Define a view called

ShipperView (CustName, CustAddr, ShipID, BookIsbn, BookQuantity)

that is required by a person (or process) responsible for "shipping" **books purchased by customers that are ready to be shipped**: that is, the books

- have been paid for [PaymentClearanceDate is not null],
- the books have not been shipped yet [ShipDate is null], and
- there are enough copies in stock of the book to satisfy the order

Use the accompanying UML as your guide to table definitions.

See next page for "hint"

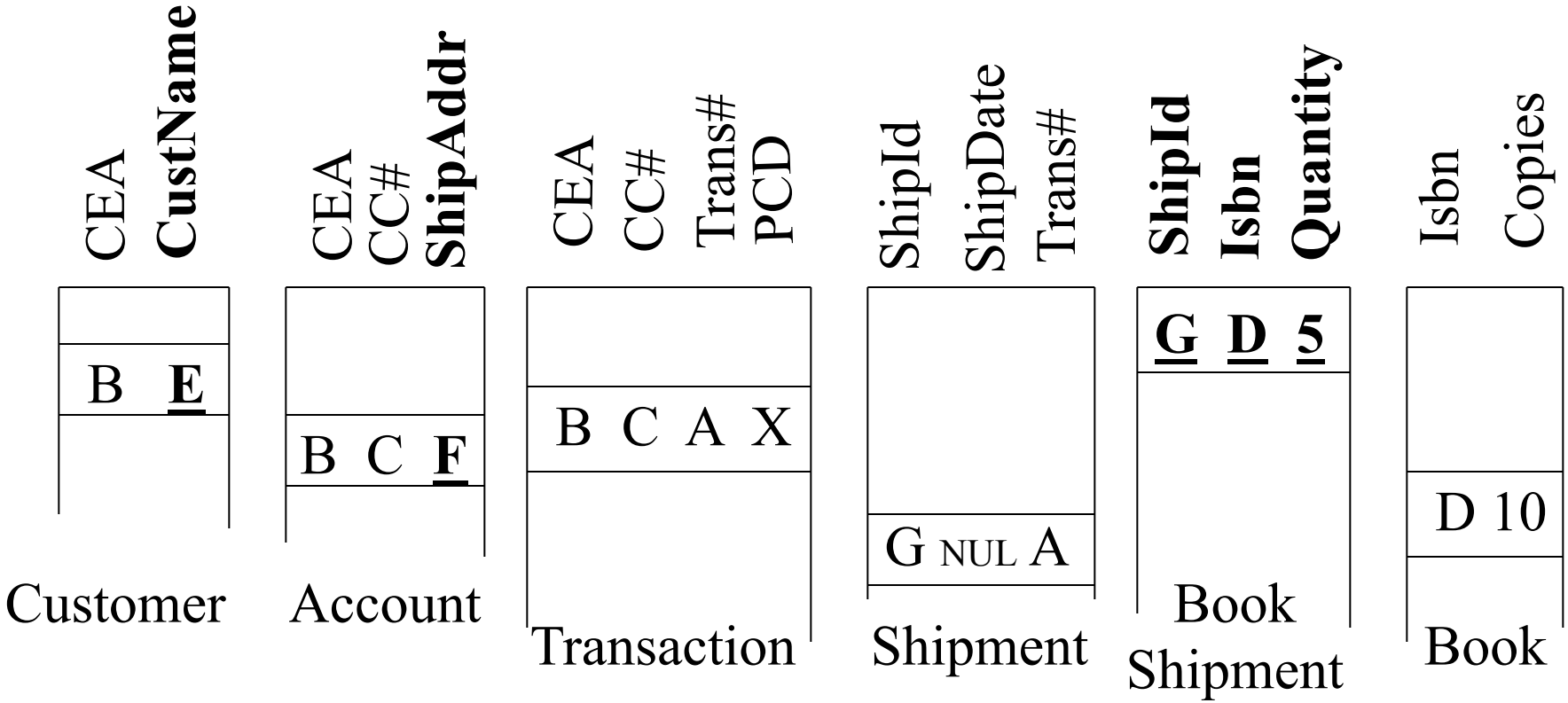
Define a view called

ShipperView (CustName, CustAddr, ShipID, BookIsbn, BookQuantity)

that is required by a person (or process) responsible for "shipping" **books purchased by customers that are ready to be shipped**: that is, the books

- have been paid for [PaymentClearanceDate is not null],
- the books have not been shipped yet [ShipDate is null], and
- there are enough copies in stock of the book to satisfy the order

Use the accompanying UML as your guide to table definitions.



Define a view called

ShipperView (CustName, CustAddr, ShipID, BookIsbn, BookQuantity)

that is required by a person (or process) responsible for "shipping" **books purchased by customers that are ready to be shipped**: that is, the books have been paid for [PaymentClearanceDate is not null], the books have not been shipped yet [ShipDate is null], and there are enough copies in stock of the book to satisfy the order

```
CREATE VIEW ShipperView (Name, Addr, ShipId, Isbn, Quantity)
AS SELECT C.CustName, A.ShippingAddr, BSt.ShipId, BSt.Isbn, BSt.Quantity
    FROM Customer C, Account A, Transaction T,
            Shipment St, BookShipment BSt, Book B
    WHERE C.CustEmailAddr = A.CustEmailAddr AND
            A.CustEmailAddr = T.CustEmailAddr AND
            A.CreditCardNumber = T.CreditCardNumber AND
            T.PaymentClearanceDate IS NOT NULL AND
            T.TransNumber = St.TransNumber AND
            St.ShipDate IS NULL AND
            St.ShipId = BSt.ShipId AND
            BSt.Isbn = B.Isbn AND
            B.CopiesInStock >= BSt.Quantity
```

Important aside: a view can show information on a need-to-know basis. For example. even though we need the Transaction table to create the ShipperView, there is no reason that a 'shipper' have access to a customer's credit card number, so this information from Transaction is not shown a shipper

Now, write an INSTEAD OF TRIGGER that implements

DELETE FROM ShipperView WHERE ShipID = 'X'

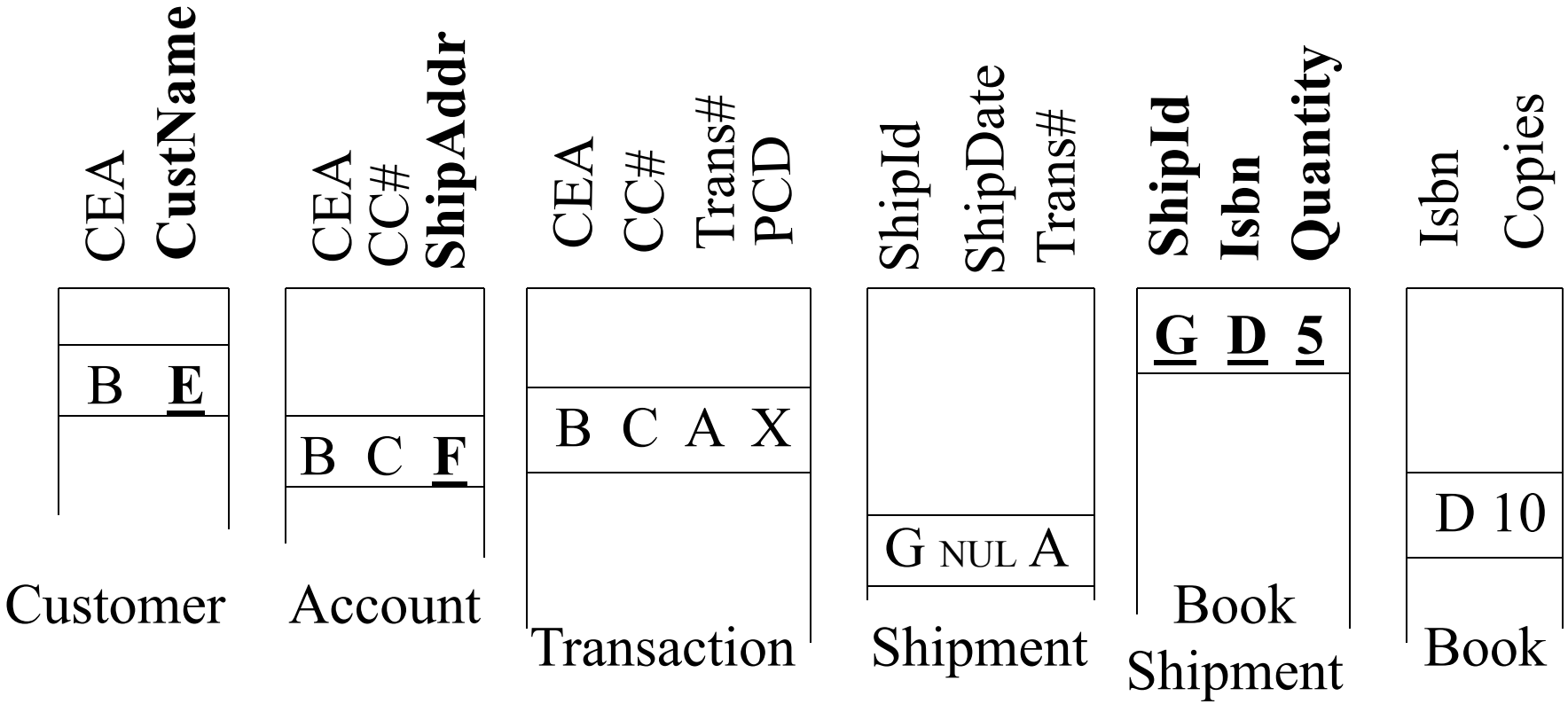
```
CREATE VIEW ShipperView (Name, Addr, ShipId, Isbn, Quantity)
AS SELECT C.CustName, A.ShippingAddr, BSt.ShipId, BSt.Isbn, BSt.Quantity
    FROM Customer C, Account A, Transaction T,
            Shipment St, BookShipment BSt, Book B
    WHERE C.CustEmailAddr = A.CustEmailAddr AND
            A.CustEmailAddr = T.CustEmailAddr AND
            A.CreditCardNumber = T.CreditCardNumber AND
            T.PaymentClearanceDate IS NOT NULL AND
            T.TransNumber = St.TransNumber AND
            St.ShipDate IS NULL AND
            St.ShipId = BSt.ShipId AND
            BSt.Isbn = B.Isbn AND
            B.CopiesInStock >= BSt.Quantity
```

We could delete a row from the ShipperView in many ways by operating on the base tables. For example, to implement

DELETE FROM ShipperView WHERE ShipID = G

we could delete a row from Shipment where ShipID = G, or we could delete a row from BookShipment where ShipID = G.

But in this case to delete a row from the ShipperView as indicated, we update Shipment Date of Shipment.ShipDate by making non-NULL (suppose CurrDate represents the current date)



```
CREATE VIEW ShipperView (Name, Addr, ShipId, Isbn, Quantity)
AS SELECT C.CustName, A.ShippingAddr, BSt.ShipId, BSt.Isbn, BSt.Quantity
      FROM Customer C, Account A, Transaction T,
                Shipment St, BookShipment BSt, Book B
      WHERE C.CustEmailAddr = A.CustEmailAddr AND A.CustEmailAddr = T.CustEmailAddr AND
                A.CreditCardNumber = T.CreditCardNumber AND T.PaymentClearanceDate IS NOT NULL AND
                T.TransNumber = St.TransNumber AND St.ShipDate IS NULL AND St.ShipId = BSt.ShipId AND
                BSt.Isbn = B.Isbn AND B.CopiesInStock >= BSt.Quantity
```

Write an INSTEAD OF TRIGGER that implements

DELETE FROM ShipperView WHERE ShipID = 'X'

using policy

Update Shipment.ShipDate by making non-NULL

A reference to a row of
The view that is deleted

```
CREATE TRIGGER DeleteFromShipperView
INSTEAD OF DELETE ON ShipperView
/* FOR EACH ROW */
BEGIN
  UPDATE Shipment SET ShipDate = CurrDate WHERE ShipID = Old.ShipID;
END;
```

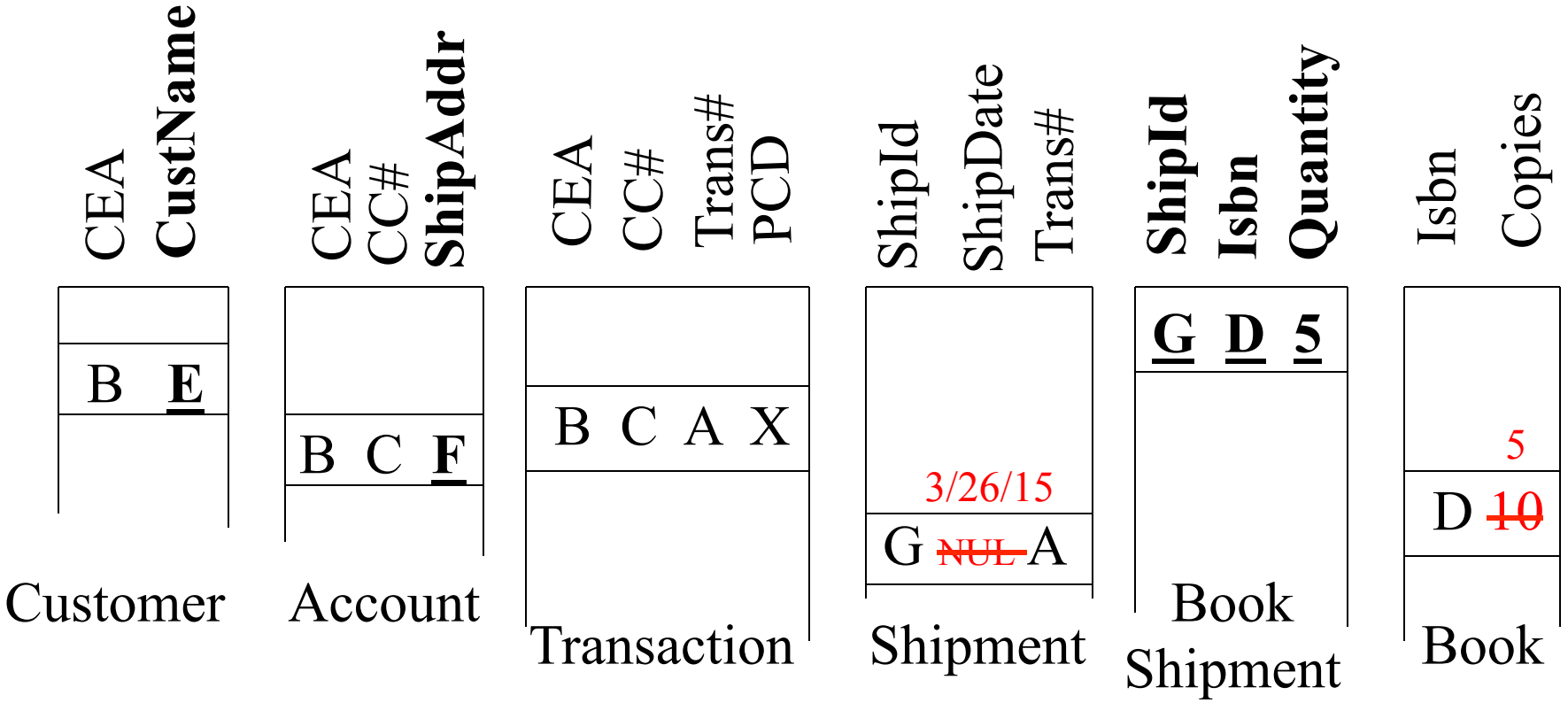
But we also want to update Book copies in stock

```
CREATE VIEW ShipperView (Name, Addr, ShipId, Isbn, Quantity)
AS SELECT C.CustName, A.ShippingAddr, BSt.ShipId, BSt.Isbn, BSt.Quantity
      FROM Customer C, Account A, Transaction T,
                Shipment St, BookShipment BSt, Book B
      WHERE C.CustEmailAddr = A.CustEmailAddr AND A.CustEmailAddr = T.CustEmailAddr AND
                A.CreditCardNumber = T.CreditCardNumber AND T.PaymentClearanceDate IS NOT NULL AND
                T.TransNumber = St.TransNumber AND St.ShipDate IS NULL AND St.ShipId = BSt.ShipId AND
                BSt.Isbn = B.Isbn AND B.CopiesInStock >= BSt.Quantity
```

Write an INSTEAD OF TRIGGER that implements
DELETE FROM ShipperView WHERE ShipID = 'X'

```
CREATE TRIGGER DeleteFromShipperView
INSTEAD OF DELETE ON ShipperView
/* FOR EACH ROW */
BEGIN
  UPDATE Shipment SET ShipDate = CurrDate WHERE ShipID = Old.ShipID;
  UPDATE Book SET CopiesInStock = CopiesInStock – Old.Quantity
                WHERE Isbn = Old.Isbn;
END;
```

Results of previous page's trigger when deleting one row of the ShipperView



Note that when CopiesInStock (Copies for short) is decremented, it may fall below the Quantity in other BookShipments with the same Isbn that are in the view.

Question: Will any rows of the view that now violate the

B.CopiesInStock >= BSt.Quantity

constraint be removed from the ShipperView "automatically" or must something additionally be done ? Is the answer dependent on the SQL environment?

An aside. If we wanted to depend on automatic view updates only (without INSTEAD OF TRIGGERS) we could add single base-table views, to which we can do updates (including deletes and inserts). For example, consider these two additional views

```
CREATE VIEW ShipperView2 (Isbn, Copies)
AS SELECT B.Isbn, B.CopiesInStock FROM Book B

CREATE ShipperView3 (ShipId, ShipDate)
AS SELECT St.ShipId, St.ShipDate FROM Shipment St
```

And to update, use some form of embedded SQL

```
FOR each tuple, t, of ShipperView DO
      <physical aspects of book processing such as printing
        labels, getting and packaging books, etc>

      UPDATE ShipperView2 S2
      SET S2.CopiesInStock = S2.CopiesInStock – t.Quantity
      WHERE S2.Isbn = t.Isbn

      UPDATE ShipperView3 S3
      SET S3.ShipDate = CurrentDate
      WHERE S3.ShipId = t.ShipId
```

We can also define views in terms of other views:

CREATE HandlerView (Isbn, Quantity)
AS SELECT S.Ibn, S.BookQuantity From ShipperView

And remember that views can hide information, so that we can grant access privileges to users for a view (e.g., the ShipperView) but NOT for some of the base tables that are used in the view's definition (e.g., Transaction).

We can grant various kinds of privileges to base tables and views to specified users.

GRANT and REVOKE (privilege) commands.

GRANT <SELECT, INSERT, DELETE, UPDATE>
ON <table or view> TO <ids> (with GRANT OPTION)

REVOKE <GRANT option for> PRIVILEGES
ON <table or view> FROM <ids> {RESTRICT | CASCADE}

For those doing data diaries (or anyone), you may want to do the Widom Authorization videos from DB12 (listed under Optional Material on the Schedule.