Part 2

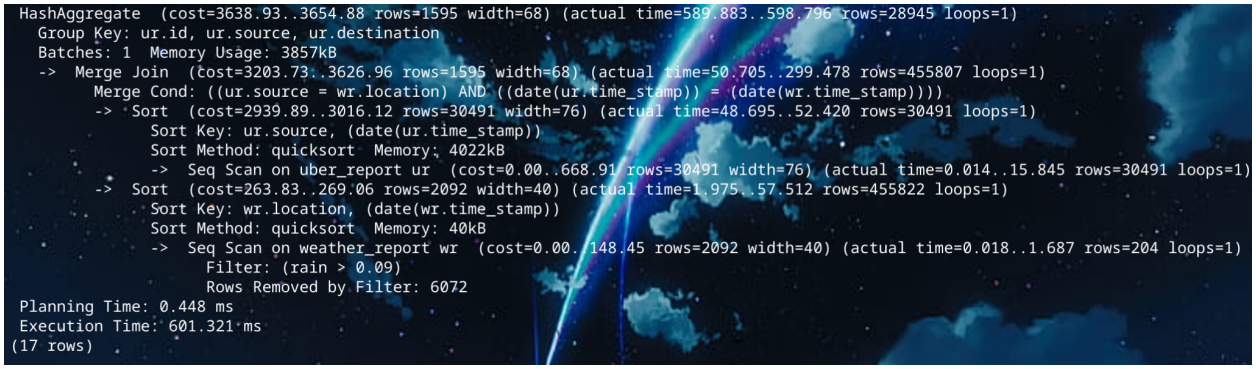
Query:

```
SELECT DISTINCT id, source, destination
FROM uber_report ur, weather_report wr
WHERE source = location AND rain > 0.09
 AND date(wr.time_stamp) = date(ur.time_stamp);
```

Part 2.1:

$\pi_{id,\ source,\ destination}\ (\sigma_{rain\ >\ 0.09}\ ((\rho_{ur}\ (uber\_report))\ \bowtie_{date(wr.time\_stamp)\ =\ date(ur.time\_stamp)\ AND\ source\ =\ location}\ (\rho_{wr}\ (weather\_report))))$

Part 2.2:



a. What happened to the selection/filter expression?
   i. The selection/filter expression is only filtering rows that adhere to this condition: rain > 0.09. The other conditions in the WHERE clause, source = location and date(wr.time_stamp) = date(ur.time_stamp), are not used in the filtering process.
b. Were either of the primary key indexes used? In what operations?
   i. All of the primary key indexes from both tables were used. The primary key from the uber_report table was used in the Group Key (ur.id). The primary keys from the weather report table was used in the Merge Conditions as well as the respective sort keys.
c. How was the cross product altered? Be sure to discuss the order of tables, the selected filter(s), and the join algorithm chosen.
   i. The cross product was altered as the other predicates in the WHERE clase were turned into conditions for the join statement. The other predicate, as mentioned above, was used as the filter. The join algorithm chosen is a merge join.
d. How did it ensure the resulting rows were unique? (what algorithm was chosen)

i. The merge join ensured the resulting rows were unique as it processes matching rows simultaneously. The merge join is also significantly faster than any of the other join methods because of the index (which means that this is a conjunctive algorithm)..

Part 2.3:

```
                                                      QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------>
 HashAggregate  (cost=3631.12..3647.07 rows=1595 width=68) (actual time=309.524..314.344 rows=28945 loops=1)
   Group Key: ur.id, ur.source, ur.destination
   Batches: 1  Memory Usage: 3857kB
   ->  Merge Join  (cost=3195.92..3619.16 rows=1595 width=68) (actual time=27.121..157.678 rows=455807 loops=1)
         Merge Cond: ((ur.source = wr.location) AND ((date(ur.time_stamp)) = (date(wr.time_stamp))))
         ->  Sort  (cost=2939.89..3016.12 rows=30491 width=76) (actual time=26.804..28.820 rows=30491 loops=1)
               Sort Key: ur.source, (date(ur.time_stamp))
               Sort Method: quicksort  Memory: 4022kB
               ->  Seq Scan on uber_report ur  (cost=0.00..668.91 rows=30491 width=76) (actual time=0.011..8.962 rows=30491 loops=1)
         ->  Sort  (cost=256.03..261.26 rows=2092 width=40) (actual time=0.297..29.176 rows=455822 loops=1)
               Sort Key: wr.location, (date(wr.time_stamp))
               Sort Method: quicksort  Memory: 40kB
               ->  Bitmap Heap Scan on weather_report wr  (cost=44.50..140.65 rows=2092 width=40) (actual time=0.070..0.148 rows=204 loops>
                     Recheck Cond: (rain > 0.09)
                     Heap Blocks: exact=25
                     ->  Bitmap Index Scan on rain_idx  (cost=0.00..43.97 rows=2092 width=0) (actual time=0.059..0.060 rows=204 loops=1)
                           Index Cond: (rain > 0.09)
 Planning Time: 0.289 ms
 Execution Time: 315.580 ms
(19 rows)
```

e. Why did you choose to create those indexes? What about the prior EXPLAIN result led you to choose them? point out concrete items in the first EXPLAIN result.
   i. I chose to create an index on the filter column because it will improve query performance and execution plan quality. My reasoning for this is because it is smaller than a full-table nonclustered index and has filtered statistics. The filtered statistics are more accurate than full-table statistics because they cover only the rows in the filtered index.

f. What changed vs. the earlier EXPLAIN ANALYZE? Be concrete and specific (join order/algorithm, other operations in the tree, execution time, or other changes you note) How are those changes related to your new index(es)?
   i. Planning and execution was reduced by almost half and this is due to the index on the rain column which is a filter condition.

Part 2.4:

```
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=1170.85..1247.08 rows=7623 width=68)
   Group Key: uber_report.id, uber_report.source, uber_report.destination
   ->  Hash Join  (cost=166.53..1113.67 rows=7623 width=68)
         Hash Cond: ((uber_report.source = weather_report.location) AND (date(uber_report.time_stamp) = date(weather_report.time_stamp)))
         ->  Seq Scan on uber_report  (cost=0.00..668.91 rows=30491 width=76)
         ->  Hash  (cost=157.28..157.28 rows=617 width=40)
               ->  HashAggregate  (cost=151.11..157.28 rows=617 width=40)
                     Group Key: weather_report.location, date(weather_report.time_stamp)
                     ->  Bitmap Heap Scan on weather_report  (cost=44.50..140.65 rows=2092 width=40)
                           Recheck Cond: (rain > 0.09)
                           ->  Bitmap Index Scan on rain_idx  (cost=0.00..43.97 rows=2092 width=0)
                                 Index Cond: (rain > 0.09)
(12 rows)
```

g. What changed vs. the earlier EXPLAIN ANALYZE? Be concrete and specific (join order/algorithm, other operations in the tree, ... ?) Do you view the rewrite useful or harmful? Explain why.

i. Instead of a merge join algorithm, it is now a hash join. Because it is a hash join, it has to filter through all the rows sequentially versus simultaneously which is why I would personally not use a subquery. Instead of being efficient, a subquery cannot simultaneously sort through the rows if there are multiple condition in the join.