Part 2.1

```
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);
```

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

Part 2.2

```
                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=3544.29..3554.43 rows=1014 width=32) (actual time=513.265..522.543 rows=28945 loops=1)
   Group Key: ur.id, ur.source, ur.destination
   Batches: 1  Memory Usage: 3857kB
   ->  Merge Join  (cost=3128.78..3536.68 rows=1014 width=32) (actual time=30.282..255.460 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=40) (actual time=28.773..33.955 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=40) (actual time=0.023..9.182 rows=30491 loops=1)
         ->  Sort  (cost=188.88..192.21 rows=1330 width=40) (actual time=1.480..51.065 rows=455822 loops=1)
               Sort Key: wr.location, (date(wr.time_stamp))
               Sort Method: quicksort  Memory: 40kB
               ->  Seq Scan on weather_report wr  (cost=0.00..119.88 rows=1330 width=40) (actual time=0.036..1.312 rows=204 loops=1)
                     Filter: (rain > 0.09)
                     Rows Removed by Filter: 6072
 Planning Time: 0.370 ms
 Execution Time: 524.849 ms
(17 rows)
```

a) What happened to the selection/filter expression?
The selection/filter expression source = location AND rain > 0.09 is still present in the query plan, but it is applied in the execution of the query rather than in the planning stage. Specifically, the Filter operation on the weather_report table in the query plan corresponds to the rain > 0.09 part of the selection expression, and the Merge Cond operation between the weather_report and uber_report tables correspond to the source = location part of the selection expression.

b) Were either of the primary key indexes used? In what operations?
For the weather_report table, the query plan shows a sequential scan (Seq Scan) operation, which means that PostgreSQL read every row in the table in sequence to filter out rows that do not satisfy the rain > 0.09 condition, so the time_stamp and location was used.

For the uber_report table, id was the primary key, but it was not used in the scan.

c) How was the cross product altered? Be sure to discuss the order of tables, the selected filter(s), and the join algorithm chosen.
The cross product between the uber_report and weather_report tables was altered through the use of a join algorithm that combined rows from both tables based on a common column, namely source in uber_report and location in weather_report. The order of the tables and the

selection filter(s) also affected the resulting join. The use of a merge join algorithm and the application of selection filter(s) can improve the performance of a query that involves joining large tables by reducing the number of rows that need to be processed.

d) How did it ensure the resulting rows were unique? (what algorithm was chosen)
The use of HashAggregate ensures that only unique rows with respect to the grouping columns (id, source, and destination) are returned by the query.

Part 2.3

```
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=3525.22..3551.14 rows=2592 width=32) (actual time=602.238..611.237 rows=28945 loops=1)
   Group Key: ur.id, ur.source, ur.destination
   Batches: 1  Memory Usage: 3857kB
   ->  Merge Join  (cost=3096.17..3505.78 rows=2592 width=32) (actual time=48.112..297.104 rows=455807 loops=1)
         Merge Cond: ((wr.location = ur.source) AND ((date(wr.time_stamp)) = (date(ur.time_stamp))))
         ->  Sort  (cost=156.28..156.79 rows=204 width=22) (actual time=1.963..2.016 rows=204 loops=1)
               Sort Key: wr.location, (date(wr.time_stamp))
               Sort Method: quicksort  Memory: 40kB
               ->  Seq Scan on weather_report wr  (cost=0.00..148.45 rows=204 width=22) (actual time=0.019..1.672 rows=204 loops=1)
                     Filter: (rain > 0.09)
                     Rows Removed by Filter: 6072
         ->  Sort  (cost=2939.89..3016.12 rows=30491 width=40) (actual time=46.112..106.028 rows=457338 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=40) (actual time=0.008..14.549 rows=30491 loops=1)
 Planning Time: 0.853 ms
 Execution Time: 613.958 ms
(17 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.
These indexes were the columns involved in the primary filtering and joining operations of the query, and the query plan showed that there were sequential scans being performed on both tables. By creating indexes on the location and time_stamp columns of the weather_report table and the source column of the uber_report table, we can speed up the query by reducing the amount of data that needs to be scanned and sorted during the filter and join operations.

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)?
The join order and algorithm have remained the same, as well as the other operations in the tree. However, the cost of the Merge Join operation has decreased from (cost=3096.17..3505.78 rows=2592 width=32) to (cost=2667.13..3076.68 rows=2592 width=32). This indicates that the indexes have been used to speed up the join operation. The index (location, time_stamp, rain) is being used to filter rows from the weather_report table where rain > 0.09 before joining with the uber_report table, which reduces the number of rows that need to be joined.

Part 2.4

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=1326.09..1478.55 rows=15246 width=32)
   Group Key: ur.id, ur.source, ur.destination
   ->  Hash Join  (cost=154.07..1211.75 rows=15246 width=32)
         Hash Cond: ((ur.source = wr.location) AND (date(ur.time_stamp) = date(wr.time_stamp)))
         ->  Seq Scan on uber_report ur  (cost=0.00..668.91 rows=30491 width=40)
         ->  Hash  (cost=151.31..151.31 rows=184 width=22)
               ->  HashAggregate  (cost=149.47..151.31 rows=184 width=22)
                     Group Key: wr.location, date(wr.time_stamp)
                     ->  Seq Scan on weather_report wr  (cost=0.00..148.45 rows=204 width=22)
                           Filter: (rain > 0.09)
(10 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.
The join algorithm has changed from a merge join to a hash join, and the cost of the query has decreased significantly. It also shows a hash join between the uber_report and weather_report tables, with a hash aggregate operation to group by the required columns. The rewrite using a subquery has likely helped to improve the performance of the query. It is likely more efficient than the original query because the subquery filters and aggregates data from the weather_report table before joining it to the uber_report table, reducing the number of rows that need to be joined.