Data Storage and Retrieval

Fall 2018

Diana Sujanto

Assignment 8

November 2, 2018

Show all your work!

1. Relational algebra and query plans.
2. Write the MillionSongs database query

select artist\_name, title, year

from tracks t, artists a

where t.artist\_id = a.artist\_id

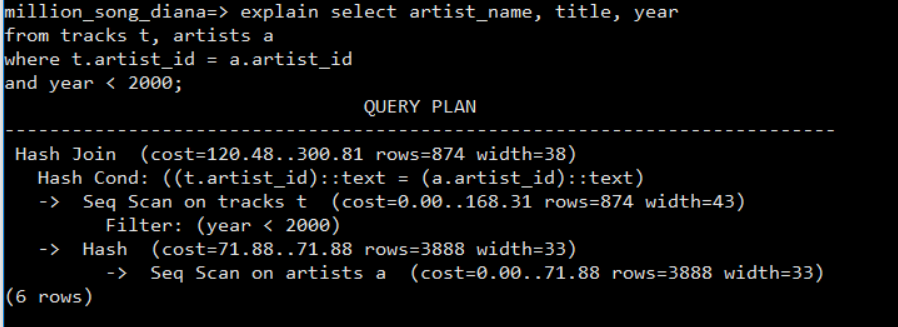
and year < 2000;

in relational algebra.

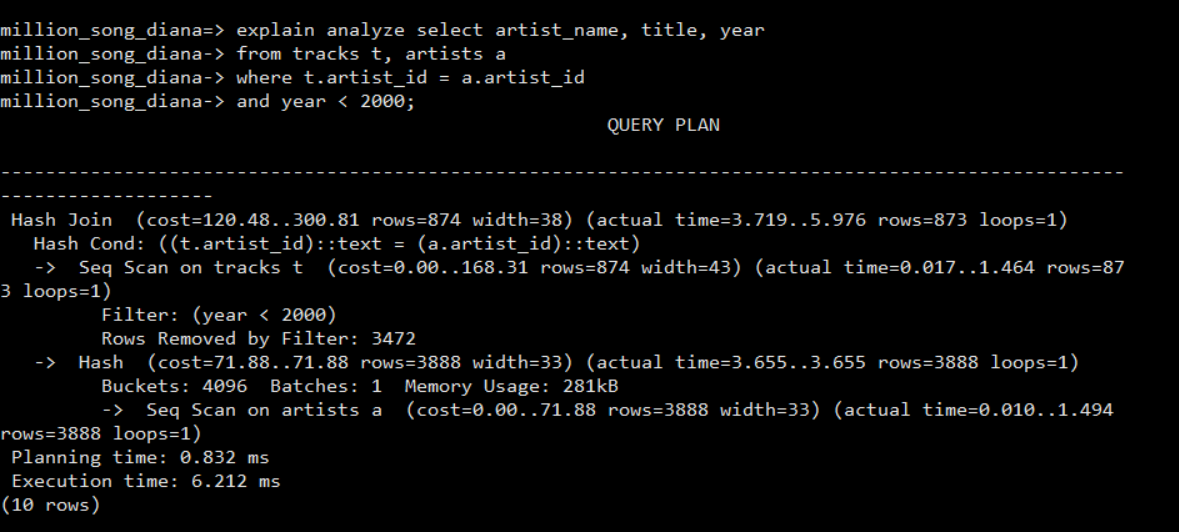
∏ artist\_name, title, year(σ year < 2000 (tracks Related image artists))

1. Show two query plans, including the one Postgres chooses.

The EXPLAIN plan shows how the table(s) referenced by the statement will be scanned — by plain sequential scan, index scan, etc. — and if multiple tables are referenced, what join algorithms will be used to bring together the required rows from each input table. It shows the estimated cost that the planner's guess at how long it will take to run the statement.

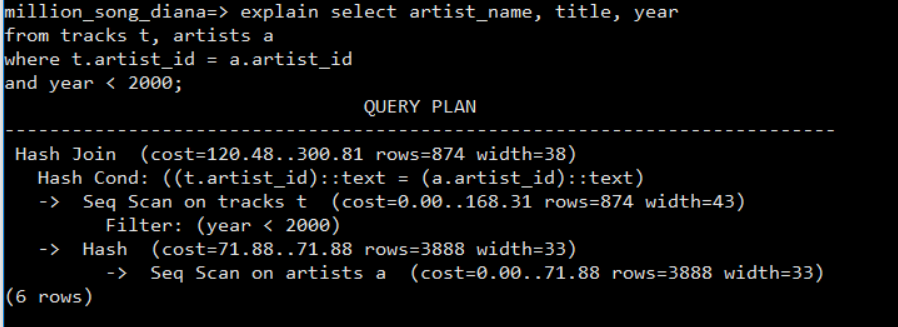


With the ANALYZE option, the statement is actually being executed by Postgres (not only planned).

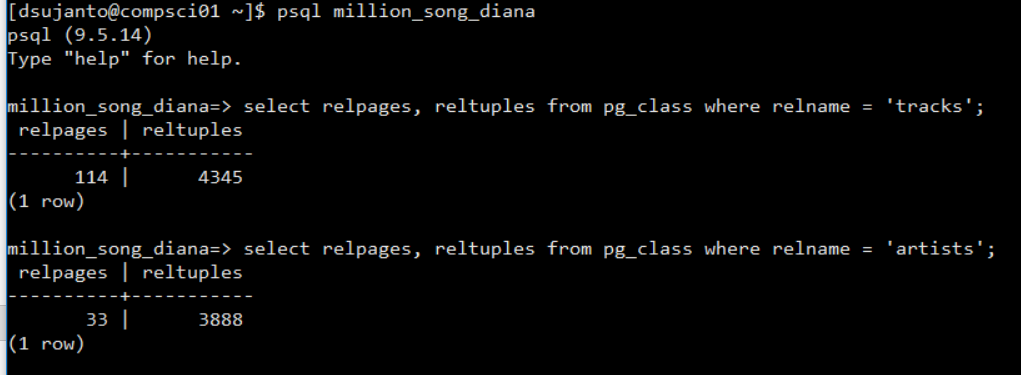


1. Calculate the costs of two query plans using Postgres’s default cost config parameters.

With the EXPLAIN only:



First, I get the disk pages read(relpages) on tracks and artists tables from the pg\_class table. At the same time, I get the number of rows(reltuples) in those tables.



Then, I calculate the cost of the query plan using explain and explain analyze:

Hash (artists )

= (disk pages read \* seq\_page\_cost) + (rows scanned \* cpu\_tuple\_cost)

= (33\*1.0) + (3888\*0.01) = 71.88

Seq Scan (tracks)

= (disk pages read \* seq\_page\_cost) + (rows scanned \* cpu\_tuple\_cost)

+ (rows scanned \* cpu\_operator\_cost)

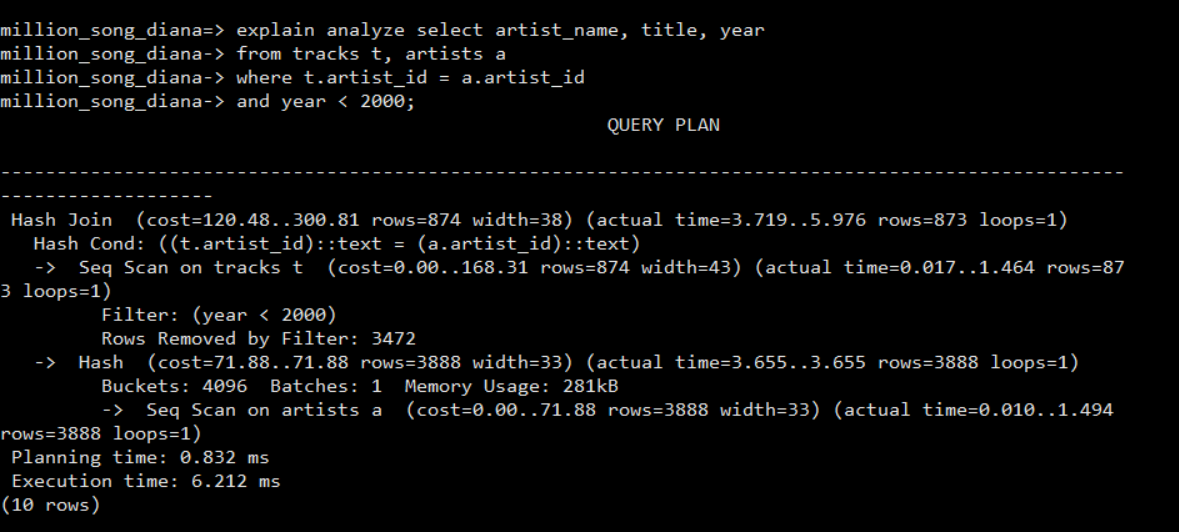
= (114\*1.0) + (4345\*0.01) + (4345\*0.0025)

= 168.31

HashJoin = 300.81

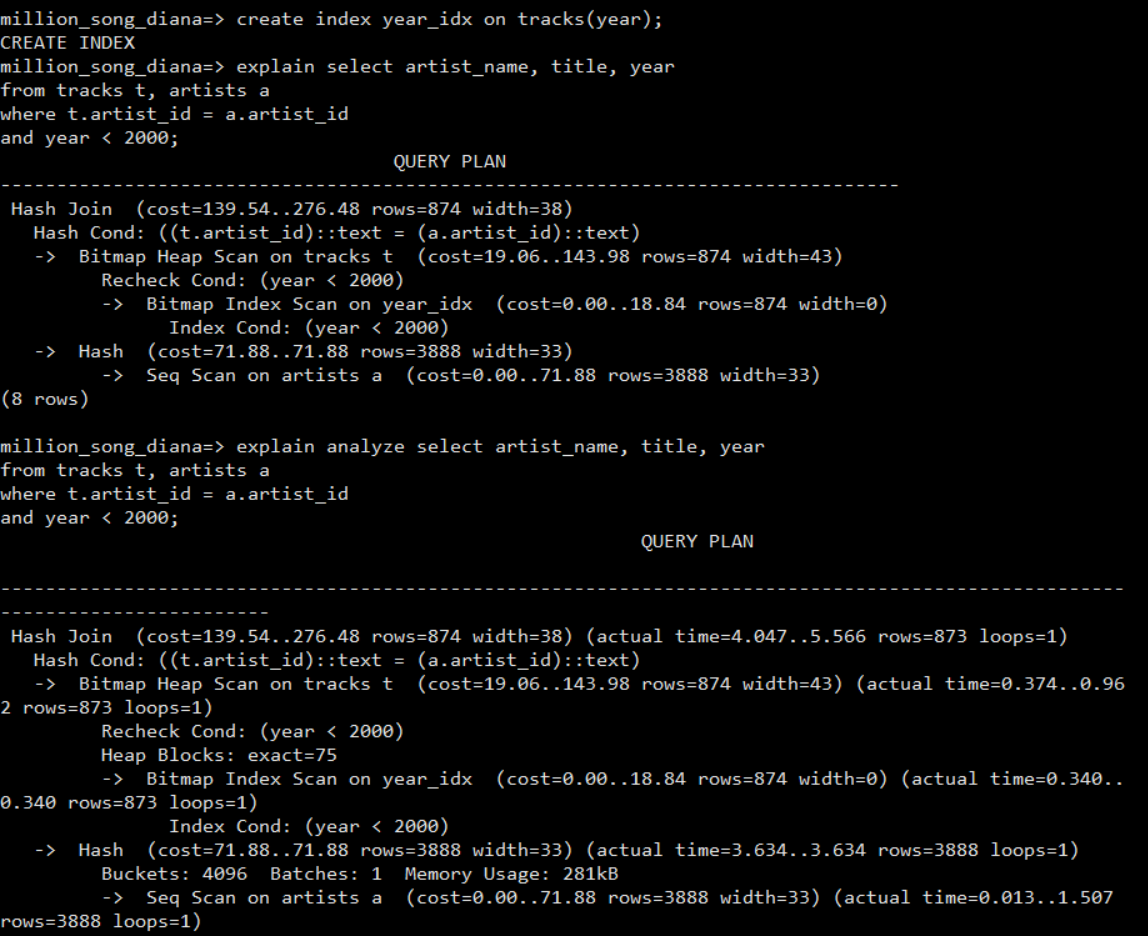
Total = 71.88 + 168.31 + 300.81 = 541

With the EXPLAIN ANALYZE:



1. For the query plan Postgres chose, compare your cost to what explain says.

The cost of the query plan that Postgres chose turns out to be the same as the one that explain says. Therefore, I decided to create an index to see how it would change.



As you can see, the total cost is less than the ones without the index.

Total cost = 276.48 + 18.84 +71.88 = 367.2

1. Elmasri & Navathe, exercise 18.14.

Consider SQL Queries: Q1, Q8, Q1B, and Q4 in Chapter 6 and Q27 in Chapter 7.

* 1. Draw at least 2 query trees that can represent each of these queries. Under what circumstances would you use each of your query trees?
  2. Draw the initial query tree for each of these queries, then show how the query tree is optimized by the algorithm outline in section 18.7
  3. For each query, compare your own query trees of part a and the initial and final query tree of part b.

(Note: I put the answers for this question altogether because it makes more sense to do the initial (canonical) tree first, before the optimization)

Query tree from Chapter 6 Q1:

SELECT Fname, Lname, Address

FROM EMPLOYEE, DEPARTMENT

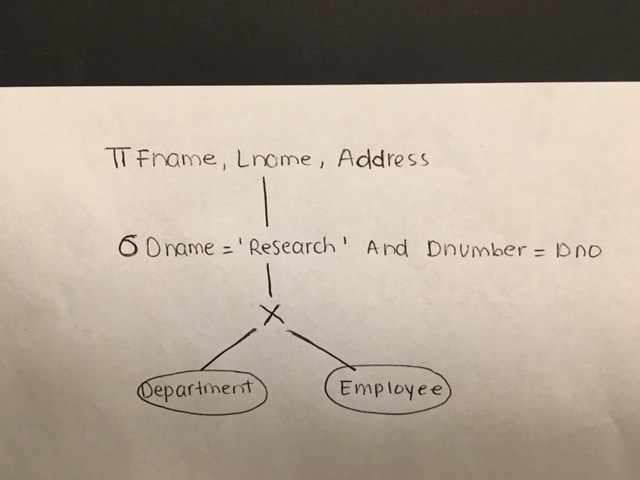
WHERE Dname = ‘Research’ AND Dnumber = Dno;

**Step 1: Converting Sql query to Relational algebra expression**

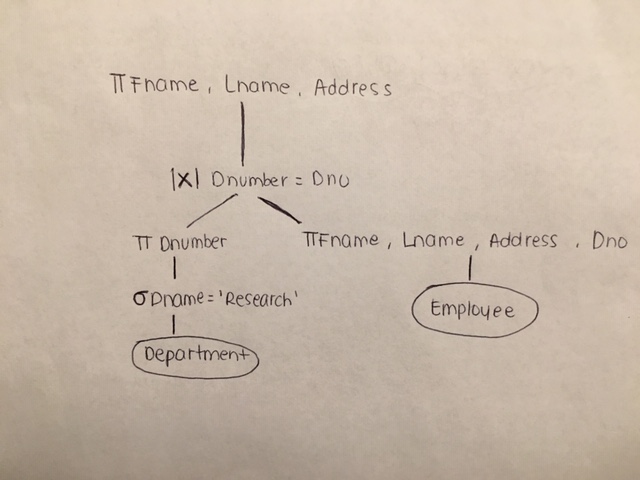
∏ Fname, Lname, Address(σ Dname = ‘Research’(EMPLOYEE Related imageDEPARTMENT))

**Step 2: Converting this Relation algebra expression to query tree**

Initial (canonical) query tree:

****

**Optimized Query Tree:**



Query tree from Chapter 6 Q8:

SELECT E.Fname, E.Lname, S.Fname, S.Lname

FROM EMPLOYEE AS E, EMPLOYEE AS S

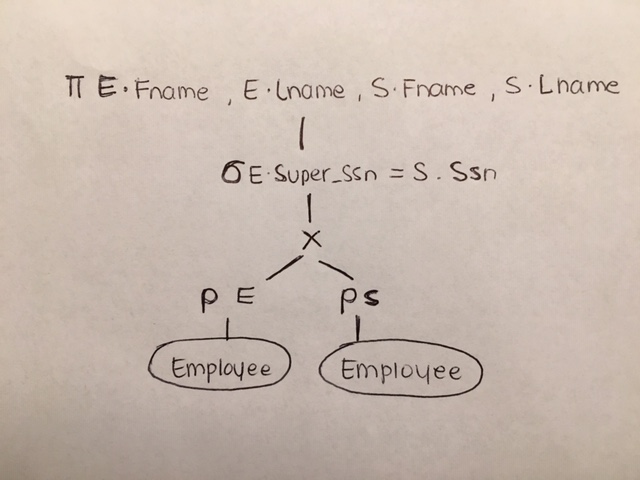
WHERE E.Super\_ssn = S.Ssn;

**Step 1: Converting Sql query to Relational algebra expression**

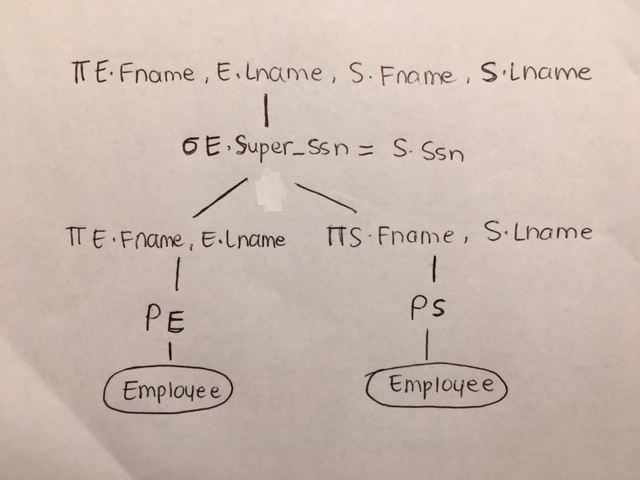
∏ E.Fname, E.Lname, S.Fname, S.Lname(σ E.Super\_ssn= S.Ssn(Related imageEMPLOYEE))

**Step 2: Converting this Relation algebra expression to query tree**

Initial (canonical) query tree:



**Optimized Query Tree:**



The reason we optimize the query tree is to provide faster query retrievals, which makes the application seem faster to the user. Secondly, the system can process more queries in the same amount of time, because each request takes less time than unoptimized queries. It also allows the server to run more efficiently (e.g. lower power consumption, less memory usage).

1. For Week 07 Lecture slide 10, show the sequence of b-trees that led to the one shown, for the given sequence of numbers inserted.

The Rules for B- Tree Insertion:

Start: one empty root node

Insertion into full root:

* 1. keep middle value
  2. spawn two child nodes, each 50% full

Insertion into full non-root node:

a. move middle entry to parent

b. split into self and sibling nodes, each 50% full

c. split may propagate to root, create another new level

To insert the following nodes: 8, 5, 1, 7, 3,12, 9, 6

