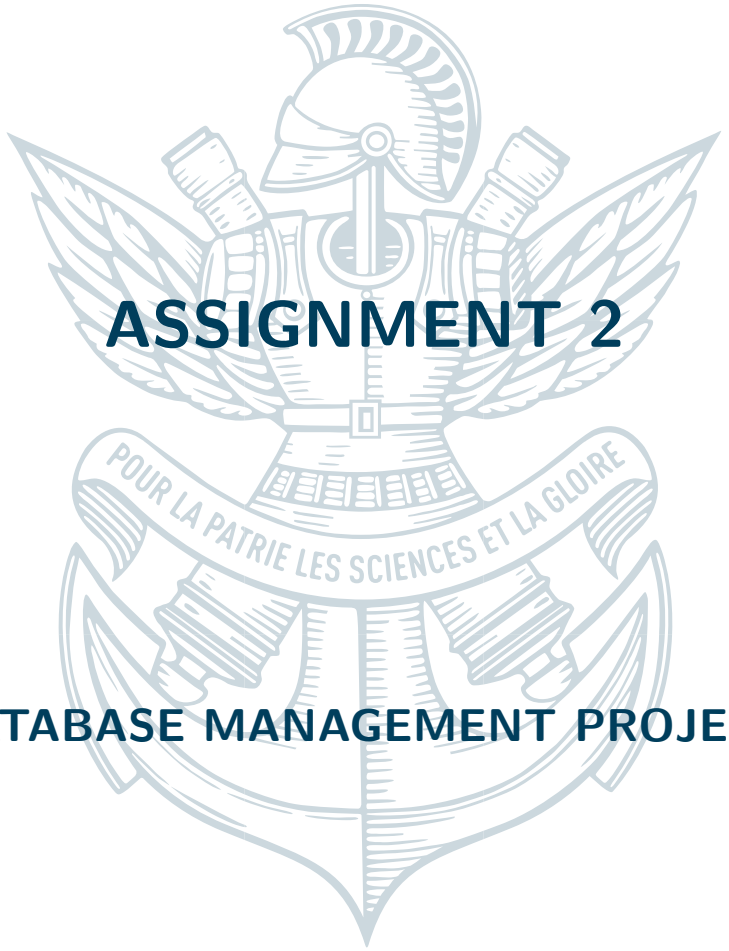

ASSIGNMENT 2

DATABASE MANAGEMENT PROJECT

15 novembre 2019

Luiz Bezerra Pinheiro, Mathews Lucas

Table des matières

1 Queries

1.1 For each country, return the country name and the names of all artists who were born in this country. Return the results ordered first by the country name, then by the artist name.

```
SELECT C.name, P0.name \newline
FROM country C, artist P0 \newline
WHERE C.id = P0.area \newline
ORDER BY (C.name, P0.name); \newline
```

1.2 Find the countries with the largest number of artists, and return the name of these countries along with the names of their artists.

```
WITH aux AS
    (
        SELECT C.id, C.name, count(*) as cnt
        FROM country C, artist P0
        WHERE C.id = P0.area
        GROUP BY (C.id)
        ORDER BY cnt DESC
    ),
max_countries AS
    (
        SELECT aux.name
        FROM aux
        WHERE aux.cnt >= ALL(SELECT cnt FROM aux)
    )

SELECT C.name, A.name
FROM country C
INNER JOIN artist A ON A.area = C.id
WHERE C.name IN (SELECT name FROM max_countries);
```

1.3 Find the ids of artists who have released an album in a country whose name starts with an A.

```

SELECT DISTINCT P0.id
FROM release R, release_country RC, country C, artist P0, release_has_artist RhA
WHERE RhA.artist = P0.id AND RhA.release = R.id AND RC.release = R.id AND RC.country =
```

1.4 For each country id, return the number of different releases in this country. Order the results in descending order according to the number of releases.

```
SELECT C.id, count(1) AS cc
FROM release_country RC, country C
WHERE RC.country = C.id
GROUP BY (C.id)
ORDER BY count(1) DESC;
```

1.5 For every artist and release such that the artist is a main contributor to this release, return the release ID and the artist ID.

```

SELECT RhA.release, RhA.artist
FROM release_has_artist RhA, release R, artist P0
WHERE RhA.artist = P0.id AND RhA.release = R.id AND RhA.contribution = 0
ORDER BY (RhA.release, RhA.artist);
```

1.6 Return the triples (id 1 , id 2 , c) such that id 1 , id 2 are ids of different artists that have collaborated for some releases, and c is the count of releases in which they have collaborated.

```

SELECT RhA1.artist, RhA2.artist, count(1)
FROM release_has_artist RhA1, release_has_artist RhA2, release R
WHERE RhA1.release = R.id AND RhA2.release = R.id AND RhA1.artist <> RhA2.artist
GROUP BY (RhA1.artist, RhA2.artist)
ORDER BY (RhA1.artist, RhA2.artist);
```

1.7 Return the pairs (country id, release id) such that the release has been made in that country and the release has at least two tracks.

```
SELECT RC.country, RC.release
FROM release_country RC, release R
WHERE RC.release = R.id AND ( (SELECT COUNT(*) FROM track T, release R1 WHERE T.relea
ORDER BY (RC.release, RC.country);
```

1.8 Return the pairs (country id, release id) such that a release with this id has been made in this country, yet the country is not the first (in temporal order) in which a release of this id has been made.

```
SELECT RC.release, RC.country, RC.syear, RC.smonth, RC.sday
FROM release_country RC
WHERE RC.country <> ( SELECT RCin.country
                                                FROM release_country RCin, country C
                                                WHERE RCin.release = RC.release AND RCin.
                                                ORDER BY (RCin.syear, RCin.smonth, RCin.s
ORDER BY (RC.syear, RC.smonth, RC.sday, RC.country);
```

1.9 For each country, and each individual artist x (a Person) from that country, the number of artists born before x in that country and the number of artists born before x globally.

First we tried to count the cases with unknown information, let's say for example we know that someone born at 1900 is younger than someone born at 1000, even if they have NULLs months or days, even though this consideration is hard to optimise and requires a lot of computational power. So we did not consider any incomplete date.

```

SELECT C.name, P0.name,
    (
        SELECT COUNT(*)
        FROM artist P
        WHERE P.type = 1 AND P.area = P0.area AND
              (
                        (P.syear IS NOT NULL) AND
                ( P0.syear > P.syear OR ( P0.syear = P.syear AND
                ( (P0.smonth IS NOT NULL AND P.smonth IS NOT NULL) AND
                ( P0.smonth > P.smonth OR (P0.smonth = P.smonth AND
                ( (P0.sday IS NOT NULL AND P.sday IS NOT NULL)
                AND (P0.sday > P.sday))))))))
        ) as nb,

    (
        SELECT COUNT(*)
        FROM artist P
        WHERE (
```

```
20                          ( P.syear IS NOT NULL) AND
21                  ( P0.syear > P.syear OR ( P0.syear = P.syear AND
22                  ( (P0.smonth IS NOT NULL AND P.smonth IS NOT NULL) AND
23                  ( P0.smonth > P.smonth OR (P0.smonth = P.smonth AND
24                  ( (P0.sday IS NOT NULL AND P.sday IS NOT NULL)
25                  AND (P0.sday > P.sday))))))))
26          ) as nb_global
27  FROM artist P0
28  INNER JOIN country C ON C.id = P0.area
29  WHERE P0.type = 1 AND P0.syear IS NOT NULL; -- 1 -> 'Person';
```

2
Query profiling

2.1 Measuring execution time

The following table contains the result of each Query at each simulation, the mean value, the standard deviation and the coefficient of variation obtained for this Query.

Query	t_1	t_2	t_3	t_4	t_5	Mean	σ	CV
1	4818.70	3757.36	3889.00	3806.05	3752.53	4004.73	458.31	11.44
2	703.84	490.44	524.18	501.36	480.07	539.98	93.05	17.23
3	6445.08	3714.95	3161.21	2330.65	2560.92	3642.56	1656.96	45.49
4	663.25	505.17	481.43	489.10	451.83	518.16	83.39	16.09
5	12828.02	9895.83	10635.54	9269.36	8277.01	10181.15	1713.98	16.83
6	6840.69	5130.18	4201.93	4070.05	3722.08	4792.99	1257.54	26.24
7	31614.85	27247.88	24050.51	27861.35	29284.52	28011.82	2779.38	9.92
8	97852.83	99856.04	102542.22	98990.35	99468.71	99742.03	1736.58	1.74
9	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN

Table 1 : Queries execution time(ms) and variable analyses

The Algorithm used at the ninth Query works well with limited values for the Query, which indicates that it is correct, but for the complete dataset it needs more than 10 minutes to execute meaning that the first method used is really inefficient.

	1	2	3	4	5	6	7	8	9
Deviation(%)	20.33	30.35	76.94	28.00	26.00	42.72	12.86	-1.89	NAN

Table 2 : Deviation between the first run and the average

We believe that is associated with the cache memory, but we can't explain this phenomena.

3
Analyze the query plan

3.1 Query 1

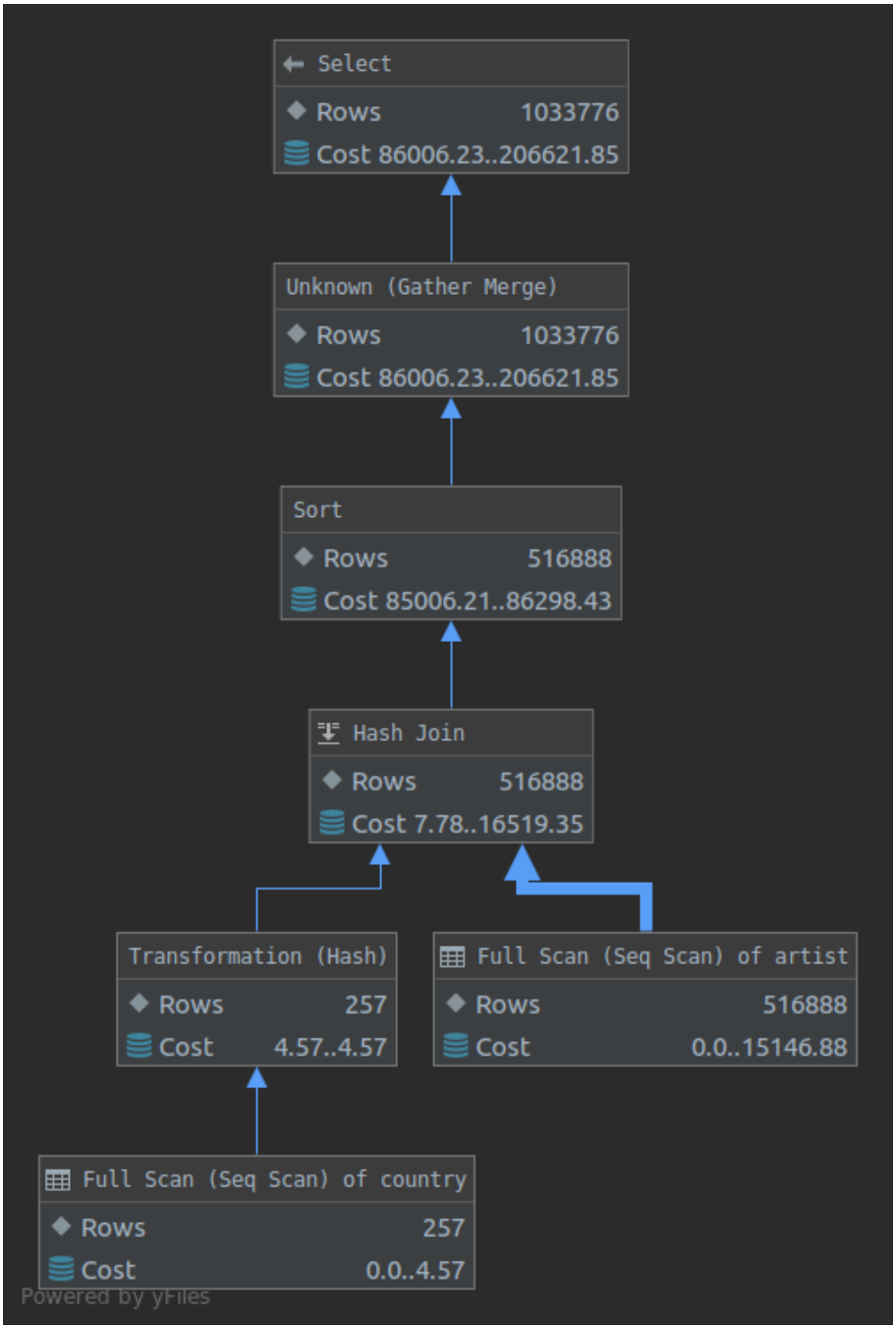

FIGURE 1 – Explain Plan of the first query

The sort operation clearly has the highest cost than the others. The operations are really simple, having a small cost compared to the dataset size. There isn't lot of room for improvement.

3.2 Query 2

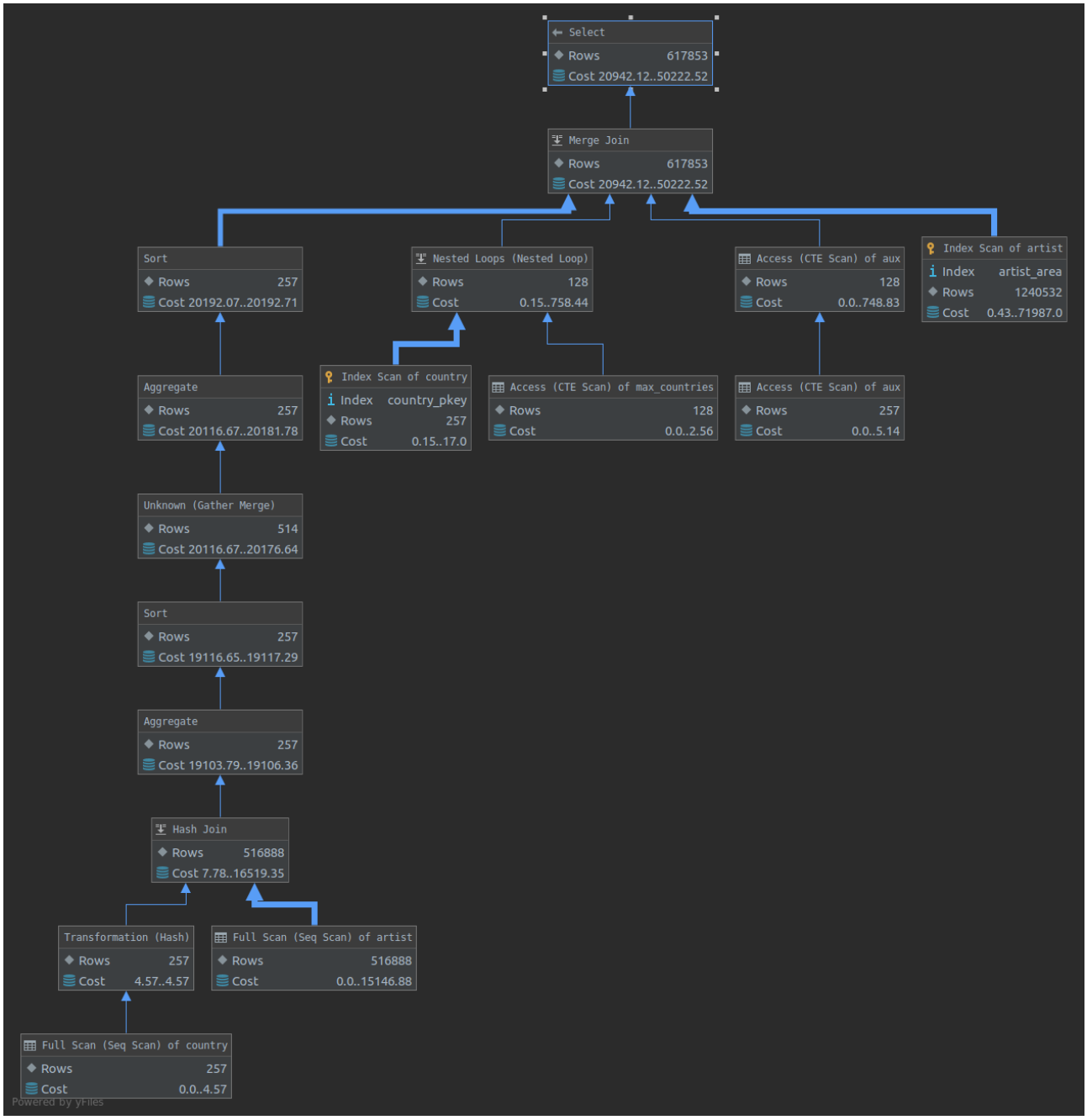

FIGURE 2 – Explain Plan of the second query

The operation with highest cost is the Aggregate of Artist with Country, at the aux creation. This operation can be optimized by doing an INNER JOIN instead of a CROSS JOIN, anyway the cost reduction is not relevant.

3.3 Query 3

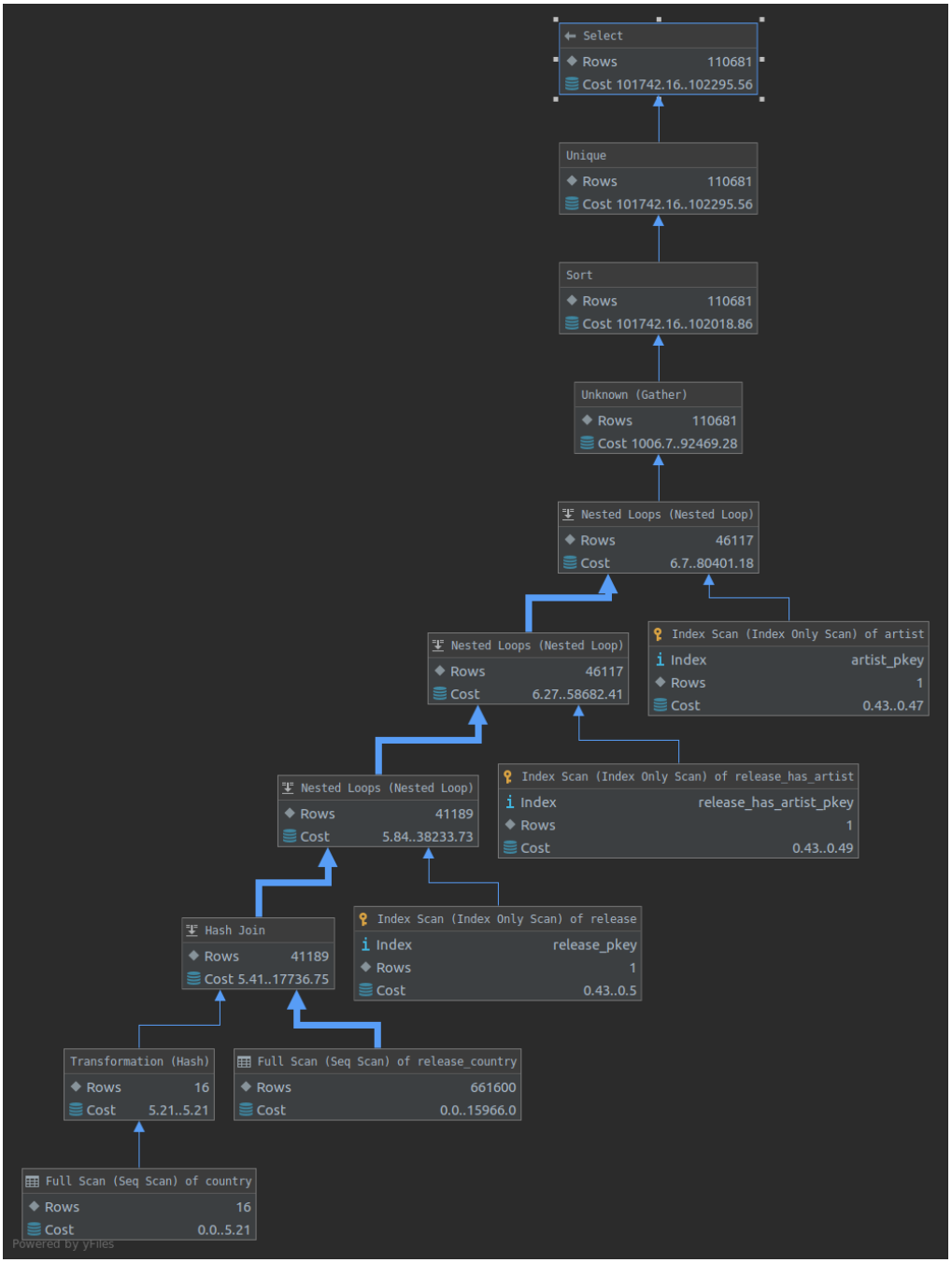

FIGURE 3 – Explain Plan of the third query

The operation with highest cost is the Sort, which can be optimized by reducing the number of Rows using an INNER JOIN instead of a CROSS JOIN.

3.4 Query 4

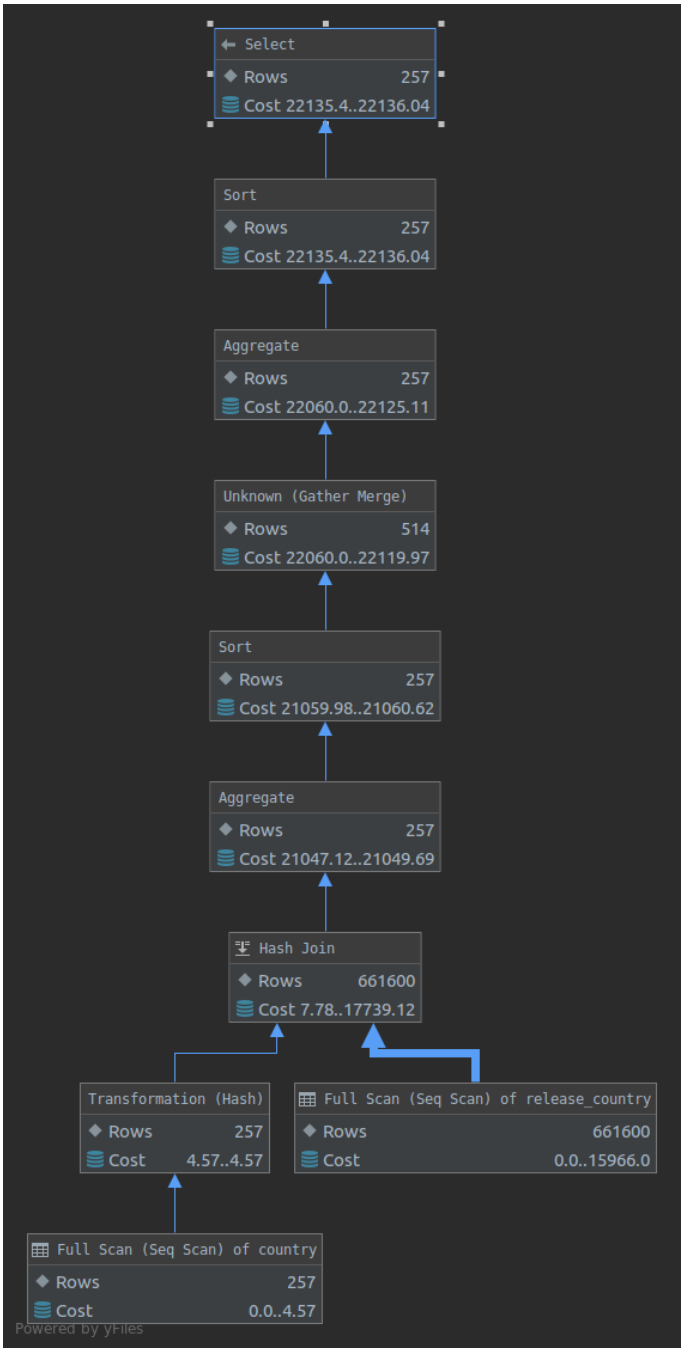

FIGURE 4 – Explain Plan of the forth query

The highest cost operation is the Aggregate of country and country_release. This is a very simple Query that can't be significantly optimised.

3.5 Query 5

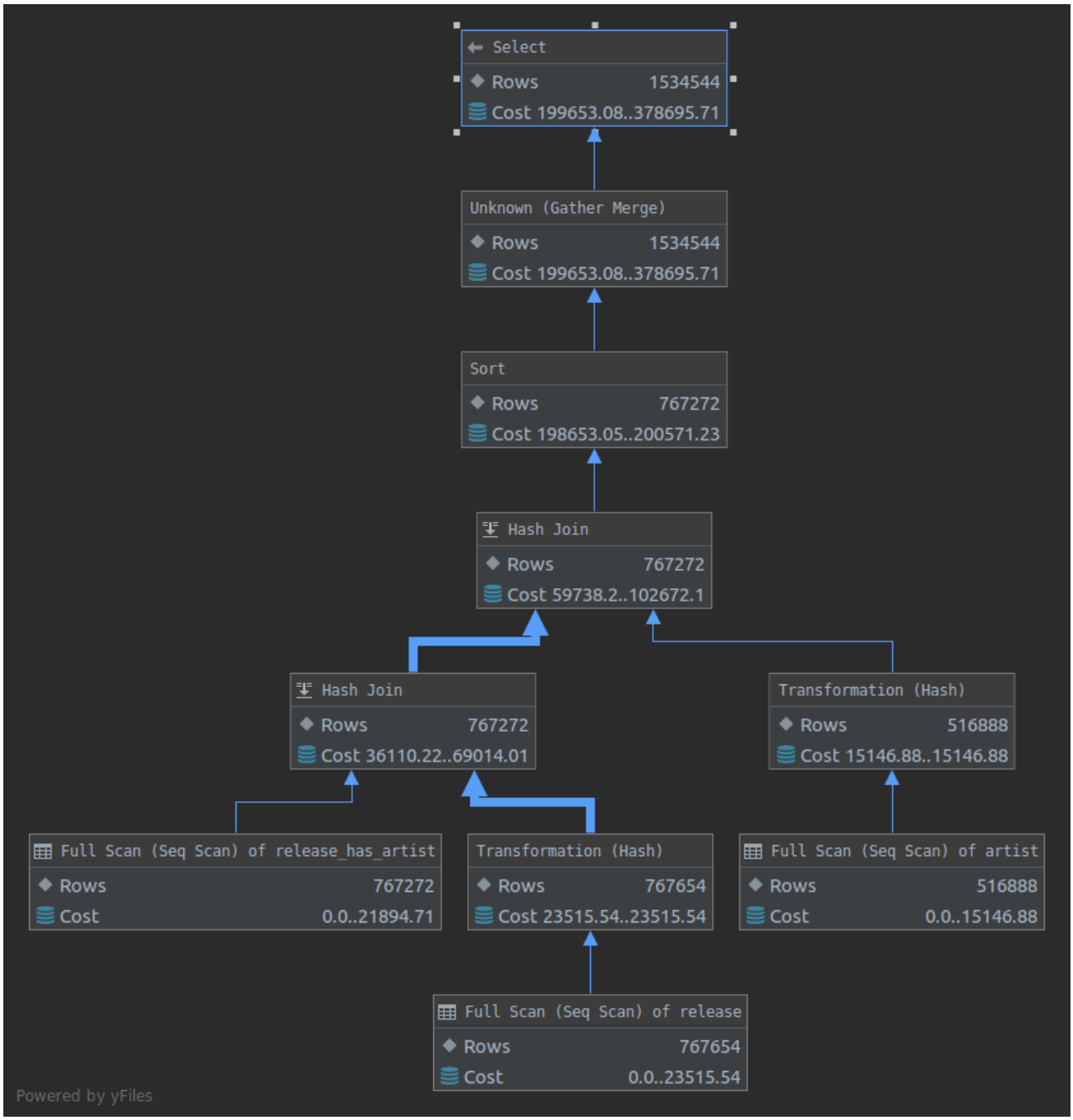

FIGURE 5 – Explain Plan of the fifth query

The operation with highest cost is the Sort, which can be optimized by reducing the number of Rows using an INNER JOIN instead of a CROSS JOIN.

3.6 Query 6

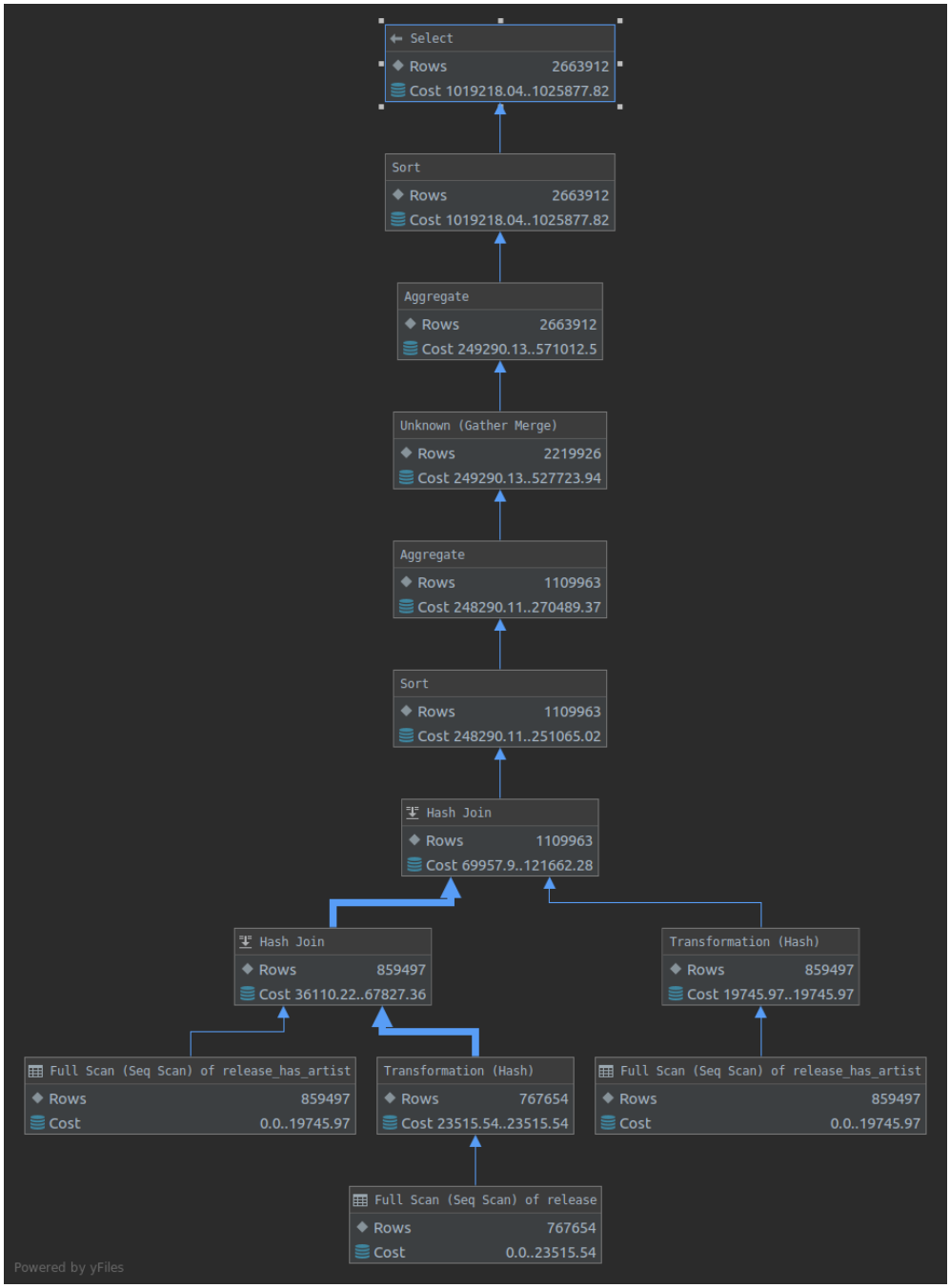

FIGURE 6 – Explain Plan of the sixth query

The highest cost operation is the last Sort operation. It can be optimized by reducing the number of Rows using an INNER JOIN instead of a CROSS JOIN, but the last sort it hard to optimise knowing that it has been applied at the final table.

3.7 Query 7

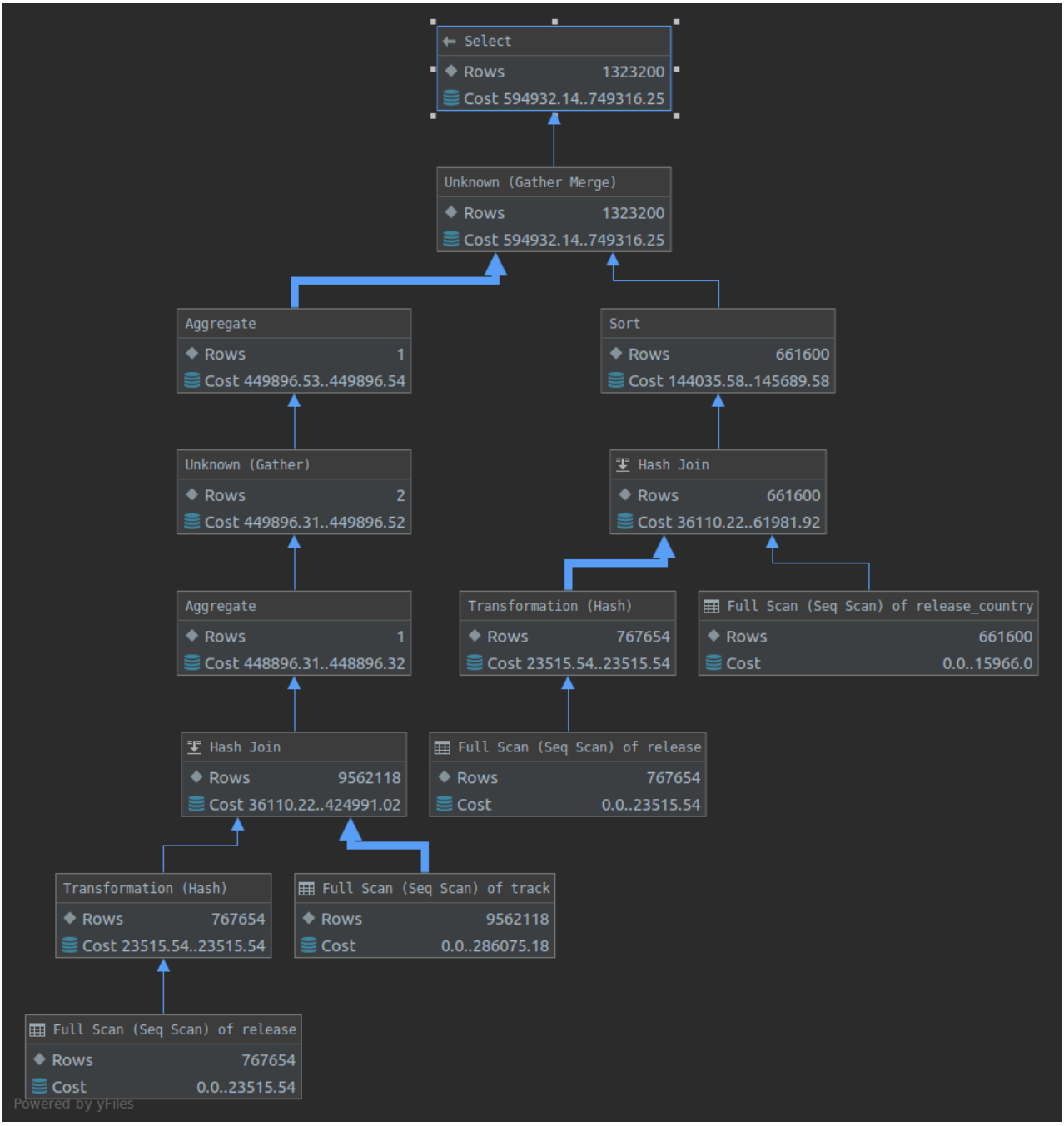

FIGURE 7 – Explain Plan of the seventh query

The highest cost operation is the first Aggregate between release and track which is expected, since a release is associated with multiples tracks.

3.8 Query 8

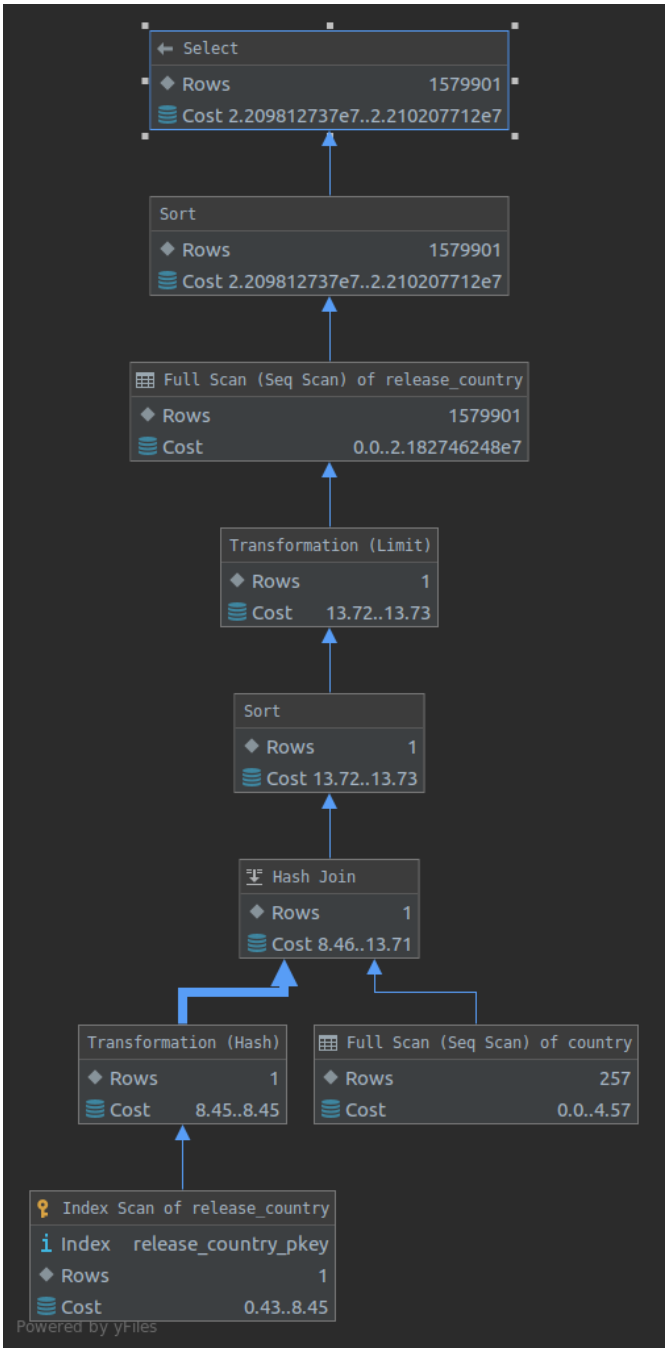

FIGURE 8 – Explain Plan of the eighth query

The highest cost operation is the last sorting that comes before the selection. This operation has a really high cost as a result of the discarded small sorts that comes first. The solution to optimize this query is by sorting the release_country at first and then getting the first country of each release.

3.9 Query 9

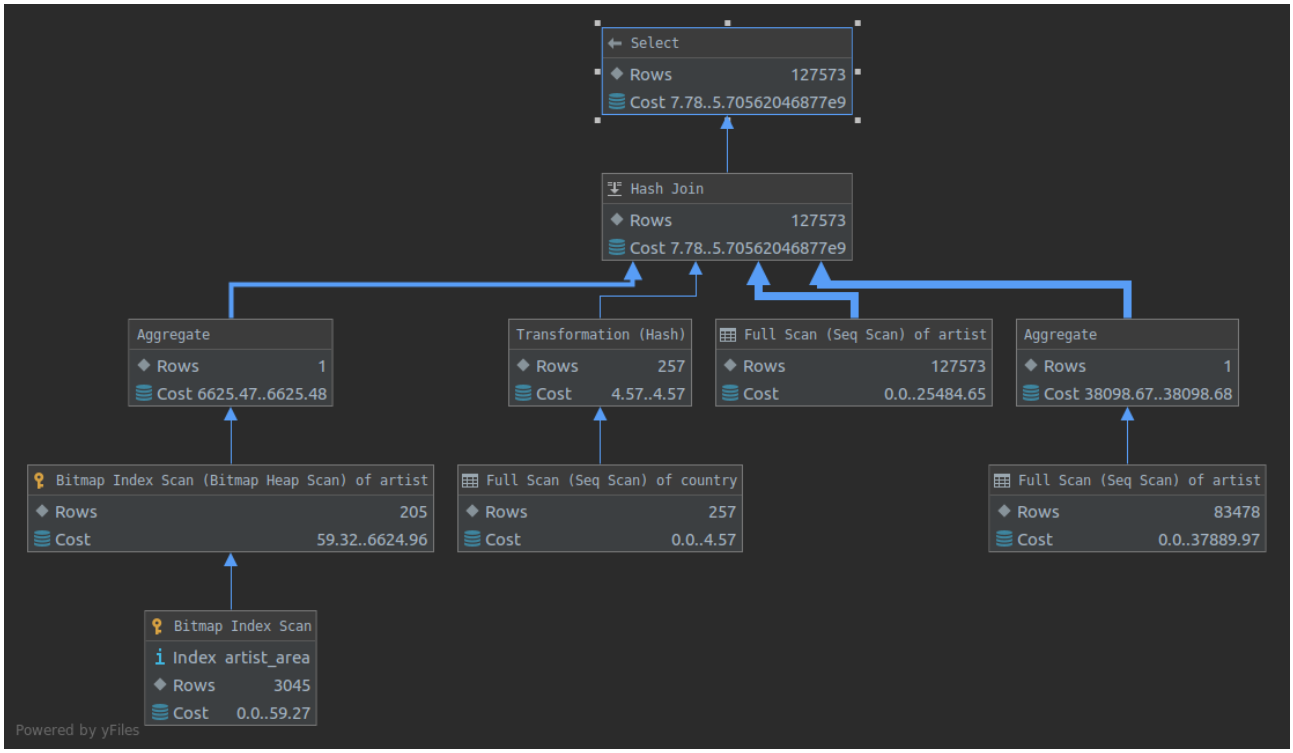

FIGURE 9 – Explain Plan of the ninth query

The Join operation of artist and country has the highest cost. As in the eighth we are not re-using a result, this way for each element of our dataset we re-calculating the same operation, which results in a enormous JOIN operation. To solve this problem we must find a way to reuse this result, for example sorting the data.

4
Queries optimisation

The fist query we are going to optimise is the third :

```
SELECT DISTINCT P0.id
FROM release R, release_country RC, country C, artist P0,  release_has_artist RhA
WHERE RhA.artist = P0.id AND RhA.release = R.id AND RC.release = R.id  AND RC.country =
```

As we can see in this query we did a Cartesian product which can be optimised by INNER JOINs or views. The optimised version is :

```
WITH ACountries AS
    (
     SELECT C.id
     FROM country C
```

```
    WHERE C.name LIKE 'A%'
  ),
   releasesAtA AS
  (
      SELECT RC.release
      FROM release_country RC
      INNER JOIN ACountries C ON RC.country = C.id
  )
  SELECT R.artist
  FROM release_has_artist R
  INNER JOIN releasesAtA RAA ON R.release = RAA.release
  GROUP BY R.artist
  ORDER BY R.artist;
```

This way :

t_1	t_2	t_3	t_4	t_5	Mean	σ	CV	Reduction
2544.8	1394.5	1464.4	1394.7	1415.8	1642.9	505.0	30.7	54.9

Table 3 : Results for the optimisation of query 3

The second optimised query was the ninth, at first time our query took undetermined time to be executed. We optimised it using views.

New version :

```
CREATE VIEW ArtistsOrdered AS
    (SELECT row_number() OVER (ORDER BY A.syear, A.smonth, A.sday) - 1 AS nb_global,
    FROM Artist A
    WHERE A.syear IS NOT NULL AND A.smonth IS NOT NULL AND A.sday IS NOT NULL);

EXPLAIN ANALYSE
SELECT C.name as country, P0.name as name, P0.nb, P0.nb_global
FROM ArtistsOrdered P0
INNER JOIN country c on P0.area = c.id
WHERE P0.type = 1
ORDER BY nb, nb_global;
```

The results :

t_1	t_2	t_3	t_4	t_5	Mean	σ	CV	Reduction
695.9	727.4	717.7	700.0	700.1	708.2	13.6	1.9	NAN

Table 4 : Results for the optimisation of query 9

The last optimisation was the eighth.

New version :

```

CREATE VIEW ReleaseCountryOrdered AS
    (SELECT row_number() OVER (PARTITION BY RC.release ORDER BY  RC.year, RC.month, R
    FROM release_country RC);

EXPLAIN ANALYSE
SELECT RC.release, RC.country
FROM ReleaseCountryOrdered RC
WHERE RC.release_order > 1;
```

Than we obtained :

t_1	t_2	t_3	t_4	t_5	Mean	σ	CV	Reduction
2742.0	2870.7	2784.7	2815.8	2756.2	2793.9	51.4	1.8	97.2

Table 5 : Results for the optimisation of query 8

4.1 New Execution plans

4.1.1 • Query 3

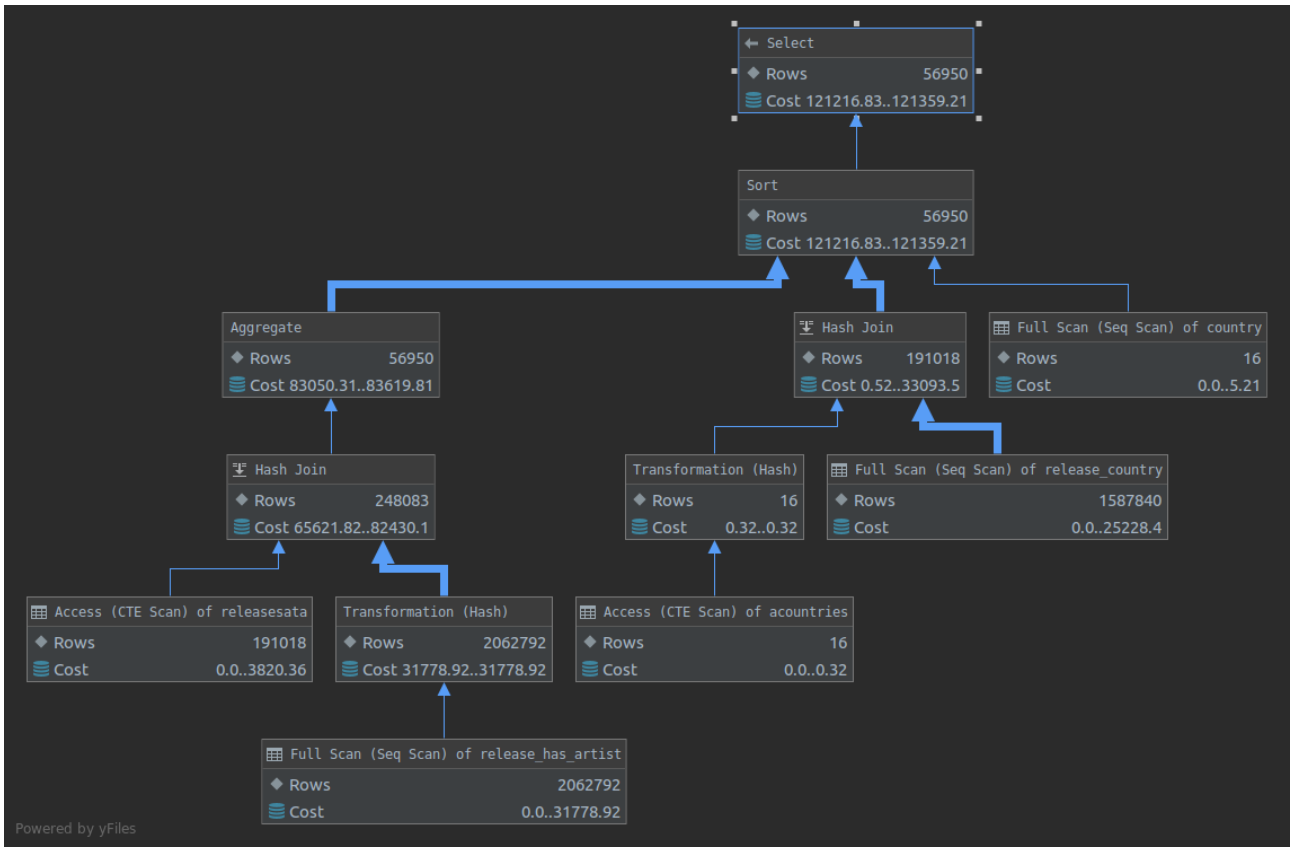

Figure 10 – Explain Plan of the third query - Optimized

4.1.2 • Query 8

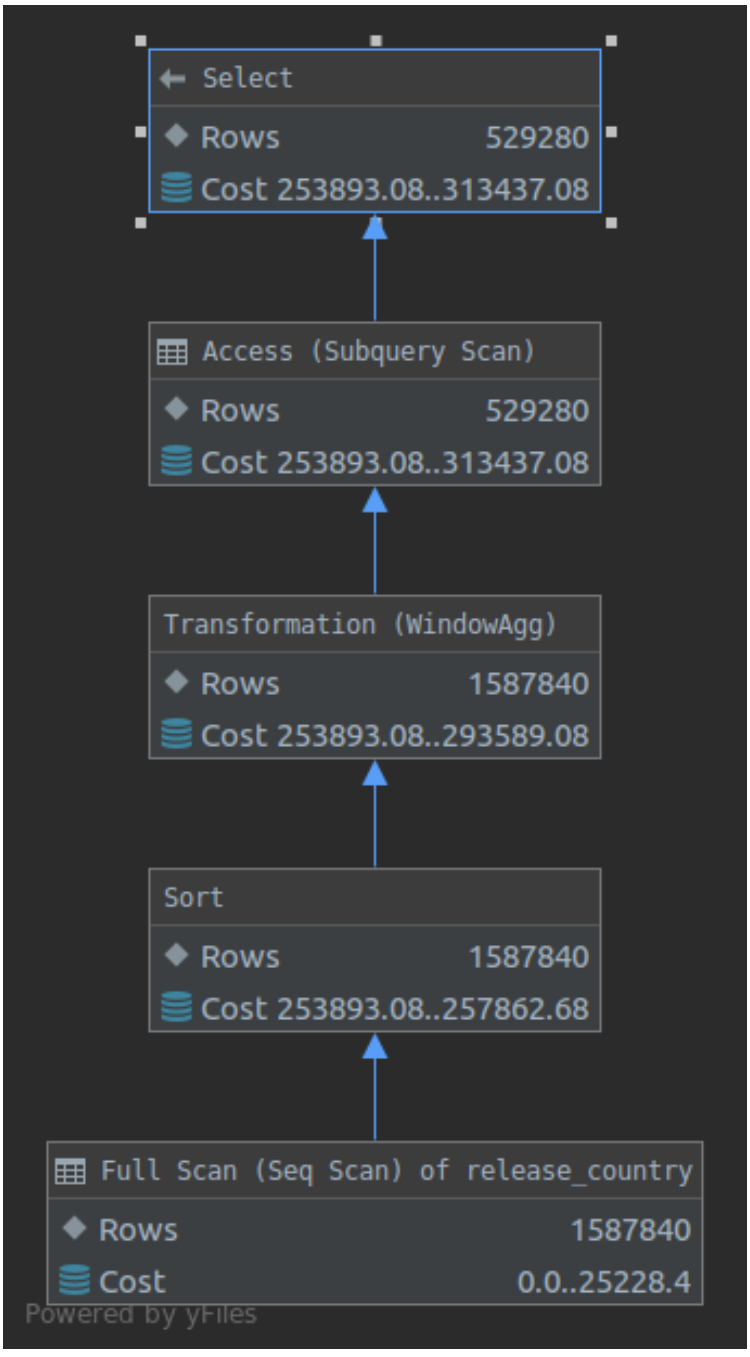

Figure 11 – Explain Plan of the eighth query - Optimized

4.1.3 • Query 9

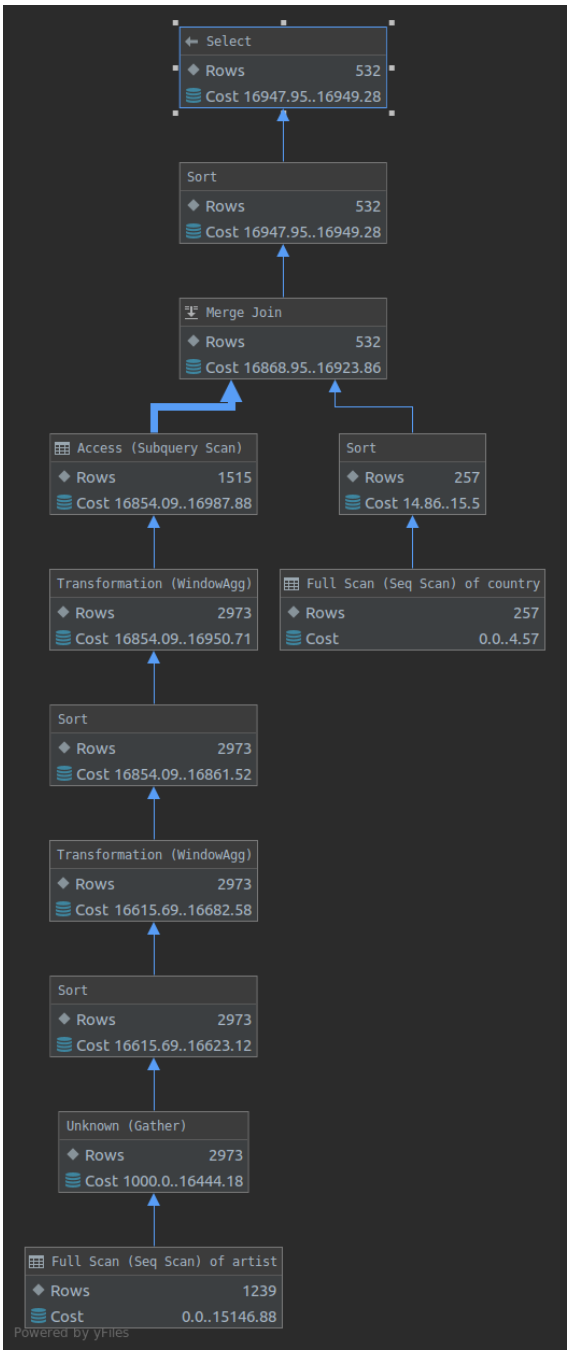

Figure 12 – Explain Plan of the ninth query - Optimized