The assignment is to be turned in before Midnight (by 11:59pm) on February 8, 2018. You should turn in the solutions to this assignment as a pdf file through the TEACH website. The solutions should be produced using editing software programs, such as LaTeX or Word, otherwise they will not be graded.

1: Query optimization (2 points)

Consider the following relations:

Product (name, production-year, rating, company-name)
Company (name, state, employee-num)

Assume each product is produced by just one company, whose name is mentioned in the company-name attribute of the Product relation. Attributes name are the primary key for both relations Product and Company. Attribute rating shows how popular a product is and its values are between 1-5.

The following query returns the products with rating of 5 that are produced after 2000 and the states of their companies.

```
SELECT p.name, c.state
FROM Product p, Company c
WHERE p.company-name = c.name and p.production-year > 2000 and p.rating = 5
```

Suggest an optimized logical query plan for the above query.

**Solution:**
The optimized logical query plan for the above query is the following. The plan first selects table Product according to the conditions on production-year and rating. It then projects Product on company-name and name. It also projects table Company on attributes name, state. Finally, it joins the projected Product and Company relations and projects the resulting table on attribute name.

2: Query optimization (2 point)

For the four base relations in the following table, find the best join order according to the dynamic programming algorithm used in System-R. You should give the dynamic programming table entries for evaluate the join orders. Suppose that we are only interested in left-deep join trees and join trees without Cartesian products. Note that you must use the Selinger-style formulas to compute the size of each join output. The database system uses hash join to compute every join. Assume that there is enough main memory to perform the hash join for every pairs of relations. Each block contains at most 5 tuples of a base or joint relation. If relations are joined on multiple attributes, you can compute the selectivity factor of the full join by multiplying the selectivity factors of joining on each attribute.
### Solution:

<table>
<thead>
<tr>
<th>Query</th>
<th>Size</th>
<th>Cost</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>R,S</td>
<td>200</td>
<td>4,200</td>
<td>SR</td>
</tr>
<tr>
<td>R,W</td>
<td>40,000</td>
<td>3,600</td>
<td>WR</td>
</tr>
<tr>
<td>R,U</td>
<td>40,000</td>
<td>3,000</td>
<td>UR</td>
</tr>
<tr>
<td>S,W</td>
<td>60,000</td>
<td>3,000</td>
<td>WS</td>
</tr>
<tr>
<td>S,U</td>
<td>IGNORE</td>
<td>IGNORE</td>
<td>IGNORE</td>
</tr>
<tr>
<td>W,U</td>
<td>20,000</td>
<td>1,800</td>
<td>UW</td>
</tr>
<tr>
<td>R,S,W</td>
<td>4,000</td>
<td>5,520</td>
<td>(SR)W</td>
</tr>
<tr>
<td>R,S,U</td>
<td>2,000</td>
<td>4,920</td>
<td>(SR)U</td>
</tr>
<tr>
<td>R,W,U</td>
<td>4,000</td>
<td>16,200</td>
<td>(UW)R</td>
</tr>
<tr>
<td>S,W,U</td>
<td>600,000</td>
<td>15,600</td>
<td>(UW)S</td>
</tr>
</tbody>
</table>

The fifth row should be ignored as it joins trees with Cartesian product.

We use the formula given in the lecture to compute the size of each join. If relations are joined on multiple attributes, we compute the selectivity of join by multiplying the selectivities of joining on each attribute. It is similar to the method of computing the selectivity factor of selections with multiple conditions given the lecture. For example, we compute the output size of join between R and S as following:

\[
\frac{4000 \times 3000}{\max(200, 100) \times \max(100, 300)} = \frac{12000000}{200 \times 300} = \frac{12000000}{60000} = 200
\]

To compute the size of join between three or more base relations, we should first calculate the number of distinct values for some attributes in the result of a join between two base relations. For example, we need to know the number of distinct values of attribute B in R ⋈ S to compute the size of (R ⋈ S) ⋈ W. Let us consider the case the attribute is used for the join of base relations, e.g., attribute B in R ⋈ S. Consider tuple r in R and tuple s in S. For r ⋈ s to be in R ⋈ S, the value of attribute B in r and s should match. Hence, the number of distinct values for B in the output relation is equal to the minimum of number of distinct values of B in R and S, which is equal to the number of distinct values of B in S. Now, let’s consider the case where we want to estimate the number of distinct values for an attribute that is not used to join the relations, e.g., attribute A in R ⋈ S. Because we assume that attributes are independent in query optimization, it is reasonable to assume that the number of distinct values for an attribute that is not used in a join remain unchanged after the join. For example, the number of distinct values for attribute A in R ⋈ B is equal to the number of distinct values of A in R.

We choose the expression ((SR)U)W for computing the join of these 4 relations.

### 3: Concurrency control (3 points)
Consider the following classes of schedules: serializable and 2PL. For each of the following schedules, state which of the preceding classes it belongs to. If you cannot decide whether a schedule belongs to a certain class based on the listed actions, explain briefly your reasons.

The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

1. T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)
2. T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)

Solution:

1. The serialization graph is T1 \rightarrow T3 \rightarrow T2. Since the serialization graph does not have any cycle, the schedule is serializable. It is also 2PL. One possible locking sequence for this schedule that is compatible with the 2PL rule is as follows. In the following schedule, SLock, SRelease, XLock, and XRelease denote acquiring a shared lock, releasing a shared lock, acquiring an exclusive lock, and releasing an exclusive lock, respectively.

   T1:SLock(X), T1:SLock(Y), T1:R(X), T1:SRelease(X),
   T2:SLock(Y), T2:R(Y),
   T3:XLock(X), T3:W(X), T3:XRelease(X),
   T2:SLock(X), T2:R(X), T2:SRelease(X),
   T2:SRelease(Y),
   T1:R(Y), T1:SRelease(Y)

2. The serialization graph is:

   ![Serialization Graph]

   Since the serialization graph has a cycle between T2 and T3, the schedule is **not** serializable. Because the schedule is not serializable, it is not a 2PL schedule.