Thought exercises on relating application layer to DB transactions. Consider the following table definitions from our earlier Books DB design (by a bookseller named RockyBooks), with a new ShopCart table added:

```
CREATE TABLE ShopCart (
   WebId  CHAR[120],
   Quantity  INTEGER,
   PlacementDate  DATE NOT NULL,
   Isbn  INTEGER,
   PRIMARY KEY (WebId, Isbn),
   FOREIGN KEY (Isbn) REFERENCES Books
      ON DELETE NO ACTION, ON UPDATE CASCADE,
   CHECK (Quantity > 0)
)

CREATE TABLE Transactions (  /* Transactions and PlacedBy */
  TransNumber   INTEGER,
  OrderDate   DATE,
  PaymentClearanceDate  DATE,       /* if NULL, then payment has not cleared */
  CustEmailAddr    CHAR[120] NOT NULL,
  CreditCardNo   INTEGER NOT NULL,
  PRIMARY KEY (TransNum),
  FOREIGN KEY (CreditCardNo, CustEmailAddr) REFERENCES Accounts
    ON DELETE NO ACTION, ON UPDATE CASCADE,
  )

CREATE ASSERTION TransactionsShippedConstraint  /* insure participation constraint on Trans in */
CHECK (NOT EXISTS (SELECT *                          /* Shipped */
                   FROM Transactions T
                   WHERE T.TransNumber NOT IN (SELECT S.TransNumber
                                               FROM Shipped S)))

CREATE TABLE Shipment (
  ShipId INTEGER,
  ShipCost CURRENCY,
  ShipDate DATE,                  /* if this is NULL, then not shipped yet */
  TransNumber INTEGER NOT NULL,
  PRIMARY KEY (ShipId)
  FOREIGN KEY (TransNumber) REFERENCES Transaction
       ON DELETE CASCADE, ON UPDATE CASCADE
)

CREATE TABLE BookShipment (
   Quantity         INTEGER
   ShipId           INTEGER,
   Isbn             INTEGER,
   PRIMARY KEY (TransNumber, ShipId, Isbn),
   FOREIGN KEY (Isbn) REFERENCES Books
     ON DELETE NO ACTION, ON UPDATE CASCADE,
   FOREIGN KEY (ShipId) REFERENCES Shipment
     ON DELETE CASCADE, ON UPDATE CASCADE,
)
```

Suppose ShopCart includes the following rows (WebId, Quantity, PlacementDate, Isbn):

```
(WIabc, 2, 2/4/06, 123)
(WIabc, 1, 1/29/06, 234)
(WIabc, 3, 1/15,06, 345)
```

At the point that a customer purchases the contents of a "shopping cart" from an account, we would expect a number of operations would occur. Assume that a program in some high-level language collects relevant Web inputs (for example), and otherwise coordinates all the activities in the Shopping-Cart-Purchasing script, accessing the database as necessary through embedded SQL or some other interface mechanism.

A high-level description of the operations to be performed upon purchase of a "shopping cart" (identified as WebId=WIabc) follow:

1) A row will be inserted into the Transactions table with the appropriate CustEmailAddr (call it CEA), CreditCardNo (call it CCN), PaymentClearanceDate of NULL, OrderDate equaling the current date (call it OD), and a unique, generated TransNumber (call it TN). I would write this insert as

```
INSERT INTO Transactions VALUES (TN, OD, NULL, CEA, CCN)
```

The INSERT above is a parameterized statement. Again, the values of TN, OD, CEA, CCN will be provided by the purchasing script.

Also note that PaymentClearanceDate is set to NULL, and presumably it will be set to non-NULL later after the credit card is cleared, but I suspect that RockyBooks issues a request for clearance at this point (as part of the script), so that it could immediately enter a non-NULL PaymentClearanceDate. If the clearance system is down, then perhaps RockyBooks enters an initial NULL value, and proceeds (later issuing an email correction to the user if the transaction fails, or perhaps it refuses the transaction). Lets not worry about it for this exercise.

2) Each row in the ShopCart table that has a WebId attribute equaling WIabc (i.e., the WebId of the customer-accessed shopping cart) will be examined. These rows can be found by

```
SELECT S.Isbn, S.Quantity FROM ShopCart S WHERE S.WebId = WIabc
```

Note that we only care about two of the attributes per row.

3) Generate a unique ShipId (relative to the Shipment table), SI

4) For each row (I, Q), corresponding to values of (Isbn, Quantity), returned by query of (2), insert a corresponding row into BookShipment

```
INSERT INTO BookShipment VALUES (Q, SI, I)
```

where Q, SI, I are as described earlier.

5) Interweaved with step 4, compute the purchase cost of books, by summing over the quantity of each book times its price. Thus, for each (I, Q) pair, do

```
SELECT B.RockyBooksPrice FROM Books B WHERE B.Isbn = I
```

and multiply the single price that is returned (Isbn is key for Books) by Q, keeping a running sum of these products. You will note that there is no PurchasePrice field of Transactions. There probably should be, so that we can store the resulting total purchase price. Alternatively, we can let

the SQL interpreter do the work and in place of the two SELECTS in (2) and (5) we could

```
(SELECT S.Isbn, S.Quantity , B.RockyBooksPrice
 FROM ShopCart S, Books B
 WHERE S.WebId = WI AND S.Isbn = B.Isbn) AS Temp

SELECT SUM(Quantity * Price) FROM Temp
```

From `Temp` the program would get the (I, Q) pairs for purposes of step (4)

In addition, we need to compute the sum of projected shipping cost (SC), again over (I, Q) pairs (though the current table `Books` doesn't include attributes necessary for a very sophisticated shipping cost calculation (e.g., no `Weight` field). Given the tables as provided, something like the following would have to suffice:

```
SELECT SUM(1.25 * Quantity) AS ShippingCost FROM Temp
```

where I have chosen to assume that it costs a $1.25 to ship each book, though this is clearly naïve. Nonetheless, the important thing for you to remember with respect to part (B) is NOT that I/you encode UPS's shipping charges accurately, but that (1) I/you identify that the current attributes are insufficient for a non-trivial estimate of shipping charges (which we will remember for final design), and (2) that I/you take a stab at providing an SQL-based estimate of the charges (for purposes of practice, if nothing else) in fleshing out this script.

6) After steps 4 and 5 are complete,

```
INSERT INTO Shipment VALUES (SI, SC, NULL, TN)
```

7) Delete the ShopCart row that represents the purchase

```
DELETE FROM ShopCart S WHERE S.WebId = WIabc
```

It is debatable as to whether I would delete the ShopCart row at this point, or wait until the `PaymentClearanceDate` becomes non-NULL (though see point step 1 discussion).

8) Other steps that we haven't previously addressed, nor introduced tables for … for example, update a "`CoBought`" relation appropriately

```
UPDATE CoBought CB
SET CB.Copurchased = CB.Copurchased + 1
WHERE (CB.Book1 = I AND
        (CB.Book2 IN (SELECT S.Isbn
                      FROM Transactions T, Shipped S
                      WHERE T.CustEmailAddr = CEA AND
                            T.TransNumber = S.TransNumber AND
                            T.PaymentClearenceDate IS NOT NULL))
      OR
      (CB.Book2 = I AND
       (CB.Book1 IN (SELECT S.Isbn
                      FROM Transactions T, Shipped S
                      WHERE T.CustEmailAddr = CEA AND
                            T.TransNumber = S.TransNumber AND
                            T.PaymentClearenceDate IS NOT NULL))
```

For each `Isbn`, I, that is being purchased, increment the a Copurchased count of any/all

Books that are paired with I in `CoBought` and that was bought by the same customer (with email CEA). Such a `CoBought` relation might be used by a recommender system ("People who bought what you just bought, also bought these books …")

See discussion of `PaymentClearanceDate` under step (1).

If this transaction represents the first pairing of two books, then the script would have to do an insert into `CoBought`.

Suppose that a customer bought book X, followed on another day by book Y, followed on another day by book X, then I don't want to "double count" this co-bought behavior by the same customer. I could avoid double counting, but it probably isn't worth the trouble.

Finally, I doubt that RockyBooks updates a CoBought-like table after every purchase. Rather, any CoBought like table that they maintain is probably updated *en masse* at certain intervals (daily?). I wonder if a sorted file by `OrderDate` of `Transactions` would help efficiency here???

At the end of all this, we COMMIT to these changes, and integrity constraints are checked.

Think through the logic of various "scripts" and identify the basic SQL statements that are important for these scripts to interact with the database. For example, you might think through what would have to logically happen if RockyBooks could not ship all books of a current transaction at once (and in a timely manner), and what would be logically involved in RockyBooks determining that this was the case. The former would likely involve "splitting" the single `Shipment` row that we initially entered at purchase time, into multiple Shipment rows (but all associated with the same `TransactionNumber`, TN. In fact, the motivation for `Shipment` table and `Shipment` as well as a `Transactions` table was so that the DB could easily record that various shipments might be for books purchased on the same transaction, something that RockyBooks would likely want to do.

The steps above represent a *serial schedule*.

1) Identify any steps missing from this serial schedule that should occur up through what is now Step 8.

2) Identify constraints that should be checked in DEFERRED mode.

3) Flesh out the script necessary to check whether shipments need be split as described in the last paragraph.

4) Suppose the serial schedule as written above (excluding Step 8) represent one big transaction, T1. Suppose another transaction, T2, corresponds to the purchase of a second "shopping cart" (WI = WIbcd). Sketch out a serializable schedule that interleaves execution of T1 and T2. Include annotation for shared and exclusive locks. Identify and annotate the schedule with **savepoint**s.

5) Identify the likely **hot spot**s in the database that would emerge in the face of many Shopping-cart purchasing" transactions like T1 and T2. Consider likely indices for various tables in your identification of likely hot spots.

6) Does your identification of hot spots motivate any database redesign?

7) Can the Shopping-Cart Purchasing script be reasonably broken up into **nested transactions**, or into different transactions entirely?