Suppose that we added an additional table to the Books DB:

```
CREATE TABLE CoBought (
   Book1    INTEGER,
   Book2    INTEGER,
   Copurchased#   INTEGER,
   PRIMARY KEY (Book1, Book2),
   FOREIGN KEY (Book1) REFERENCES Book (Isbn)
   FOREIGN KEY (Book2) REFERENCES Book (Isbn)
   CHECK (Book1 < Book2)
)
```

The CoBought relation lists the total number of times that each pair of books (pair of Isbns) has been bought by a same customer, though not necessarily (or typically) on the same transaction. Write a query that lists the top 100 co-purchased pairs of books (i.e., list the pairs of Isbns). Also list the number of copurchases for each pair. Thus, your query should produce a table of three fields

(Book1, Book2, Copurchased#).

Your result may actually be greater than 100 rows in the case where there are ties between pairs of co-purchases. For example, if we were interested in the top 5, then we might get a result like this

Book1	Book2	Copurchased#
Abc	cde	10230
Der	fgc	10230
Dve	plu	10195
Wqi	zpf	10180
Tgh	uvw	10074
Ghj	mnv	10074

This would be a start, but incomplete:

```
SELECT C.Book1, C.Book2, C.Copurchased#
FROM CoBought C
ORDER BY C.CoPurchased# DESC
```

The query below retains all rows of CoBought that have fewer than 100 other rows ranked above them. This query does not rely on GROUP BY, and my guess is that even if the nested SELECT evaluates to the empty relation, that 0 will be returned (whereas an empty GROUP would not be considered at all).

```
SELECT C.Book1, C.Book2, C.Copurchased#
FROM CoBought C
WHERE 100 > (SELECT COUNT (*)
             FROM CoBought C2
             WHERE C.Copurchased# < C2.Copurchased#)
ORDER BY C.Copurchased# DESC
```

If you wanted to label the ranks of each row as well, consider:

```
SELECT 1 AS RowNum, C.Book1, C.Book2, C.Copurchased
FROM CoBought C
WHERE C.Copurchased# = (SELECT MAX(Copurchased#) FROM CoBought
UNION
SELECT Temp.RowNum, Temp.Book1, Temp.Book2, Temp.Copurchased#
FROM (SELECT COUNT(*)+1 AS RowNum, C1.Book1, C1.Book2, C1.Copurchased#
        FROM CoBought C1, CoBought C2
        WHERE C1.Copurchased# < Temp2.Copurchased#
        GROUP BY C1.Book1, C1.Book2, C1.Copurchased#) AS Temp
 WHERE Temp.RowNum <= 100
```

I could have written this as well – note that the top-level SELECT (and its WHERE clause) after UNION is gone, and has been replaced by a HAVING clause.

```
SELECT 1 AS RowNum, C.Book1, C.Book2, C.Copurchased
FROM CoBought C
WHERE C.Copurchased# = (SELECT MAX(Copurchased#) FROM CoBought
UNION
SELECT COUNT(*)+1 AS RowNum, C1.Book1, C1.Book2, C1.Copurchased#
FROM CoBought C1, CoBought C2
WHERE C1.Copurchased# < C2.Copurchased#
GROUP BY C1.Book1, C1.Book2, C1.Copurchased#
HAVING  COUNT(*)+1 <= 100
```

The queries above will return possibly more than 100 rows, as the example below illustrates. In particular, if the 100 row participates in a tie, then all tied cobought books will be included too.