Raymond Lin
304937942
CS143

Homework 4 Writeup

1.

Method 0: No subquery, using 'DISTINCT'

```
EXPLAIN
SELECT
        highway,
        area,
        COUNT(DISTINCT EXTRACT(DOY FROM reported)) * 100 / 365 AS percentage_of_days_closed_365,
        COUNT(DISTINCT EXTRACT(DOY FROM reported)) * 100 / 353 AS percentage_of_days_closed_353
FROM hw2.caltrans
WHERE condition LIKE '%CLOSED%DUE TO SNOW%' OR condition LIKE '%CLOSED%FOR THE WINTER%'
GROUP BY highway, area
ORDER BY percentage_of_days_closed_365 DESC;

                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=4461.84..4465.18 rows=1338 width=53)
   Sort Key: (((count(DISTINCT date_part('doy'::text, reported)) * 100) / 365)) DESC
   ->  GroupAggregate  (cost=4336.02..4392.36 rows=1338 width=53)
         Group Key: highway, area
         ->  Sort  (cost=4336.02..4340.25 rows=1690 width=45)
               Sort Key: highway, area
               ->  Seq Scan on caltrans  (cost=0.00..4245.41 rows=1690 width=45)
                     Filter: ((condition ~~ '%CLOSED%DUE TO SNOW%'::text) OR (condition ~~ '%CLOSED%FOR THE WINTER%'::text))
```

Method 1: using 'SELECT' within a 'SELECT'

```
EXPLAIN
SELECT
  highway,
  stretch,
  COUNT(1) AS days_closed,
  100 * COUNT(1) / 365 AS pct_closed_365,
  100 * COUNT(1) / 353 AS pct_closed_353
FROM (
  SELECT
    highway AS highway,
    area AS stretch,
    DATE(reported) AS closure
  FROM hw2.caltrans
  WHERE condition LIKE '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
  GROUP BY highway, stretch, closure
) result
GROUP BY highway, stretch
ORDER BY pct_closed_365 DESC;

                                              QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=4524.56..4524.95 rows=155 width=61)
   Sort Key: (((100 * count(1)) / 365)) DESC
   ->  GroupAggregate  (cost=4510.39..4518.93 rows=155 width=61)
         Group Key: caltrans.highway, caltrans.area
         ->  Group  (cost=4510.39..4512.34 rows=155 width=41)
               Group Key: caltrans.highway, caltrans.area, (date(caltrans.reported))
               ->  Sort  (cost=4510.39..4510.78 rows=156 width=41)
                     Sort Key: caltrans.highway, caltrans.area, (date(caltrans.reported))
                     ->  Seq Scan on caltrans  (cost=0.00..4504.71 rows=156 width=41)
                           Filter: ((condition ~~ '%CLOSED%'::text) AND ((condition ~~ '%FOR THE WINTER%'::text) OR (condition ~~ '%DUE
TO SNOW%'::text)))
```

```
  Method 2: using 'JOIN' as a filter
  EXPLAIN
  SELECT
    closures.highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
  FROM (
    SELECT
      c.highway AS highway,
      c.area AS stretch,
      DATE(c.reported) AS closure
    FROM hw2.caltrans c
  JOIN (
    SELECT
      DISTINCT highway,
      area
    FROM hw2.caltrans
    WHERE condition like '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO NOW%'))
snow_highways
  ON c.highway = snow_highways.highway
    WHERE condition like '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
    GROUP BY c.highway, stretch, closure
  ) closures
  GROUP BY closures.highway, closures.stretch
  ORDER BY pct_closed_365 DESC;
```

```
                                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------
  Sort  (cost=9031.43..9031.67 rows=94 width=61)
  Sort Key: (((100 * count(1)) / 365)) DESC
  ->  GroupAggregate  (cost=9023.18..9028.35 rows=94 width=61)
        Group Key: c.highway, c.area
        ->  Group  (cost=9023.18..9024.36 rows=94 width=41)
              Group Key: c.highway, c.area, (date(c.reported))
              ->  Sort  (cost=9023.18..9023.42 rows=94 width=41)
                    Sort Key: c.highway, c.area, (date(c.reported))
                    ->  Hash Join  (cost=4514.02..9020.10 rows=94 width=41)
                          Hash Cond: ((c.highway)::text = (snow_highways.highway)::text)
                          ->  Seq Scan on caltrans c  (cost=0.00..4504.32 rows=156 width=45)
                                Filter: ((condition ~~ '%CLOSED%'::text) AND ((condition ~~ '%FOR THE WINTER%'::text) OR (condition ~~
'%DUE TO SNOW%'::text)))
                          ->  Hash  (cost=4512.21..4512.21 rows=145 width=5)
                                ->  Subquery Scan on snow_highways  (cost=4509.65..4512.21 rows=145 width=5)
```

```
                              ->  Unique  (cost=4509.65..4510.76 rows=145 width=37)
                                    ->  Sort  (cost=4509.65..4510.02 rows=148 width=37)
                                          Sort Key: caltrans.highway, caltrans.area
                                          ->  Seq Scan on caltrans  (cost=0.00..4504.32 rows=148 width=37)
                                                Filter: ((condition ~~ '%CLOSED%'::text) AND ((condition ~~ '%FOR THE WINTER%'::text) OR
(condition ~~ '%DUE TO NOW%'::text)))
```

```
  Method 3: using an 'IN' subquery as a filter
  EXPLAIN
  SELECT
    highway,
    stretch,
    COUNT(1) AS days_closed,
    100 * COUNT(1) / 365 AS pct_closed_365,
    100 * COUNT(1) / 353 AS pct_closed_353
  FROM (
    SELECT
      c.highway AS highway,
      c.area AS stretch,
      DATE(c.reported) AS closure
    FROM hw2.caltrans c
    WHERE (highway, area) IN (
      SELECT
        DISTINCT highway,
        area
      FROM hw2.caltrans
      WHERE condition like '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%'))
        AND condition LIKE '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
      GROUP BY highway, stretch, closure
    ) closures
  GROUP BY highway, stretch
  ORDER BY pct_closed_365 DESC;
```

```
                                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
---------
 Sort  (cost=9020.21..9020.22 rows=1 width=61)
   Sort Key: (((100 * count(1)) / 365)) DESC
   ->  GroupAggregate  (cost=9020.14..9020.20 rows=1 width=61)
         Group Key: c.highway, c.area
         ->  Group  (cost=9020.14..9020.16 rows=1 width=41)
               Group Key: c.highway, c.area, (date(c.reported))
               ->  Sort  (cost=9020.14..9020.15 rows=1 width=41)
                     Sort Key: c.highway, c.area, (date(c.reported))
                     ->  Hash Join  (cost=4515.00..9020.13 rows=1 width=41)
                           Hash Cond: (((c.highway)::text = (caltrans.highway)::text) AND ((c.area)::text = (caltrans.area)::text))
                           ->  Seq Scan on caltrans c  (cost=0.00..4504.32 rows=156 width=45)
                                 Filter: ((condition ~~ '%CLOSED%'::text) AND ((condition ~~ '%FOR THE WINTER%'::text) OR (condition ~~
'%DUE TO SNOW%'::text)))
                           ->  Hash  (cost=4512.70..4512.70 rows=153 width=37)
                                 ->  Unique  (cost=4510.00..4511.17 rows=153 width=37)
                                       ->  Sort  (cost=4510.00..4510.39 rows=156 width=37)
                                             Sort Key: caltrans.highway, caltrans.area
                                             ->  Seq Scan on caltrans  (cost=0.00..4504.32 rows=156 width=37)
```

```
                                        Filter: ((condition ~~ '%CLOSED%'::text) AND ((condition ~~ '%FOR THE WINTER%'::text) OR
(condition ~~ '%DUE TO SNOW%'::text)))
```

Method 4: Formal Left Semijoin

```
EXPLAIN
SELECT
  highway,
  stretch,
  COUNT(1) AS days_closed,
  100 * COUNT(1) / 365 AS pct_closed_365,
  100 * COUNT(1) / 353 AS pct_closed_353
FROM (
  SELECT
    c.highway AS highway,
    c.area AS stretch,
    DATE(c.reported) AS closure
  FROM hw2.caltrans c
  WHERE EXISTS (
    SELECT
      1
    FROM hw2.caltrans
    WHERE condition like '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%'))
      AND condition LIKE '%CLOSED%' AND (condition LIKE '%FOR THE WINTER%' OR condition LIKE '%DUE TO SNOW%')
    GROUP BY highway, stretch, closure
  ) closures
GROUP BY highway, stretch
ORDER BY pct_closed_365 DESC;
```

```
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=4553.44..4553.83 rows=155 width=61)
   Sort Key: (((100 * count(1)) / 365)) DESC
   ->  GroupAggregate  (cost=4539.26..4547.80 rows=155 width=61)
         Group Key: c.highway, c.area
         ->  Group  (cost=4539.26..4541.21 rows=155 width=41)
               Group Key: c.highway, c.area, (date(c.reported))
               InitPlan 1 (returns $0)
                 ->  Seq Scan on caltrans  (cost=0.00..4504.32 rows=156 width=0)
                       Filter: ((condition ~~ '%CLOSED%'::text) AND ((condition ~~ '%FOR THE WINTER%'::text) OR (condition ~~ '%DUE
TO SNOW%'::text)))
               ->  Sort  (cost=4510.39..4510.78 rows=156 width=41)
                     Sort Key: c.highway, c.area, (date(c.reported))
                     ->  Result  (cost=0.00..4504.71 rows=156 width=41)
                           One-Time Filter: $0
                           ->  Seq Scan on caltrans c  (cost=0.00..4504.71 rows=156 width=41)
                                 Filter: ((condition ~~ '%CLOSED%'::text) AND ((condition ~~ '%FOR THE WINTER%'::text) OR (condition ~~
'%DUE TO SNOW%'::text)))
```

Overall, it seems that two operations take up most of the query's run time. The first one is the sequential scan, which is basically the first operation for all queries because we must know which relation we are working with. The filtering of the 'WHERE' clause also occurs at this stage. The second operation is the join. All types of joins take a long time to complete, around the same time as the sequential scan. As a result, the queries that contain joins are the slowest ones (method 2, 3). The other queries that don't use joins (method 0, 1, 4) take around half their run times. Of these queries, the one without any subqueries and using the keyword 'DISTINCT' (method 0) appears to be the fastest, as there are less operations to run.

2.

1. Relation R is scanned 100 000 times, once for each tuple within relation L.
2.
    a) Tuples in L are scanned once, defined by the outer loop. Tuples in R are scanned twice, once to build the index, another time for each index lookup.
    b) Create the index: 2 * bR (once to read from memory, once to write to memory)
    Move the index into memory: N block transfers
    Move L's blocks into memory: bL / (M - N) block transfers
    Extract tuple from disk using the indexed pointer: J * bR

    Block Transfer complexity (with precomputed index): N + J * bR + bL / (M - N)
    Block Transfer complexity (without precomputed index): 2 * bR + N + J * bR + bL / (M - N)

3.

1. Yes, the decomposition is lossless. Here's why:
    If a decomposition is lossless, these 3 statements must hold:
    1. R1 U R2 = R - true
    2. R1 n R2 != {0} - true, A is a common attribute
    3. R1 n R2 -> R1 or R1 n R2 -> R2, ie. Is A a superkey for either R1 or R2?
        We must test functional dependencies.

        Given F, we can find the closure F+:
        A->B and A->C because of decomposition rule
        A->D because of the transitivity axiom
        A->CD because of the union rule
        A->E because of the transitivtiy axiom

        Because there is a functional dependency from A on every attribute of R2 (D, E), A is a superkey for R2.
        Therefore, the decomposition is lossless.
2. F: {A->B, C->B, C->A, BC->A}
3. It is not in BCNF, because based on the closure of F, there exist non-trivial dependencies, and A is the only attribute that has functional dependencies. However, A is not a superkey because the attribute F cannot be reached. So the relation R is not in BCNF.

    (A, B, C, D, E, F)
    = (A, B, C, D, F) U (C, E)
    = (A, B, C, F) U (B, D) U (C, E)
    = (A, F) U (A, B, C) U (B, D) U (C, E)
4. there is actually an implied multidependence A->>D
    tuples:
      //(a, b1, c1, d1)
    (a, b1, c1, d2)
    (a, b1, c1, d3)
    (a, b1, c2, d1)
    (a, b1, c2, d2)
    (a, b1, c2, d3)
    (a, b1, c3, d1)
    (a, b1, c3, d2)
    (a, b1, c3, d3)

    (a, b2, c1, d1)
    (a, b2, c1, d2)
    (a, b2, c1, d3)
    (a, b2, c2, d1)
      //(a, b2, c2, d2)

(a, b2, c2, d3)
(a, b2, c3, d1)
(a, b2, c3, d2)
(a, b2, c3, d3)

(a, b3, c1, d1)
(a, b3, c1, d2)
(a, b3, c1, d3)
(a, b3, c2, d1)
(a, b3, c2, d2)
(a, b3, c2, d3)
(a, b3, c3, d1)
(a, b3, c3, d2)
//(a, b3, c3, d3)

5. The relation is not in 4NF because it is not in BCNF. The functional dependency AB->E is not trivial and AB is not a superkey for the relation.
   We can decompose the relation into: R1(A, B, E) U R2(A, B, C, D, F)
   Now both tables are in BCNF.

   R1 is not in 4NF because it has multivalued dependency A->>B.
   So, we decompose R1 into: R3(A, B) U R4(A, E)

   R2 is not in 4NF because it has two multivalued dependencies.
   So, we decompose R2 into: R5(A, B, C) U R6(A, B, D, F)

   R5 is not in 4NF because it has multivalued dependency A->>B.
   So, we decompose R5 into: R7(A, B) U R8(A, C)

   R6 is not in 4NF because it has multivalued dependency A->>B.
   So, we decompose R6 into: R9(A, D, F) U R10(A, B)

   Finally, we have (A, D, F), (A, E), (A, C), (A, B)
6. The relation has functional dependencies and multivalued dependencies. Let us use A, B, C, D to represent venue, year, band, and genre respectively.
   The functional dependencies are as follows:
   F={C->D}
   The multivalued dependencies are as follows:
   BC->>A, C->>B

   The relation is not in 2NF, because D depends only on C, which is part of the candidate key (A, B, C).
   The relation is not in 3NF, because it is not already in 2NF.
7. The new relation has functional dependencies as follows:
   F={ABC->D}
   The new relation has multivalued dependencies as follows:
   BC->>A, C->>B

   The relation is in 2NF, because every attribute is either in the candidate key (A, B, C) or depends on the entire candidate key.
   The relation is in 3NF, because there are no transitive functional dependencies.