Contents

1 Basic Test Results

```
Extracting Archive:
Archive:  /tmp/bodek.Wpquy9/db/ex3/mohammadgh/presubmission/submission
  inflating: ex3.pdf
  inflating: q1.sql
  inflating: q2.sql
  inflating: q3.sql
  inflating: q4.sql
  inflating: q5.sql
  inflating: q6.sql
  inflating: README

************************************************************
** Testing that all necessary files were submitted:
README:
    SUBMITTED
ex3.pdf:
    SUBMITTED
q1.sql:
    SUBMITTED
q2.sql:
    SUBMITTED
q3.sql:
    SUBMITTED
q4.sql:
    SUBMITTED
q5.sql:
    SUBMITTED
q6.sql:
    SUBMITTED

************************************************************
** Checking for correct README format:
Output:
CREATE TABLE
CREATE TABLE
CREATE TABLE

Inserting conferences.csv
Output:
COPY 75

Inserting institutions.csv
Output:
COPY 401

Inserting authors.csv
Output:
COPY 3810

Note: The output is capped at 100 characters.

 Running q1.sql
Output:
    region    | countrycount
--------------+--------------
 africa       |            2
 asia         |           18
 australasia  |            2
 canada       |            1
```

```
 europe       |            1
 europe       |           27
 southamerica |            4
(7 rows)



 Running q2.sql
Output:
    region    |         insavg
--------------+------------------------
 africa       |     1.5000000000000000
 asia         |     6.0000000000000000
 australasia  |     9.5000000000000000
 canada       |    32.0000000000000000
 europe       | 1.0000000000000000000
 europe       |     8.0740740740740741
 southamerica |     5.0000000000000000
(7 rows)



 Running q3.sql
Output:
            name
-----------------------------
 Abhradeep Thakurta
 Aditya Akella
 Adrian Perrig
 Ahmed Eldawy
 Alastair F. Donaldson
 Alejandro Russo
 Alessandro Orso
 Alexander Aiken
 Alex C. Snoeren
 Ali Mesbah 0001
 Anand Sivasubramaniam
 Andreas Moshovos
 Andreas Podelski
 Andrew C. Myers
 Anthony K. H. Tung
 Antonio Filieri
 Antony L. Hosking
 Benjamin C. Pierce
 Bingsheng He
 Bin Ren
 Björn Franke
 Boris Glavic
 Carlos Maltzahn
 Chris Parnin
 Christian S. Jensen
 Christopher W.

 Running q4.sql
Output:
    region    | country | totalcount
--------------+---------+------------
 africa       | za      |        2.0
 asia         | cn      |      498.0
 australasia  | au      |      121.0
 canada       | ca      |      219.0
 europe       | it      |        4.0
 europe       | de      |      324.0
 southamerica | br      |       18.0
(7 rows)
```

```
 Running q5.sql
Output:
                name
-----------------------------------
 Aapo Hyvärinen
 Aaron Roth 0001
 Aaron Schulman
 Aaron Sidford
 Aarti Gupta
 Aarti Singh
 Abdallah Saffidine
 Abhik Roychoudhury
 Abhinav Gupta 0001
 Abhi Shelat
 Abhradeep Thakurta
 Abraham Bernstein
 Abusayeed Saifullah
 Achi Brandt
 Achim Jung
 Adam D. Smith
 Adam Herout
 Adam Klivans
 Aditya Akella
 Aditya G. Parameswaran
 Aditya Grover
 Aditya Kanade
 Aditya P. Mathur
 Adriana Iamnitchi
 Adrián Jarabo
 Adrian Perrig


 Running q6.sql
Output:
             name
------------------------------
 Aaron Sidford
 Adam D. Smith
 Adrian Vetta
 Alexander A. Sherstov
 Alexander Rakhlin
 Angelika Steger
 Benny Chor
 Bernard Chazelle
 Boaz Barak
 Byung-Gon Chun
 C. Greg Plaxton
 Chang Xu 0002
 Charles X. Ling
 Chien-Ju Ho
 Daniel C. Alexander
 David Steurer
 David Zuckerman
 Derek Nowrouzezahrai
 Dimitris Fotakis
 Eli Upfal
 Glencora Borradaile
 Guillermo Sapiro
 Hao Chen 0003
 Hong Cheng 0001
 Huy L. Nguyen
 Jack Snoeyink
 Jie Shao

```

2 README

```
1 mohammadgh,muaz.abdeen
```

(67506) Databases – Spring 2022 – Exercise (3)

Muaz Abdeen 300575297

Mohammad Ghanayem 208653220

Question (2): Indexing

A := authors (<u>name</u>, <u>conference</u>, <u>year</u>, institution, count, adjustedcount)

C := conferences (<u>conference</u>, area, subarea)

I := institutions (<u>institution</u>, region, country)

1.

(a)

```
SELECT DISTINCT year
FROM authors
WHERE institution='Hebrew University of Jerusalem';
```

(b)

```
                                   QUERY PLAN
-------------------------------------------------------------------------------
HashAggregate  (cost=3904.75..3905.28 rows=53 width=4)
               (actual time=19.875..19.905 rows=43 loops=1)
     Group Key: year
     ->  Seq Scan on authors  (cost=0.00..3902.30 rows=978 width=4)
                              (actual time=0.428..19.060 rows=929 loops=1)
    Filter: ((institution)::text = 'Hebrew University of Jerusalem'::text)
    Rows Removed by Filter: 163895
 Planning Time: 0.100 ms
 Execution Time: 19.964 ms
(7 rows)
```

Here the planner has decided to use a two-step plan: the child plan node scans the entire table sequentially, this is the meaning of `Seq Scan`, and then the upper plan node `HashAggregate` groups the records into a temporary hash table using the attribute `year` as a `Group Key`.

The grouping cost estimate `(cost=3904.75..3905.28 rows=53 width=4)` means that Postgres expects that the grouping will start at cost `3904.75` and finishes at `3905.28`, the difference is the estimated cost in an arbitrary unit of computation to perform this operation. `rows` is the estimated number of rows this `HashAggregate` will return, and `width` is the estimated size in bytes of the returned

rows. The actual estimate (`actual time=0.071..0.074 rows=4 loops=1`) tell us actual time involved in execution. Here we have a new value `loop` which says the entire table was scanned one time.

The `Seq Scan` estimated cost (`cost=0.00..3902.30 rows=978 width=4`) is `3902.30` units of computation, starting at `0.00`. where the actual cost was (`actual time=0.428..19.060 rows=929 loops=1`). The `WHERE` clause has been applied as a "`Filter`" condition attached to the `Seq Scan` plan node. This means that the plan node checks the condition for each row it scans, and outputs only the ones that pass the condition.

(c) `CREATE INDEX ON authors(institution);`

(d)

```
                                   QUERY PLAN
-------------------------------------------------------------------------------------
HashAggregate  (cost=1636.41..1636.94 rows=53 width=4)
               (actual time=1.694..1.728 rows=43 loops=1)
      Group Key: year
      ->  Bitmap Heap Scan on authors  (cost=32.00..1633.96 rows=978 width=4)
                                (actual time=0.107..0.882 rows=929 loops=1)
            Recheck Cond: ((institution)::text = 'Hebrew University of
                           Jerusalem'::text)
            Heap Blocks: exact=53
            ->  Bitmap Index Scan on authors_institution_idx
                     (cost=0.00..31.76 rows=978 width=0)
                     (actual time=0.093..0.093 rows=929 loops=1)
                     Index Cond: ((institution)::text = 'Hebrew University of
                                  Jerusalem'::text)
Planning Time: 0.124 ms
Execution Time: 1.799 ms
```

Here the planner has decided to use a three-step plan: the child plan node visits an index to find the locations of rows matching the index condition, and then the intermediate plan node actually fetches those rows from the table itself. Fetching rows separately is much more expensive than reading them sequentially, but because not all the pages of the table have to be visited, this is still cheaper than a sequential scan. The most upper plan is the grouping one as before.

Notice that the actual time using our defined index is `1.799 ms`, which is almost tenth of the cost of the query without this index `19.964 ms`.

2.a. (1) Without index we have to scan the entire table sequentially, that is to read all the blocks where the table is stored. Each block contains $\left\lfloor \frac{2000}{180} \right\rfloor = \mathbf{11}$ rows, then the `authors` table takes $\left\lceil \frac{12000}{11} \right\rceil = \mathbf{1091}$ blocks to store.

So, the query cost is: **1091**.

(2) Optimal branching factor is: $d = \left\lfloor \frac{b+s}{p+s} \right\rfloor = \left\lfloor \frac{2000+8}{8+8} \right\rfloor = \mathbf{125}$.

(3) Step 1: Find first relevant leaf: $\left\lceil \log_{\lfloor \frac{d}{2} \rfloor} N \right\rceil = \lceil \log_{[63]} 12{,}000 \rceil = \mathbf{3}$.

Step 2: Read all relevant leaves: since count values uniformly distributed over [1,20] range then there are: $\frac{12{,}000}{20} = \mathbf{600}$ rows corresponding to `count=2`, then there are $\left\lceil \frac{m}{\lfloor \frac{d}{2} \rfloor - 1} \right\rceil = \left\lceil \frac{600}{62} \right\rceil = \mathbf{10}$ leaves matching these rows.

Total cost = **13**.

2.b. (1) As the previous section, we have to traverse the entire tree, read all blocks, the query cost is: **1091**.

(2) The same as the previous section, d = **125**.

(3) In addition to the step 1 and step 2 in the previous section, there is a step 3 here: to access the matching rows, we have to read all the **600** blocks containing the matching rows. The total = **613**.

2.c. (1) Scan the entire table: **1091**.

(2) Optimal branching factor is: $d = \left\lfloor \frac{b+s}{p+s} \right\rfloor = \left\lfloor \frac{2000+26}{8+26} \right\rfloor = \mathbf{59}$.

(3) First, we traverse the tree down to the first matching leaf: $\lceil \log_{\lceil 30 \rceil} 12{,}000 \rceil = \mathbf{3}$. Second, we traverse all matching leaves, notice the table have at most $\left\lceil \frac{12{,}000}{20 \times 80} \right\rceil = \mathbf{7}$ matching rows (maybe less, depends on how many `year=1999` values are there), which fit in $\left\lceil \frac{7}{29} \right\rceil = \mathbf{1}$ one leaf.

Total cost $= \mathbf{4}$.

2.d. (1) Optimal branching factor is: $d = \left\lfloor \frac{b+s}{p+s} \right\rfloor = \left\lfloor \frac{2000+22}{8+22} \right\rfloor = \mathbf{67}$.

(2) We have WHERE with a disjoint condition "or", so we have to scan the tree two times, one for each value, the calculations for both are the same.

Traversing the tree down costs: $\lceil \log_{\lceil 34 \rceil} 12{,}000 \rceil = \mathbf{3}$, traversing the matching leaves costs: $\left\lceil \frac{150}{33} \right\rceil = \mathbf{5}$, where $150 = \left\lceil \frac{12{,}000}{80} \right\rceil$ since the conference values uniformly distributed over 80 values. Now, since we are also indexing on name, there is no need to read the block containing the row.

The total cost for both conditions is $= 2 \times (3 + 5) = \mathbf{16}$.

2.e. (1) Optimal branching factor is: $d = \left\lfloor \frac{b+s}{p+s} \right\rfloor = \left\lfloor \frac{2000+8}{8+8} \right\rfloor = \mathbf{125}$.

(2) Tree traverse until leaf costs at most $\lceil \log_{\lceil 63 \rceil} 12{,}000 \rceil = \mathbf{3}$.

There are 19 `count` values to scan, which spans $\frac{12{,}000}{20} \times 19 = 11{,}400$ rows, so the matching rows will be in at most $\left\lceil \frac{11400}{62} \right\rceil = \mathbf{184}$ leaves.

There are approximately 11,400 matching rows, each may be in a different block, but we will go over each block of the table at most once = **1091** .

Total $= 3 + 184 + 1091 = \mathbf{1278}$.

4 q1.sql

```
1  SELECT region, COUNT(DISTINCT country) AS countryCount
2  FROM institutions
3  GROUP BY region
4  ORDER BY region;
```

5 q2.sql

```
1  SELECT region, COUNT(DISTINCT institution) * 1.0 / COUNT(DISTINCT country) AS insAvg
2  FROM institutions
3  GROUP BY region
4  ORDER BY region;
```

6 q3.sql

```
1  SELECT DISTINCT name
2  FROM authors
3  WHERE conference in (SELECT conference
4               FROM conferences
5               WHERE area='systems')
6  GROUP BY name
7  HAVING SUM(count) > 1
8  EXCEPT
9  SELECT DISTINCT name
10 FROM authors
11 WHERE conference in (SELECT conference
12              FROM conferences
13              WHERE area='systems')
14 GROUP BY name
15 HAVING MAX(year) < 2014
16 ORDER BY name;
```

7 q4.sql

```
1  WITH PaperCounts(region, country, totalCount) as
2       (SELECT region, country, SUM(count) + 0.0
3        FROM institutions NATURAL JOIN authors
4        GROUP BY region, country)
5  SELECT P1.region, P1.country, P1.totalCount
6  FROM PaperCounts P1
7  WHERE P1.totalCount >= all (SELECT P2.totalCount
8                              FROM PaperCounts P2
9                             WHERE P1.region = P2.region)
10 ORDER BY region, country;
```

8 q5.sql

```
1  WITH NotSpecConfs(conference) AS
2       (SELECT conference
3        FROM authors
4        GROUP BY conference
5        HAVING COUNT(DISTINCT year) < 10)
6  SELECT DISTINCT name
7  FROM authors
8  EXCEPT
9  SELECT DISTINCT name
10 FROM authors
11 WHERE conference IN (SELECT * FROM NotSpecConfs)
12 ORDER BY name;
```

9 q6.sql

```
1  WITH RECURSIVE T(name, dist) AS (
2          VALUES('Noam Nisan', 0)
3          UNION
4          SELECT second, dist + 1
5          FROM T INNER JOIN (SELECT A1.name AS first, A2.name AS second
6                             FROM authors A1 INNER JOIN authors A2
7                                 ON (A1.name != A2.name and
8                                   A1.conference = A2.conference and
9                                     A1.year = A2.year)) D
10           ON (T.name = D.first)
11         WHERE dist < 2
12 )
13 SELECT DISTINCT name FROM T ORDER BY name;
```