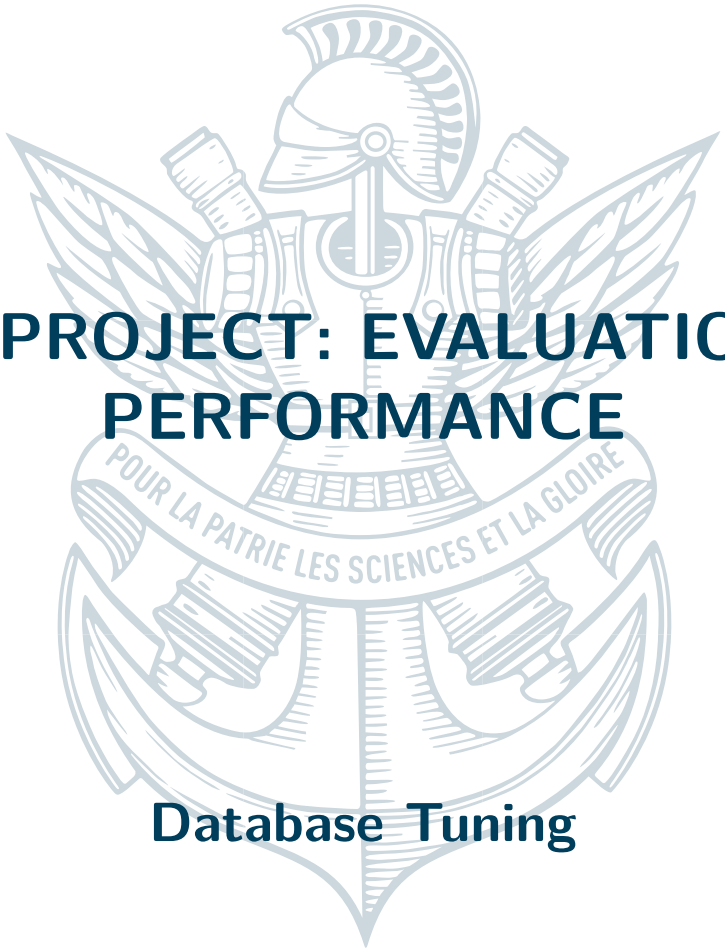

INF553 PROJECT: EVALUATION AND PERFORMANCE

Database Tuning

15 novembre 2019

Guilherme Marra, João Henrique Oliveira

1 Queries

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 country.name, artist.name
FROM artist, country
WHERE artist.area = country.id
ORDER BY country.name, artist.name;
```

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

```
SELECT country.name, artist.name
FROM artist, country
WHERE artist.area = country.id and artist.area = (SELECT   artist.area
                                                  FROM     artist
                                                  GROUP BY artist.area
                                                  ORDER BY COUNT(artist.area)
                                                  LIMIT 1) --> for n=1
                                                  -- countries with the
                                                  -- largest number of
                                                  -- artists
ORDER BY artist.name;
```

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

```
SELECT artist.id
FROM release_country, release_has_artist, country, artist
WHERE release_has_artist.artist = artist.id
        and release_has_artist.release = release_country.release
        and  release_country.country = country.id and country.name LIKE 'A%'
GROUP BY artist.id
ORDER BY  artist.id;
```

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 release_country.country, COUNT(release_country.country)
FROM release_country
GROUP BY release_country.country
ORDER BY COUNT(release_country.country) DESC;
```

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 release_has_artist.release, release_has_artist.artist
FROM release_has_artist
WHERE release_has_artist.contribution = 0;
```

6. Return the triples (*id1, id2, c*) such that *id1, id2* are ids of different artists that have collaborated for some releases, and *c* is the count of releases in which they have collaborated.

```
SELECT table1.artist, table2.artist, count(1)
FROM release_has_artist as table1, release_has_artist as table2
WHERE table1.release = table2.release and table1.artist != table2.artist
GROUP BY (table1.artist, table2.artist)
ORDER BY (table1.artist, table2.artist);
```

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 release_country.country, track.release
FROM track, release_country
WHERE track.release = release_country.release
GROUP BY track.release, release_country.country
HAVING COUNT (track.release) > 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 table1.release, table1.country
FROM release_country as table1, release_country as table2
WHERE table1.release=table2.release and table1.country != table2.country
    and (table1.year>table2.year
            or (table1.year=table2.year and table1.month > table2.month)
            or (table1.year=table2.year and table1.month = table2.month
                                        and table1.day>table2.day) )
GROUP BY table1.release, table1.country
ORDER BY table1.release;
```

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.

```
SELECT country.name as country, artist.name as name,
(
        SELECT COUNT(*)
        FROM artist as aux1
        WHERE aux1.area = artist.area AND
              (aux1.syear*10000 + aux1.smonth*100 + aux1.sday <
                      artist.syear*10000 + artist.smonth*100 + artist.sday)
        LIMIT 1
) as nb,
(
        SELECT COUNT(*)
        FROM artist as aux2
        WHERE (aux2.syear*10000 + aux2.smonth*100 + aux2.sday <
                      artist.syear*10000 + artist.smonth*100 + artist.sday)
        LIMIT 1
) as nb_global
FROM artist
INNER JOIN country ON country.id = artist.area
WHERE artist.type = 1;
```

2
Query profiling

2.1 Measuring execution time

Queries	Time1(ms)	Time2(ms)	Time3(ms)	Time4(ms)	Time5(ms)
1	1627	1288	1178	1248	1164
2	858	762	928	818	761
3	1589	1403	890	870	871
4	552	564	505	504	547
5	12	14	25	27	24
6	4286	4205	4325	4407	4695
7	36293	30134	30736	26245	25626
8	398	426	405	410	544
9	107164	111404	123695	118247	111459

Table 1 – Execution time of each query in milliseconds

2.2 Analyzing the query plan

2.2.1 • Query 1

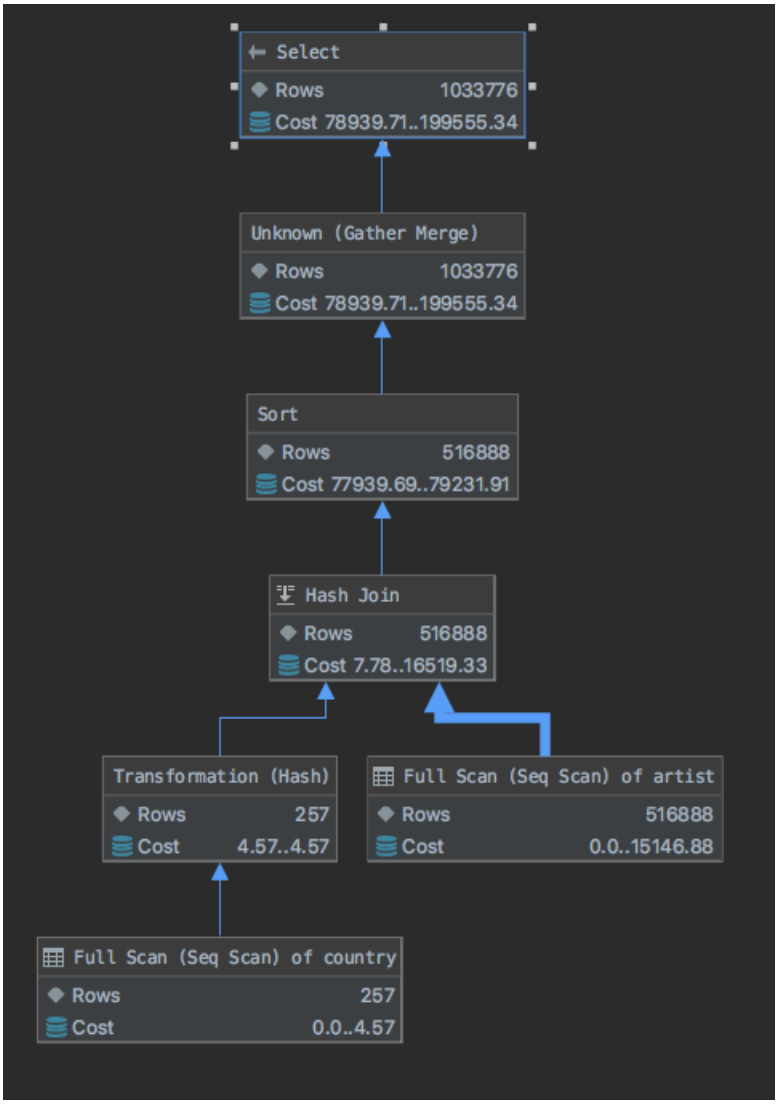

Figure 1 – Explain plan of Query 1

The cost of this query is dominated by the sort related to ORDER at the end of the query. Since it is a very direct query, we don't see much room for improvement.

2.2.2 • Query 2

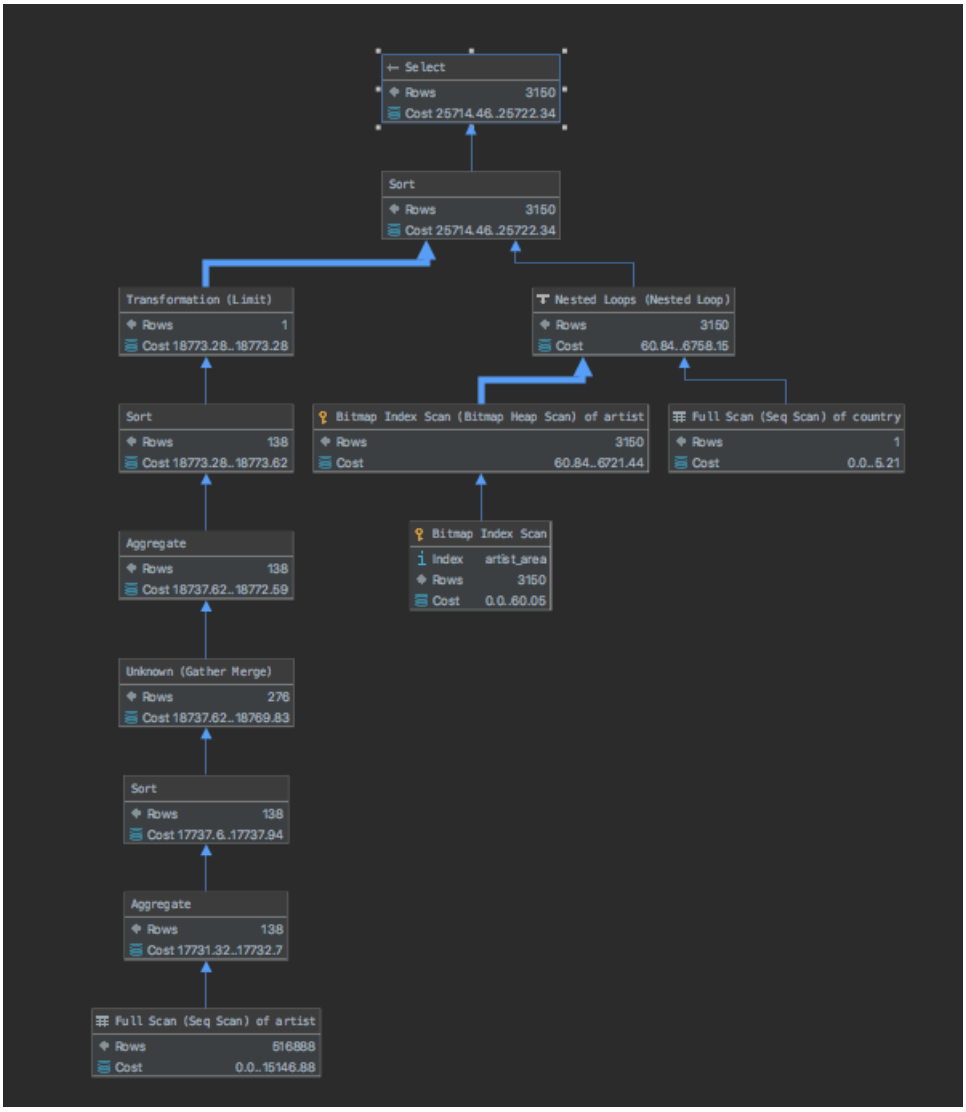

Figure 2 – Explain plan of Query 2

2.2.3 • Query 3

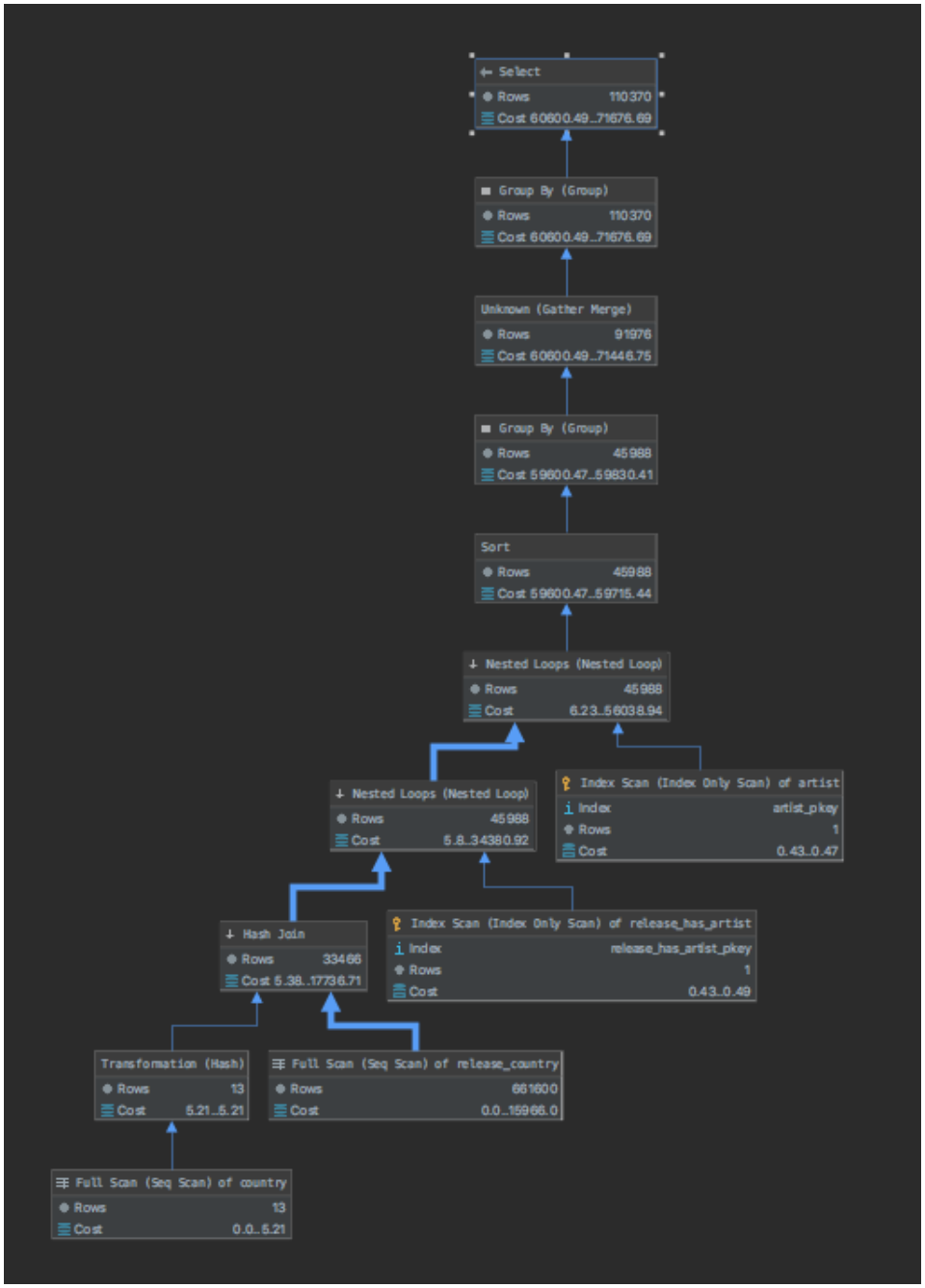

Figure 3 – Explain plan of Query 3

The complexity of query 3 is essentially given by the sorting required by GROUP BY and ORDER done at artist.id. However the execution time would not vary much.

2.2.4 • Query 4

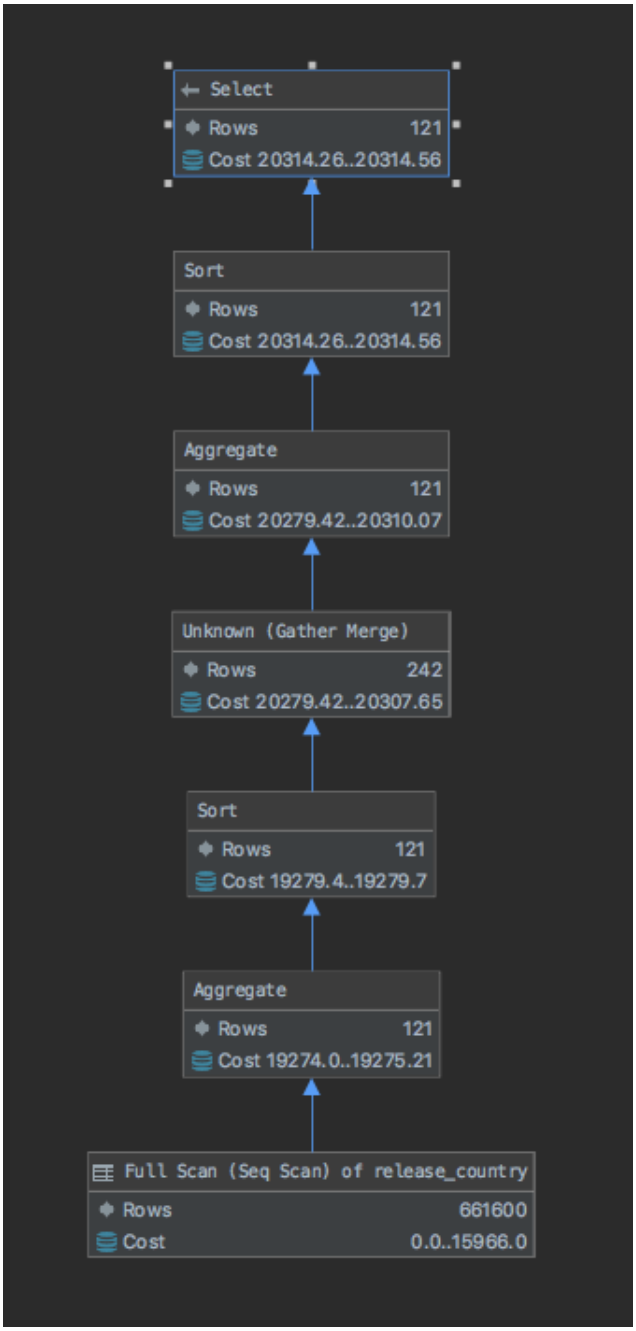

Figure 4 – Explain plan of Query 4

Query 4 has its cost well distributed in sorts and the related Gather Merge in these sorts to count per country releases. This way we don't see much room for optimmization in this query.

2.2.5 • Query 5

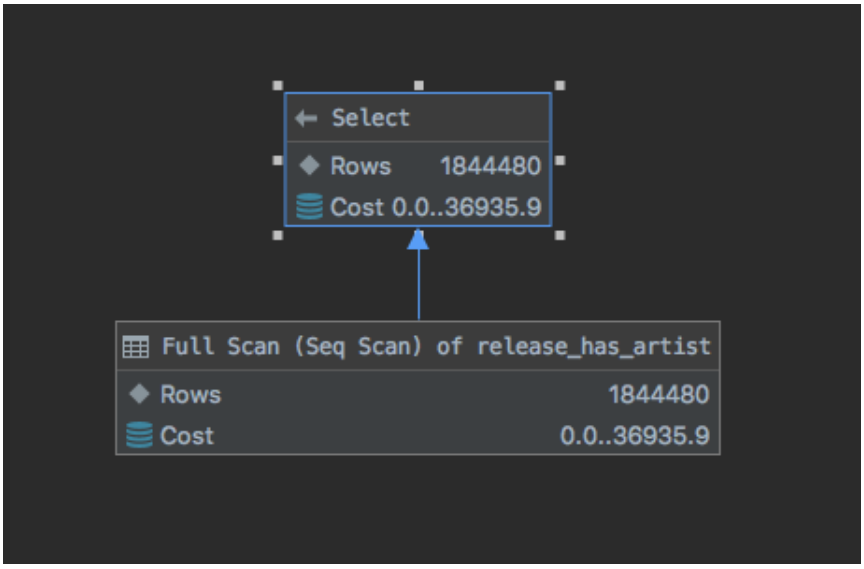

Figure 5 – Explain plan of Query 5

The only operation of the fifth query is a scan across the release_has_astist table. The only way to improve its performance would be to create an index by 'contribution'.

2.2.6 • Query 6

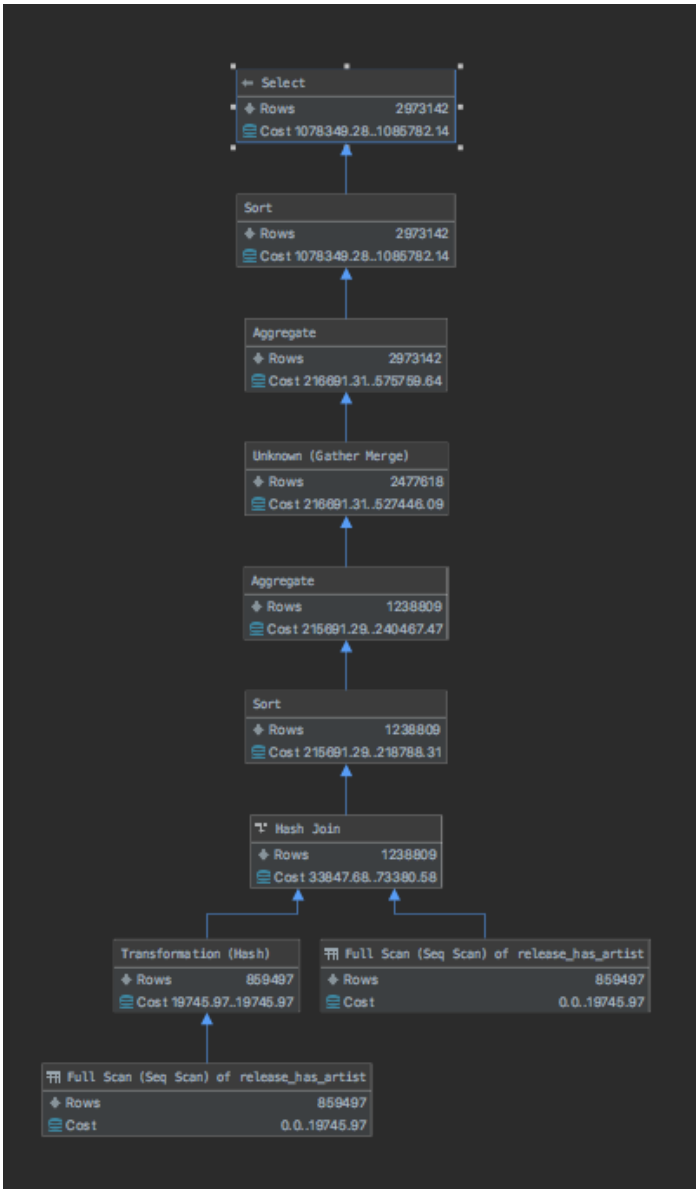

Figure 6 – Explain plan of Query 6

The cost of this query is dominated by the sort related to GROUP BY and ORDER BY at the end of the query. Probably by adding more parallel processes we can optimize this query.

2.2.7 • Query 7

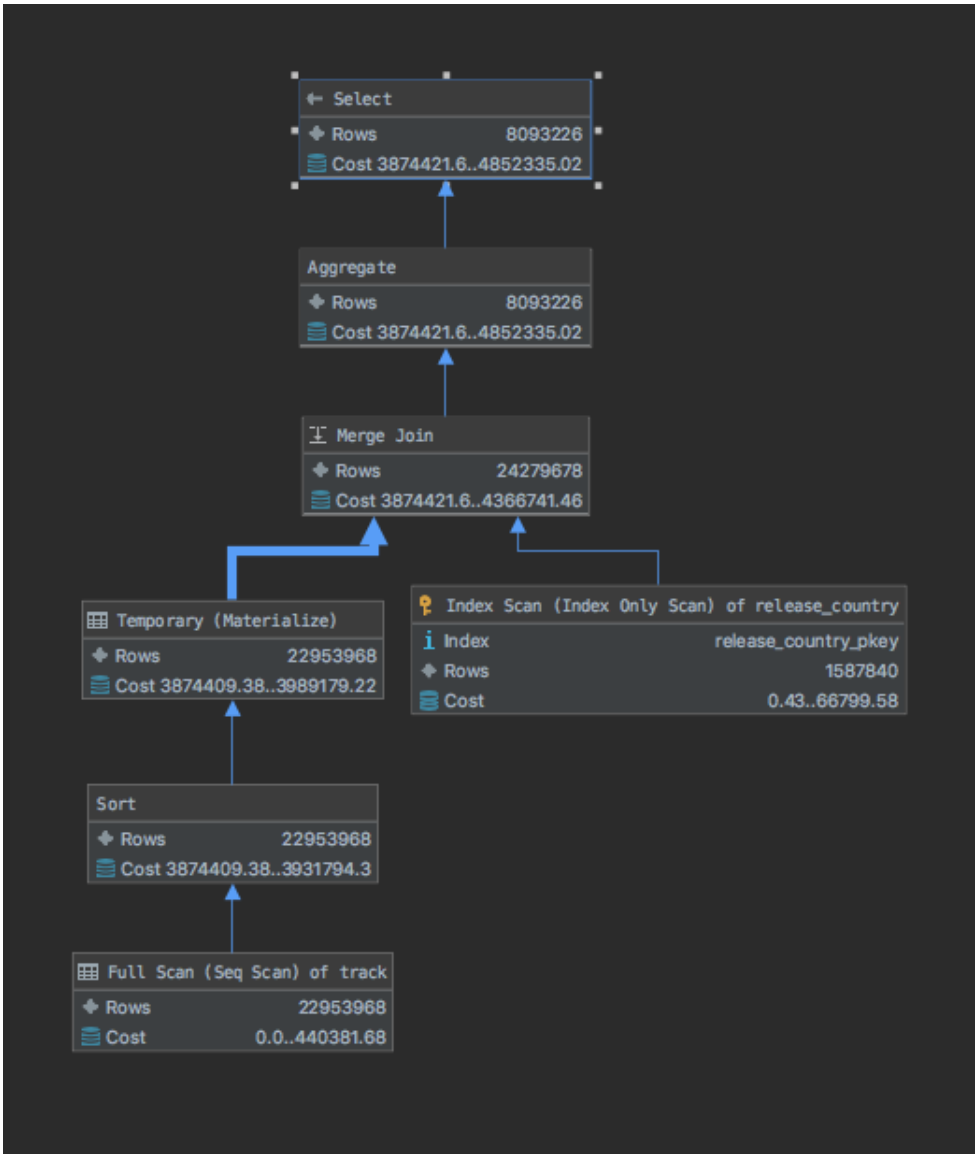

Figure 7 – Explain plan of Query 7

The complexity of the query is largely dominated by the sorting in the track table by the release number. This can be solved by creating a simple index by 'track.release'.

2.2.8 • Query 8

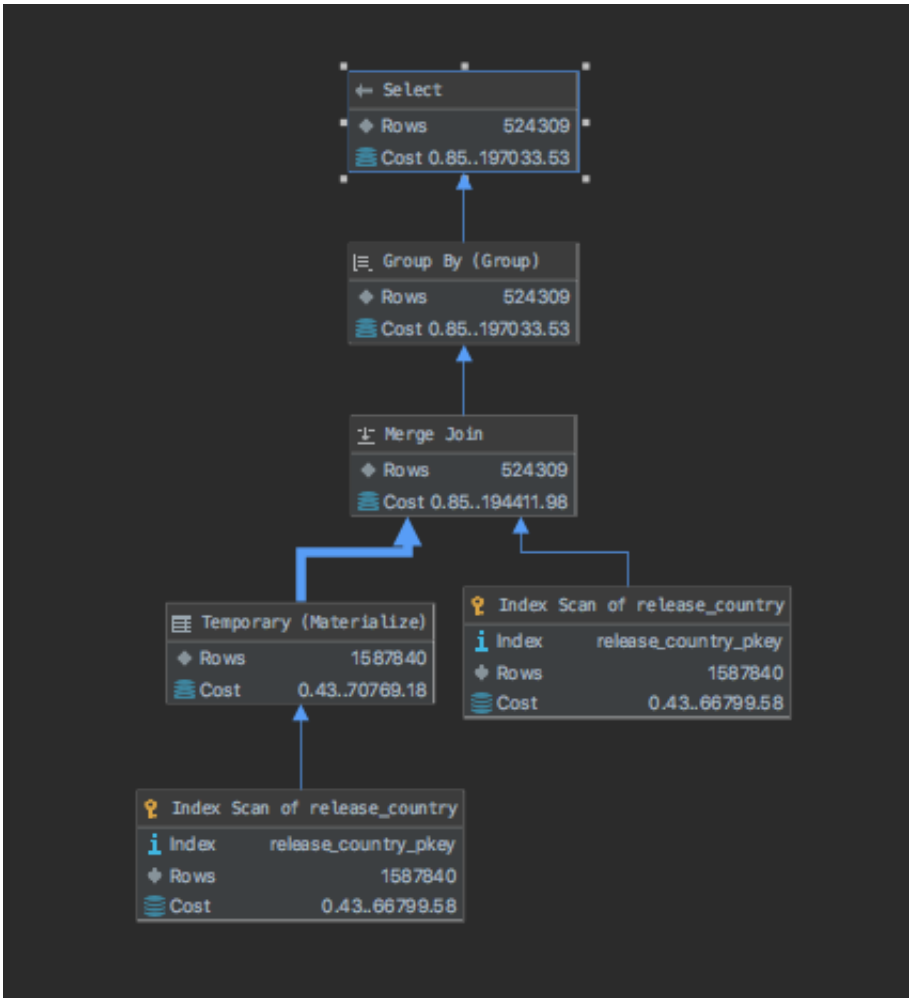

Figure 8 – Explain plan of Query 8

Query 8 has its cost well distributed and optimized, since it scans the 'release_country' table twice using its primary key (release, country). Next, the Join and Group By process is simplified.

2.2.9 • Query 9

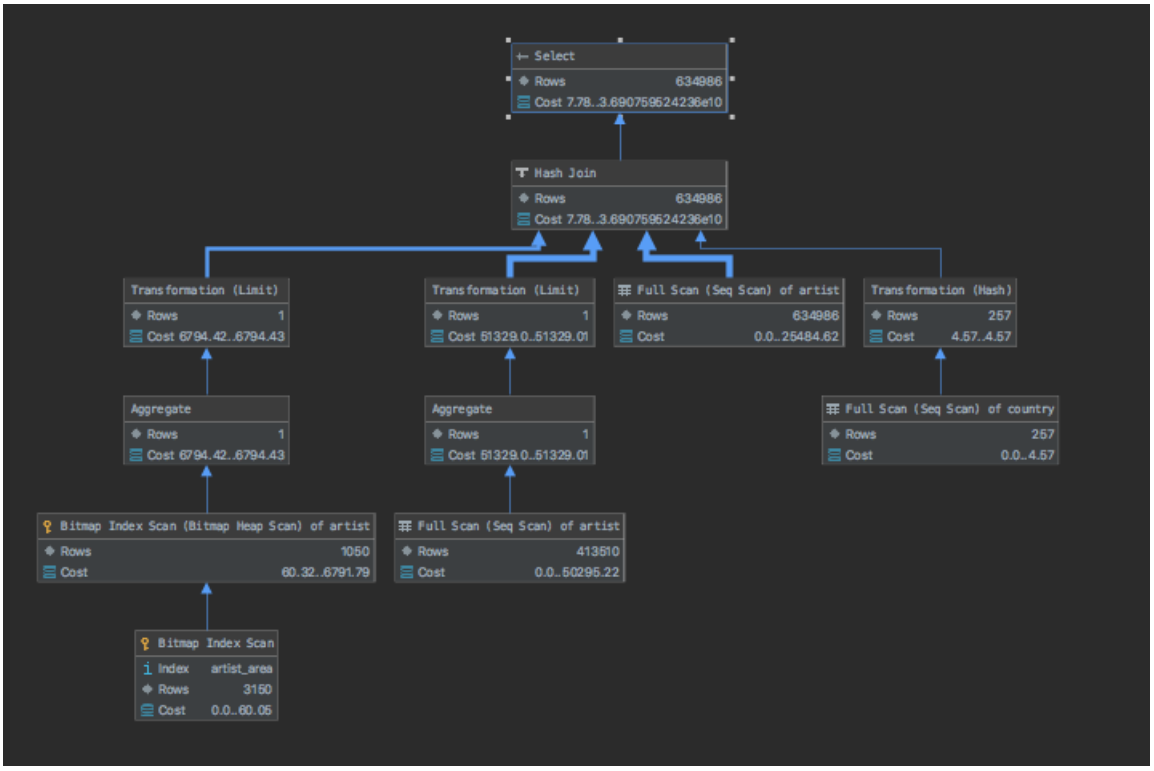

Figure 9 – Explain plan of Query 9

Query 9 is the one with the most room for improvement. Its greater complexity occurs in the countless times that a Full Scan of the Artifact table is made, to find nb and nb_global. Thus, a solution to optimize this query is to create a VIEW that orders the table by order of birth, so the Full Scan would become an Index Scan.

3
Performance improvement

For us it was clear that we should choose queries 7 and 9 to optimize, since they were our queries with more opportunity for improvement, based on the analysis of exercise 2.

Queries	Time1(ms)	Time2(ms)	Time3(ms)	Time4(ms)	Time5(ms)
7	1602	48	44	72	73
9	2124	1284	1155	1120	1984

Table 2 – Execution time of each query in milliseconds

3.1 Query 7

Keeping the same query with the index in the track table in 'track.release' the query had an extremely superior performance, going from execution times in the range of 25s to below 100 ms. Which leads us to the criticism that some simple things can have great effects on the performance of SQL queries.

```
CREATE INDEX track_release
ON track (release);
```

3.2 Query 9

With the creation of the new ArtistsOrdered table (by birth date) the Full Scan done before to calculate nb_global becomes an index scan. Using also the PARTITION method we can do the same for nb. This way we reduce a query with the wait around two minutes to around two seconds!

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

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;
```