Sparsh Arora
UID: 804653078

CS143 – HW3

PART 1

```
SELECT company,
SUM(value='agile-dev') AS AgileDev,
SUM(value='benefit-company') AS BenefitCompany,
SUM(value='bonded-by-product') AS BondedByProduct,
SUM(value='continuous-delivery') AS ContinuousDelivery,
SUM(value='creative-innovation') AS CreativeInnovation,
SUM(value='cross-dep') AS CrossDep,
SUM(value='customer-first') AS CustomerFirst,
SUM(value='data-driven') AS DataDriven,
SUM(value='diverse-team') AS DiverseTeam,
SUM(value='engages-community') AS EngagesCommunity,
SUM(value='engineering-driven') AS EngineeringDriven,
SUM(value='eq-iq') AS EqIq,
SUM(value='fast-paced') AS FastPaced,
SUM(value='feedback') AS Feedback,
SUM(value='flat-organization') AS FlatOrganization,
SUM(value='flex-hours') AS FlexHours,
SUM(value='friends-outside-work') AS FriendsOutsideWork,
SUM(value='good-beer') AS GoodBeer,
SUM(value='impressive-teammates') AS ImpressiveTeammates,
SUM(value='inclusive') AS Inclusive,
SUM(value='internal-mobility') AS InternalMobility,
SUM(value='internal-promotion') AS InternalPromotion,
SUM(value='interns') AS Interns,
SUM(value='junior-devs') AS JuniorDevs,
SUM(value='light-meetings') AS LightMeetings,
SUM(value='lunch-together') AS LunchTogether,
SUM(value='many-hats') AS ManyHats,
SUM(value='new-tech') AS NewTech,
SUM(value='office-layout') AS OfficeLayout,
SUM(value='open-communication') AS OpenCommunication,
SUM(value='open-source') AS OpenSource,
SUM(value='pair-programs') AS PairPrograms,
SUM(value='parents') AS Parents,
SUM(value='personal-growth') AS PersonalGrowth,
SUM(value='physical-wellness') AS PhysicalWellness,
SUM(value='product-driven') AS ProductDriven,
SUM(value='project-ownership') AS ProjectOwnership,
SUM(value='psychologically-safe') AS PsychologicallySafe,
SUM(value='quality-code') AS QualityCode,
SUM(value='rapid-growth') AS RapidGrowth,
SUM(value='remote-ok') AS RemoteOk,
SUM(value='retention') AS Retention,
SUM(value='risk-taking') AS RiskTaking,
SUM(value='safe-env') AS SafeEnv,
SUM(value='team-oriented') AS TeamOriented,
SUM(value='worklife-balance') AS WorklifeBalance
FROM keyvalues
GROUP BY company;
```

Number of rows = 67
Number of columns = 47

PART 2

1. **NO Subquery with DISTINCT**

```
id      select_type     table   partitions      type    possible_keys key
        key_len         ref     rows    filtered        Extra
1       SIMPLE          caltrans        NULL    ALL     PRIMARY       NULL
        NULL    NULL    103449          20.99   Using where; Using temporary;
Using filesort
```

2. **SELECT within a SELECT**

```
id      select_type     table   partitions      type    possible_keys key
        key_len         ref     rows    filtered        Extra
1       PRIMARY         <derived2>      NULL    ALL     NULL    NULL  NULL
        NULL    2411    100.00Using temporary; Using filesort
2       DERIVED         caltrans        NULL    ALL     PRIMARY       NULL
        NULL    NULL    103449          2.33    Using where; Using temporary;
Using filesort
```

3. **JOIN as a FILTER**

```
id      select_type     table   partitions      type    possible_keys key
        key_len         ref     rows    filtered        Extra
1       PRIMARY         <derived2>      NULL    ALL     NULL    NULL  NULL
        NULL    24129 100.00Using temporary; Using filesort
2       DERIVED         c       NULL    ALL     PRIMARY       NULL    NULL
        NULL    103449          2.33    Using where; Using temporary; Using
filesort
2       DERIVED         <derived3>      NULL    ref     <auto_key0>
        <auto_key0>     8       hw2.c.highway           10      100.00NULL
3       DERIVED         caltrans        NULL    ALL     PRIMARY       NULL
        NULL    NULL    103449          2.33    Using where; Using temporary
```

4. **USING an IN subquery as a Filter**

```
id      select_type     table   partitions      type    possible_keys key
        key_len         ref     rows    filtered        Extra
```

```
1       PRIMARY      <derived2>    NULL  ALL     NULL   NULL   NULL
        NULL  2411   100.00Using temporary; Using filesort
2       DERIVED      c        NULL  ALL    PRIMARY       NULL   NULL
        NULL  103449        2.33    Using where; Using temporary; Using
filesort
2       DERIVED      <subquery3> NULL  eq_ref <auto_key>
        <auto_key>   265     hw2.c.highway,hw2.c.area  1        100.00
        NULL
3       MATERIALIZED        caltrans         NULL  ALL     NULL   NULL
        NULL  NULL  103449          2.33    Using where
```

5. **FORMAL Left Semijoin**

```
id      select_type     table    partitions       type    possible_keys key
        key_len         ref      rows   filtered          Extra
1       PRIMARY      <derived2>    NULL  ALL     NULL   NULL   NULL
        NULL  2411   100.00Using temporary; Using filesort
2       DERIVED      c        NULL  ALL    PRIMARY       NULL   NULL
        NULL  103449        2.33    Using where; Using temporary; Using
filesort
3       SUBQUERY     caltrans         NULL  ALL     NULL   NULL   NULL
        NULL  103449        2.33    Using where
```

From the outputs we notice that query 1 included no sub queries or derived tables. Query 2 has 1 derived table, Query 3 has 3 derived tables, Query 4 has 2 derived tables and Query 5 has 1 sub query and 1 derived table. The column type in the above outputs represents the type of join. In the case of MySQL this is always ALL to represent a Cartesian join. The column rows filtered shows us the percentage of rows it actually filtered. The ref column shows the number of rows to actually filter. The last extra column represents extra information about the query for instance 'using where' means that the query had a where clause, 'filesort' refers to some kind of sort, 'temporary' refers to having a temporary table etc. We notice that Query 3 has a large number of rows it filters and also has temporary tables. These not only take up more memory but also take longer and thus are more inefficient. Similarly, Query 4 filters many rows as well. We see that Query 2 is pretty efficient (more than Query 1) because it uses its derived table to filter rows quicker and is thus more efficient. This is the same for Query 5.

PART 3

1.
R is scanned 100,000 times because it goes through the loop each time.

2.
a) Each tuple in L and R is scanned once.
b)
in the worst case, number of block transfers would be

number of blocks in L + (height of the B+ tree of index R+!) * (number of tuples in L)

3.
We prefer Block Nested Loop Join if the outer table contains fewer blocks in memory.
We prefer the Naïve Nested loop Join in conditions where the block size is almost equal to the tuple size. This is also preferred when no index is used on the join.
We prefer the Indexed Nested Loop Join when R is indexed accurately and the length is small so the time it takes is less.

4.

If a is a tuple in R1 and b is a tuple in R2. But since we have R2-R3, b is also a tuple in R3.

Thus tuple x will show up in R1 join (R2-R3) if a is in R1, b is in R2 and there is a b isn't in R3.
The tuple x again will show up in (R1 join R2) – (R2 join R3) if a is in R1 and b is in R2 and NOT (a is in R1 and b is in R3)

But as we know from above, a is in R1, we can say that the above two conditions are the same.
Thus this equation holds true.

This can help make the joins more efficient because we can reduce the number of fields before doing the join (which is tedious).

5.

Consider two relations R and S as shown below:

R:

A	B
1	2
2	3
3	4

S:

A	B
1	6
2	3

7	8

For these set of relations,

$\pi_A(R - S) =$

A
1
3

But, $\pi_A(R) - \pi_A(S) =$

A
3

In this instance, the equation doesn’t hold.