Written Assignment #4
Due Wednesday May 13

1. Do the following problems from the textbook. You only need to write the queries and updates in SQL, you do *not* need to evaluate the queries or updates over the sample data. Write your queries so that the answers will never include duplicates, but add the keyword `distinct` *only* in cases where duplicates would otherwise be produced.

   (i) Exercise 5.2.2 (page 262) parts (a)–(d)

   (ii) Exercise 5.3.1 (page 269) parts (a)–(c)

   (iii) Exercise 5.5.1 (pages 277–278) parts (b)–(e)

   (iv) Exercise 5.6.1 (pages 284–285) parts (b), (d), (g)

2. Suppose we wish to issue the following SQL query:

```
select * from R
where (A,B) in (select A,B from S)
```

   but unfortunately the restricted DBMS we are using does not support multi-attribute `in` conditions. Rewrite the above query into an equivalent one that uses a single-attribute `in` condition.

3. Consider the following query that uses the `intersect` operator.

   (`select` $R_1.A$ `from` R_1 `where` $cond_1$)
   `intersect`
   (`select` $R_2.A$ `from` R_2 `where` $cond_2$)

   Give an equivalent query that does not use the `intersect` operator.

4. Consider a relation `Flights(from,to,cost,airline)` containing nonstop flights from one city to another. Note that the flights from city A to city B are independent of the flights from B to A. For example:

from	to	cost	airline
SF	Denver	300	Frontier
SF	Denver	350	United
Denver	SF	250	United
Denver	SF	250	Frontier
Denver	Chicago	250	American
Chicago	NY	250	Delta
Denver	NY	500	American
Denver	NY	400	TWA
SF	NY	750	United

(a) Give a single SQL query that returns the cost of the cheapest nonstop flight between each pair of cities. For example, the result over the above relation instance should be:

from	to	cost
SF	Denver	300
Denver	SF	250
Denver	Chicago	250
Chicago	NY	250
Denver	NY	400
SF	NY	750

(b) Give a single SQL query that returns the cheapest cost of flying between each pair of cities assuming we are willing to stop up to two times en route. For example, by stopping once (in Denver), we can get from SF to NY for 700 instead of 750. In this example, we could stop twice (in Denver and Chicago), but that would be more expensive ($300 + 250 + 250 = 800$).

(c) Is it possible to write a single SQL query that returns the cheapest cost of flying between each pair of cities regardless of the number of stops? If so, give the query. If not, briefly explain why.

5. Consider a relation `SalesReps(name,region,salary)`. Assume that no two sales-reps in the same region have the same salary.

(a) Give a single query that returns each region along with the top salary in that region.

(b) Give a sequence of SQL statements, ending with a query that returns the top two salaries for each region. The result of your final query should be an ordered two-column relation, where each region is represented by two adjacent tuples, with the higher salary for each region listed first. If a region only has one sales-rep, your result should still include a single tuple for that region. For example, we might see the following result, where the "South" region has only one sales-rep:

region	salary
East	40,000
East	39,000
North	100,000
North	80,000
South	25,000
West	30,000
West	20,000

Hint: In your sequence of SQL statements you may want to create and query additional relations.

6. In Oracle and other database systems you can create a *sequence*, which can be used in the `select` clause of SQL queries as a generator of integers. For example, if relation R has three tuples then the commands:

```
create sequence seq;
select seq.nextval as num from R
```

will return the result:

num
1
2
3

If the `select` statement is issued again we get:

num
4
5
6

Consider a relation `Sales(productID,price,store,date)`. Suppose we wish to retrieve a small sample of the data in relation `Sales` for analysis. Assume that the sales data is evenly distributed and that the tuples are stored randomly in the relation. Suppose a sequence `seq` has been initialized as above. Give a single SQL query that returns at least 1% but fewer than 1.1% of the tuples in the `Sales` relation. You may assume that `Sales` has at least a million tuples, but you do not know in advance the exact number. Note that some (but not all) solutions to this problem require modulo arithmetic. SQL does support the `mod` operator: `mod`(m,n) returns the remainder when m is divided by n.