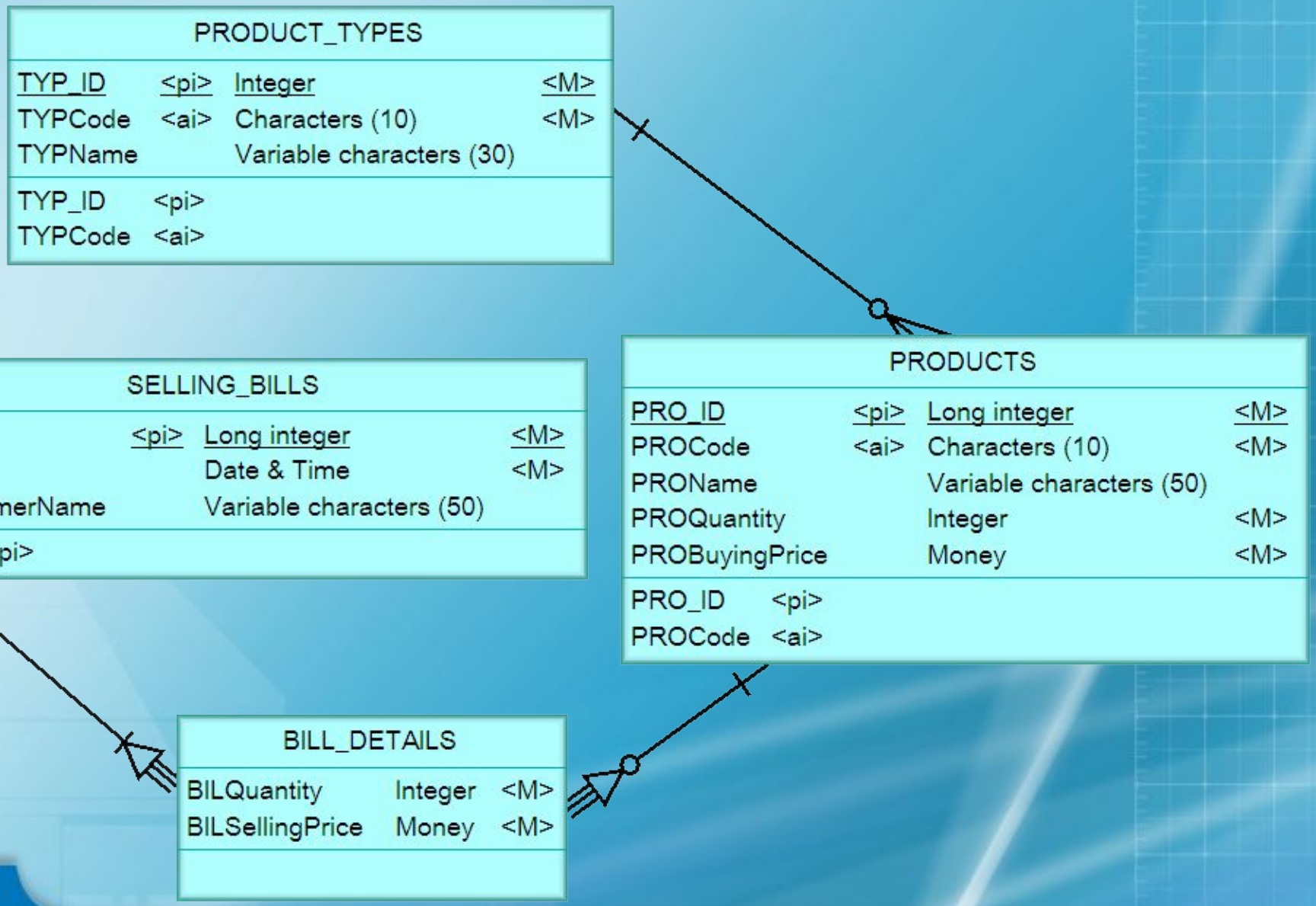
Do the following requirements

- *Design a database with ERD & DB Design diagram.*
- *Create DDL statements to implement the DB on SQL Server 2005.*
- *Create a stored procedure to insert product type information.*
- *Create a stored procedure to insert product information.*
- *Create a function to return the number of products for a given product type.*
- *Create a function to return the number of products for each product type.*
- *Create a function to return the number of rows for PRODUCT_TYPES and PRODUCTS tables.*
- *Create a triggers to check the constraint on SELLING_BILLS for inserting information.*
- *Create a trigger to check the constraint on BILLS_DETAILS for deleting information.*
- *Create a trigger to check the selling-price must be greater than or equal to buying-price of the same product on BILLS_DETAILS table.*
- *Create a stored procedure to delete a bill*
- *Create a stored procedure to insert bill and bill details information.*

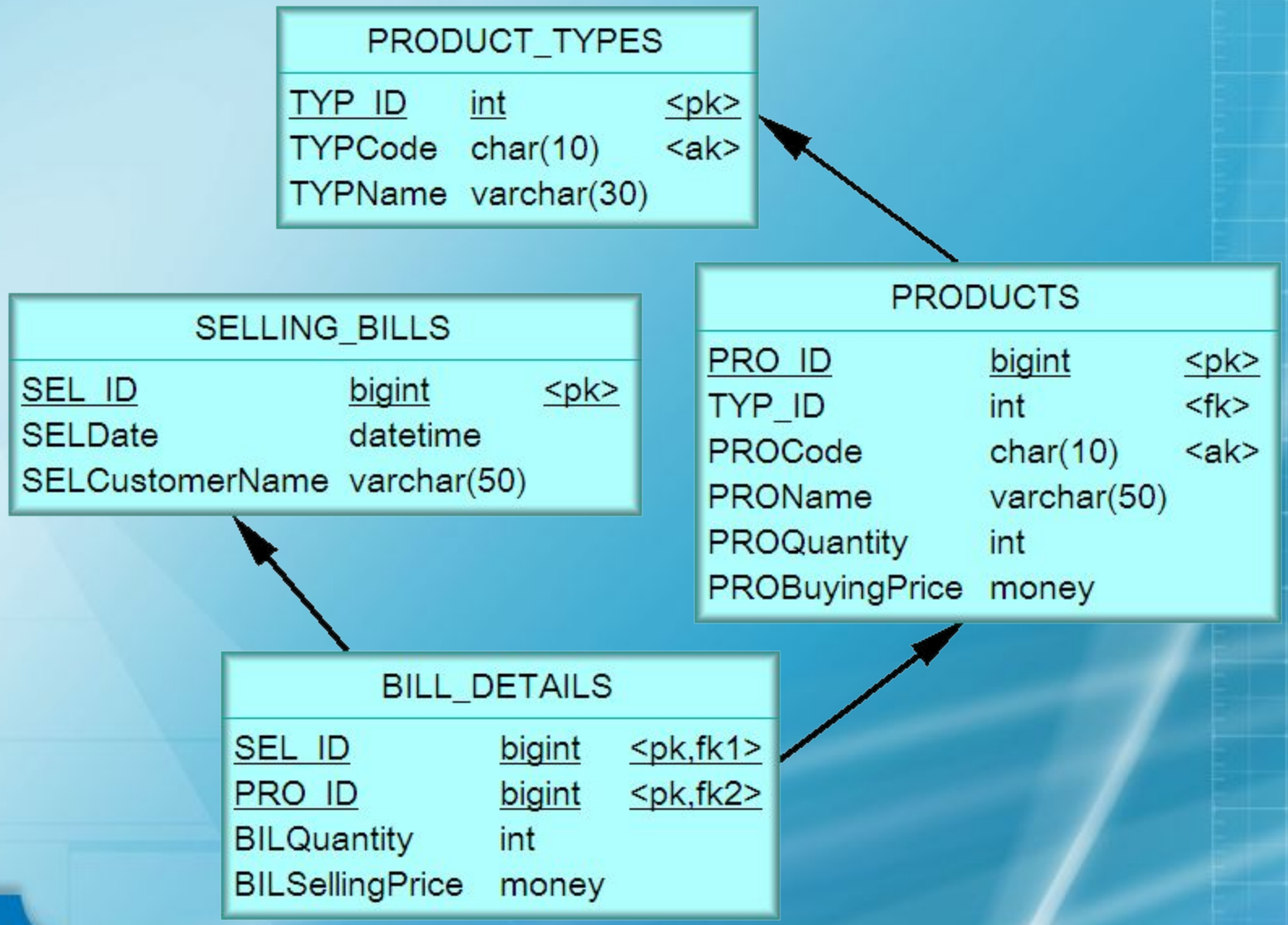
B. Solution

1. Create Logical Diagram
2. Create Physical Diagram
3. Write DDL statements
4. Create stored procedures
5. Create functions
6. Create triggers
7. Create advance stored procedures

1. Create Logical Diagram



2. Create Physical Diagram



3. Write DDL Statements

```
CREATE TABLE PRODUCT_TYPES (
        TYP_ID          int             not null   Primary Key ,
    TYPCode         char(10)        not null   unique,
    TYPName         varchar(30)
);
CREATE TABLE PRODUCTS (
    PRO_ID          bigint          Primary Key     identity,
    TYP_ID          int                 not null,
    PROCode         char(10)            not null    unique,
    PROName         varchar(50),
    PROQuantity     int                 not null    default 0,
    PROBuyingPrice      money       not null    default 0,
      Constraint FK_Product Foreign Key (TYP_ID) References
                  PRODUCT_TYPES (TYP_ID) on update cascade,
      Constraint CKC_PROQuantity check (PROQuantity >= 0),
      Constraint CKC_PROBuyingPrice check (PROBuyingPrice >= 0)
);
```

```
CREATE TABLE SELLING_BILLS (
  SEL_ID              bigint          identity Primary Key,
  SELDate             datetime        not null      default getdate(),
  SELCustomerName     varchar (50)
);
CREATE TABLE BILL_DETAILS (
  SEL_ID              bigint        not null,
  PRO_ID              bigint        not null,
  BILQuantity         int           not null      default 1,
  BILSellingPrice     money         not null,
  Constraint CKC_BILQuantity check (BILQuantity >= 1),
  Constraint CKC_BILSelling check (BILSellingPrice >= 0),
  Constraint PK_BILL_DETAILS Primary Key (SEL_ID, PRO_ID),
  Constraint FK_BILL_DETAILS1 Foreign Key (PRO_ID)
        REFERENCES PRODUCTS (PRO_ID) on update cascade,
  Constraint FK_BILL_DETAILS2 Foreign Key (SEL_ID)
    References SELLING_BILLS (SEL_ID)
            on update cascade on delete cascade  );
```

4. Create stored procedures

4.1. What is a stored procedure?

4.2. Stored Procedure vs. SQL Statement

4.3. Stored procedure for Product type table

4.4. Stored procedure for Product table

4.1. What is a stored procedure?

- A stored procedure is a collection of T-SQL statements that SQL Server compiles into a single execution plan.
- Procedure is stored in cache area of memory when the stored procedure is created so that it can be used repeatedly. SQL Server does not have to recompile it every time the stored procedure is run.
- It can accept input parameters, return output values as parameters, or return success or failure status messages.
- Syntax

```
CREATE PROC[EDURE] procedure_name
    [ {@parameter_name data_type} [= default] [OUTPUT]] [,...,n]
AS
     T-SQL_statement(s)
```

4.2. Stored Procedure vs. SQL Statement

SQL Statement

First Time
- Check syntax
- Compile
- Execute
- Return data

Second Time
- Check syntax
- Compile
- Execute
- Return data

Stored

Creating
- Check syntax
- Compile

First Time
- Execute
- Return data

Second Time
- Execute
- Return data

4.3. Stored procedure for Product type table

Create a stored procedure to insert product type information.

```
CREATE PROC insertProductType   @TYP_ID int,
        @TYPCode char(10),  @TYPName varchar(30) = null
AS
        Insert into Product_Types
        values (@TYP_ID, @TYPCode, @TYPName)
```

Use:

```
Exec insertProductType 1,'TYP-1','Product Type 1'
```

4.4. Stored procedure for Product table

Create a stored procedure to insert product information.

```
CREATE PROC insertProduct   @typid int, @procode char(10),
            @proname varchar(50) = null, @proquantity int = 0,
            @proprice money = 0
AS
    Declare @checkexist int
    Select @checkexist = TYP_ID from Product_Types
    Where TYP_ID = @typid
    IF  (@checkexist is null)
    Begin
            Print 'This product type does not exist in system!'
            Return
    End
    Insert into Products
    values (@typid,@procode,@proname,@proquantity,@proprice)
```

5. Create functions

5.1. What is a function?

5.2. Scalar function

5.3. Inline table-valued function

5.4. Multi-statement table-valued function

5.1. What is a function?

- Similar to Stored Procedure with value returning.
- SQL Server supports three types of user-defined functions:
  - Scalar function
  - Inline table-valued function
  - Multi-statement table-valued function
- Syntax

```
CREATE FUNCTION function_name ([parameter(s)])
    RETURNS    Data-Type
AS
BEGIN
    T-SQL Statements
END
```

5.2. Scalar function

Create a function to return the number of products for a particular product type.

```
CREATE FUNCTION numberOfProduct (@typid int)
        RETURNS int
AS
BEGIN
    DECLARE @numpro int
    SELECT @numpro = count(PRO_ID) FROM Products
    WHERE TYP_ID = @typid
    RETURN @numpro
END
```

Use:

```
SELECT  dbo.numberOfProduct (1)
```

5.3. Inline table-valued function

Create a function to return the number of products for each product type.

```
CREATE FUNCTION numberOfProductAll ()  RETURNS  table
AS
     RETURN (SELECT TYPCode, TYPName,
                                  Count(PRO_ID) as numpro
                    FROM Product_Types PT, Products P
                    WHERE  PT.TYP_ID = P.TYP_ID
                    GROUP BY   TYPCode, TYPName)
```

Use:

```
SELECT  * FROM numberOfProductAll()
```

5.4. Multi-statement table-valued function

Create a function to return the number of rows for product type and product tables.

```
CREATE FUNCTION rowOfTables () RETURNS
            @table table (TableName varchar(50), Rows int)
AS
BEGIN
    Declare  @num int
    Select @num = count(TYP_ID) From Product_Types
    Insert into @table values('Product_Types', @num)
   Select @num = count(PRO_ID) From Products
    Insert into @table values('Product', @num)
    Return
END
```

Use:

```
Select * From rowOfTables()
```

6. Create a trigger

6.1. What is a trigger?

6.2. Deleted and Inserted tables

6.3. Trigger creating syntax

6.4. Trigger on SELLING_BILLS table

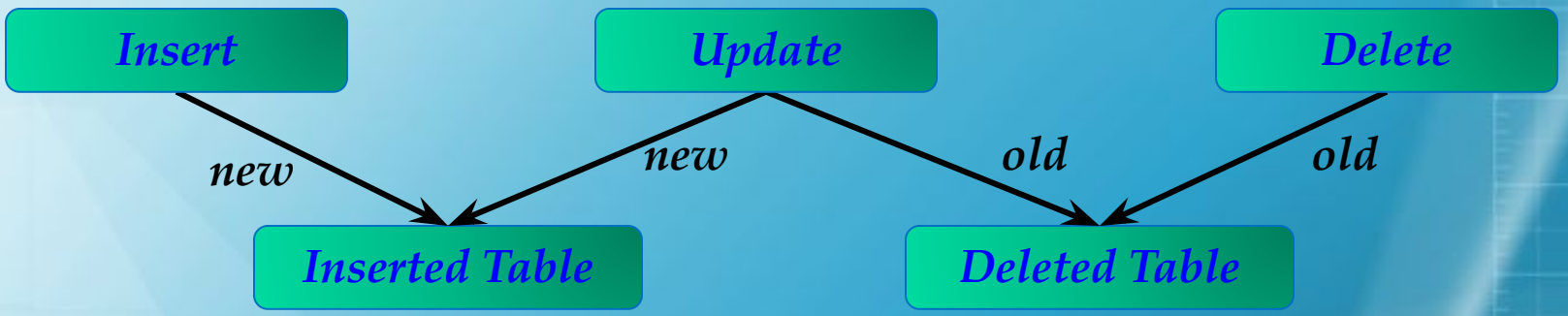
6.5. Triggers on BILL_DETAILS table

6.1. What is a trigger?

- *A trigger is a special type of stored procedure that is executed automatically as part of a data modification.*
- *A trigger is created on a table and associated with one or more actions linked with a data modification (INSERT, UPDATE, or DELETE).*
- *When one of the actions for which the trigger is defined occurs, the trigger fires automatically*
- *Following are some examples of trigger uses:*
  - *Maintenance of duplicate and derived data*
  - *Complex column constraints*
  - *Cascading referential integrity*
  - *Complex defaults*
  - *Inter-database referential integrity*

6.2. Deleted and Inserted tables

- When you create a trigger, you have access to two temporary tables (the deleted and inserted tables). They are referred to as tables, but they are different from true database tables. They are stored in memory—not on disk.
- When the insert, update or delete statement is executed. All data will be copied into these tables with the same structure.



- The values in the inserted and deleted tables are accessible only within the trigger. Once the trigger is completed, these tables are no longer accessible.

6.3. Trigger creating syntax

```
CREATE TRIGGER trigger_name
    ON <table_name>
    {FOR | AFTER}
    {[DELETE] [,] [INSERT] [,] [UPDATE]}
AS
BEGIN
    T-SQL Statements
END
```

6.4. Trigger on SELLING_BILLS table

Create a trigger to check the constraint of SELLING_BILLS table when a row is inserted: One bill must have at least one bill item.

```
CREATE TRIGGER SellingBillsOnInsert
ON  Selling_Bills  FOR Insert
AS
BEGIN
      Declare @selid int, @numItems int
     Select @selid = SEL_ID From inserted
     Select @numItems = count(PRO_ID) From BILL_DETAILS
     Where SEL_ID = @selid
     IF (@numItems = 0)
     BEGIN
      print 'This bill has no bill item!'
      rollback tran
     END
END
```

6.5. Triggers on BILL_DETAILS table

6.5.1 Trigger on deleting

6.5.2 Trigger for Interrelation constraint

6.5.1 Trigger on deleting

Create a trigger to check the constraint of BILL_DETAILS table when a row is deleted: One bill must have at least one bill item.

```
Create trigger BillDetailsOnDelete
on Bill_Details
for delete
as
    Declare @selid int, @numCurrentRows int
    Select @selid = SEL_ID from Deleted group by SEL_ID
    Select @numCurrentRows = count(PRO_ID)
    From Bill_Details Where SEL_ID = @selid
    IF (@numCurrentRows = 0)
    BEGIN
        print 'Cannot delete this (these) bill item(s)!'
        rollback tran
    END
```

6.5.2 Trigger for Interrelation constraint

Create a trigger to check the selling-price must be greater than or equal to buying-price of the same product on bill details table.

```
Create trigger checkSellingPrice
on BILL_DETAILS
for insert, update
as
    Declare @sellprice money, @buyprice money, @proid int
    Select @proid= PRO_ID, @sellprice = BILSellingPrice from
inserted
    Select @buyprice = PROBuyingPrice
    From PRODUCTS where PRO_ID = @proid
    IF (@sellprice < @buyprice)
    BEGIN
        print 'Selling Price must be greater than or equal to Buying
Price!'
rollback tran
END
```

7. Create advance stored procedures

7.1. Stored procedure to delete a bill

7.2. Stored procedure to insert bill and bill details

7.1. Stored procedure to delete a bill

```
CREATE PROC deleteBill (@selid int)
as
    ALTER TABLE Bill_Details
        Disable trigger BillDetailsOnDelete
    Delete from Selling_Bills where SEL_ID = @selid
     ALTER TABLE Bill_Details
        Enable trigger BillDetailsOnDelete
```

7.2. Stored procedure to insert bill and bill details

```
CREATE PROC insertBill @proid int, @quant int, @price money,
    @cusname varchar(50) = null
AS
    Declare @lastBillID int
    ALTER TABLE Selling_Bills
            Disable trigger SellingBillsOnInsert
    IF (@cusname is not null)
            INSERT INTO Selling_Bills(SELCustomerName)
                Values(@cusname)
    Select @lastBillID = max(SEL_ID) From Selling_Bills
    Insert into Bill_Details
                Values(@lastBillID, @proid, @quant, @price)
    ALTER TABLE selling_bills
        Enable trigger SellingBillsOnInsert
```