National University of Singapore
School of Computing

CS3223: Database Systems Implementation
Semester 2, 2022/23

Assignment 2 (10 marks)

Due: 11:59pm, March 10, 2023 (Friday, 2359)

1 Introduction

In this assignment, you will examine the query plans for evaluating selection queries in PostgreSQL 15.0. This is an individual assignment to be submitted by each student.

The assignment involves writing SQL scripts to execute specific query plans for five selection queries. The queries will be issued using psql, PostgreSQL's terminal-based client.

Late submission penalty: There will be a late submission penalty of 1 mark per day up to a maximum of 3 days. If your assignment is late by more than 3 days, it will not be graded and you will receive 0 marks for the assignment.

1.1 Evaluation Strategies for Selection Queries

PostgreSQL supports the following evaluation strategies for selection queries:

1. **Sequential Scan**. This is also known as the table scan method which simply scans every data page for matching records.

2. **Index Scan**. This is an index scan followed by RID lookups. The RIDs are not sorted before the lookups.

3. **Index-Only Scan**. This is an index scan without RID lookups.

4. **Bitmap Index Scan**. This is almost equivalent to the Index Scan method except that the RIDs of matching records are sorted before the lookups. PostgreSQL implements the RID sorting by using an in-memory bitmap array with each bit corresponding to a record in the indexed relation. All the bits in the array are initialized to zeroes before the index scan, and the bits corresponding to matching records are set to ones during the index scan. At the end of the index scan, the constructed bitmap is used to retrieve the matching records (which are stored in a heap file). PostgreSQL refers to this partial scanning of the heap file via the bitmap as *Bitmap Heap Scan*.

   The memory allocated for storing the bitmap array is controlled by the *work_mem* parameter which has a default value of $4MB$. If the cardinality of the indexed relation is too large for its bitmap array to be stored entirely in the allocated memory, PostgreSQL will instead construct an approximate bitmap array where some of the bits may represent data pages instead of data records. Specifically, a ''lossy'' bit in the bitmap corresponds

to a data page such that the bit is set to one if and only if there is some matching record residing on that data page. During RID lookups, for each retrieved data page that corresponds to a ''lossy'' bit, PostgreSQL will need check each record in that page to identify the matching data records.

5. **BitmapAnd Index Scan**. This method is used for evaluating a conjunction of multiple terms in a selection predicate (e.g., (b = 10) and (c = 20)). Each term is first evaluated using an appropriate Bitmap Index Scan, and the collection of constructed bitmaps are then ANDed to generate a new bitmap representing the matching records for the entire selection predicate. This is followed by a Bitmap Heap Scan using the new bitmap.

6. **BitmapOr Index Scan**. This method is used for evaluating a disjunction of multiple terms in a selection predicate (e.g., (b = 10) or (c = 20)). Each term is first evaluated using an appropriate Bitmap Index Scan, and the the collection of constructed bitmaps are then ORed to generate a new bitmap representing the matching records for the entire selection predicate. This is followed by a Bitmap Heap Scan using the new bitmap.

1.2 Exploring Query Plans

In this assignment, you will examine the query plans for selection queries on a relation $r(\underline{a}, b, c, d)$, where attribute a is the relation's primary key. Besides the implicitly created B$^+$-tree index on the primary key (named r_pkey), there are three additional B$^+$-tree indexes on relation r:

- *b_idx*: index on attribute b
- *c_idx*: index on attribute c
- *cb_idx*: index on attributes (c, b)

Table 2 lists some of feasible query plans for the selection queries considered in this assignment.

In this section, we examine the query plans for the following query Q:

```
SELECT * FROM r WHERE b = 20
```

To examine the optimal query plan selected by the optimizer for a query, use the EXPLAIN command.

```
assign2=#  explain select * from r where b=20;
                          QUERY PLAN
-----------------------------------------------------------------------
Bitmap Heap Scan on r (cost=452.39..15661.55 rows=40253 width=32)
       Recheck Cond:  (b = 20)
       -> Bitmap Index Scan on b_idx (cost=0.00..442.32 rows=40253 width=0)
           Index Cond:  (b = 20)
(4 rows)
```

Query Plan	Description
SS	Sequential scan
IS-b	Index scan on b_idx
IS-c	Index scan on c_idx
IS-cb	Index scan on cb_idx
IOS-b	Index-only scan on b_idx
IOS-c	Index-only scan on c_idx
IOS-cb	Index-only scan on cb_idx
BIS-b	Bitmap index scan on b_idx
BIS-c	Bitmap index scan on c_idx
BIS-cb	Bitmap index scan on cb_idx
ABIS-b-c	BitmapAnd index scan on b_idx & c_idx

Table 1: Possible evaluation plans for selection queries on attributes b and c

The result indicates that the query plan with the lowest estimated cost for Q is BIS-b with an estimated evaluation cost of $15,661.55$ units and an estimated number of $40,253$ rows in query result.

To find out both the estimated and actual costs of a query's optimal plan, use the EXPLAIN ANALYZE command. This command will execute the specified query (without displaying its query result) to measure its actual running time along with other statistics.

```
assign2=#  explain analyze select * from r where b=20;
                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------
Bitmap Heap Scan on r (cost=452.39..15661.55 rows=40253 width=32) (actual time=9.515..75.289 rows=40000 loops=1)
     Recheck Cond:  (b = 20)
     Heap Blocks:  exact=13744
     -> Bitmap Index Scan on b_idx (cost=0.00..442.32 rows=40253 width=0) (actual time=5.662..5.663 rows=40000 loops=1)
          Index Cond:  (b = 20)
Planning Time:  1.213 ms
Execution Time:  77.123 ms
(7 rows)
```

The above output shows that the total execution time for the query is 77.123 ms with $40,000$ rows in the query result.

1.3 Run-time Statistics

To find out the I/O cost for a query plan in terms of the number of index blocks and heap data blocks accessed, we can examine the run-time statistics[1] collected by PostgreSQL in the views named pg_statio_user_tables and pg_stat_user_tables.

[1]The statistics mentioned here are for the purpose of monitoring the database activity at run-time; these should be not confused with the statistics collected (e.g., using ANALYZE command) on relations for selectivity and cost estimations.

```
assign2=#  select dropdbbuffers('assign2');
```

dropdbbuffers
t

(1 row)

```
assign2=#  select pg_stat_reset();
```

pg_stat_reset

(1 row)

```
assign2=#  explain analyze select * from r where b=20;
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
Bitmap Heap Scan on r (cost=452.39..15661.55 rows=40253 width=32) (actual time=9.515..75.289 rows=40000 loops=1)
   Recheck Cond:  (b = 20)
   Heap Blocks:  exact=13744
   -> Bitmap Index Scan on b_idx (cost=0.00..442.32 rows=40253 width=0) (actual time=5.662..5.663 rows=40000 loops=1)
      Index Cond:  (b = 20)
Planning Time:  1.213 ms
Execution Time:  77.123 ms
(7 rows)
assign2=#  select select I.relname, I.heap_blks_read, I.heap_blks_hit, S.seq_scan, S.seq_tup_read
assign2=#  from pg_statio_user_tables I, pg_stat_user_tables S
assign2=#  where I.relid = S.relid and I.relname = 'r';
```

relname	heap_blks_read	heap_blks_hit	seq_scan	seq_tup_read
r	13744	0	0	0

(1 row)

```
assign2=#  select I.relname, I.indexrelname, I.idx_blks_read, I.idx_blks_hit,
assign2=#  S.idx_scan, S.idx_tup_read, S.idx_tup_fetch
assign2=#  from pg_statio_user_indexes I, pg_stat_user_indexes S
assign2=#  where I.relid = S.relid and I.indexrelid = S.indexrelid AND I.relname = 'r';
```

relname	indexrelname	idx_blks_read	idx_blks_hit	idx_scan	idx_tup_read	idx_tup_fetch
r	r_pkey	1	0	0	0	0
r	b_idx	38	0	1	40000	0
r	c_idx	1	0	0	0	0
r	cb_idx	1	0	0	0	0

(4 rows)

The statement ''select dropdbbuffers('assign2')'', which requires the dropdbbuffers extension to be installed, is used to clear all disk blocks belonging to the database 'assign2' from the buffer pool. The statement ''select pg_stat_reset()'' is used to reset all the statistics counters before the query is run. In the view pg_statio_user_tables, **heap_blks_read** refers to the number of accessed data heap blocks that are not found in the shared buffer pool (i.e., they require reading from the disk or OS cache), while **heap_blks_hit** refers to the number of accessed data heap blocks that are found in the shared buffer pool. Both **idx_blks_read** and **idx_blks_hit** in the view pg_statio_user_indexes, are defined similarly for accessed index blocks. In the view pg_stat_user_tables, **seq_scan** refers to the number of sequential scans initiated on the table, and **seq_tup_read** refers to the number of rows fetched by sequential scans. In the view pg_stat_user_indexes, **idx_scan** refers to the number of index scans initiated on the table, **idx_tup_read** refers to the number of index entries returned by scans of the index, and **idx_tup_fetch** refers to the number of rows fetched by simple index scans (i.e., non-bitmap index scans) using the index.

Note that the above executed query plan uses only the index b_idx; the idx_blk_read value of 1 shown for each of the other three indexes are due to the index retrievals by the query optimizer.

1.4 Restricting Query Plans

PostgreSQL provides several optimization-related parameters that can be configured at run-time to influence the choice of query plans selected by the query optimizer. For this assignment, the following four boolean parameters are relevant for selection queries:

- **enable_bitmapscan**: enables or disables the query optimizer's use of bitmap-scan plan types. The default value is on.
- **enable_indexscan**: enables or disables the query optimizer's use of index-scan plan types. The default value is on.
- **enable_indexonlyscan**: enables or disables the query optimizer's use of index-only-scan plan types. The default value is on.
- **enable_seqscan**: enables or disables the query optimizer's use of sequential scan plan types. It is impossible to suppress sequential scans entirely, but turning this variable off discourages the planner from using one if there are other methods available. The default value is on.

These parameters can be configured with the SET command. For example, the following example illustrates the effect of disabling bitmap-scan type plans for query Q.

```
assign2=# set enable_bitmapscan = off;
SET
assign2=# explain analyze select * from r where b=20;
                              QUERY PLAN
-----------------------------------------------------------------------------------------------------
Seq Scan on r (cost=0.00..39706.00 rows=40253 width=32) (actual time=0.026..164.192 rows=40000 loops=1)
        Filter: (b = 20)
        Rows Removed by Filter: 1960000
Planning Time: 2.469 ms
Execution Time: 165.702 ms
(5 rows)
```

Observe that the optimal plan selected now is SS.

The next example illustrates how we could further restrict the optimizer's search space by disabling the sequential scan plan.

```
assign2=# set enable_bitmapscan = off; set enable_seqscan = off;
SET
SET
assign2=# explain analyze select * from r where b=20;
                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------
Index Scan using b_idx on r (cost=0.43..59368.67 rows=40253 width=32) (actual time=0.271..54.212 rows=40000 loops=1)
    Index Cond: (b = 20)
Planning Time: 1.336 ms
Execution Time: 55.828 ms
(4 rows)
```

Observe that the optimal plan selected now is IS-b. Although the execution time of IS-b is lower than that of SS, the estimated cost of IS-b is higher than that of SS. This explains why SS was previously selected over IS-b.

To find out the values of configurable parameters, use the SHOW command.

```
assign2=#  show enable_seqscan; show enable_bitmapscan;
 enable_seqscan
 off
(1 row)

 enable_bitmapscan
 off
(1 row)
```

To display the values of all the configurable parameters, use ‘‘SHOW ALL’’.

Suppose that we want the optimizer to choose the plan BIS-cb for Query Q. To achieve this objective, we need to disable the use of the *b_idx* index. Although PostgreSQL does not provide an explicit command to disable the use of a specific index, a convenient way to achieve this effect is to temporarily drop the index by executing all the commands as part of a transaction. In this way, we can later restore the ‘‘disabled’’ index by simply aborting the transaction.

By default, each command issued within psql is executed as a single-command transaction. To execute a sequence of commands as a single transaction, use the BEGIN command to indicate the start of a transaction, and terminate the transaction with either the COMMIT command to commit the transaction or the ROLLBACK command to abort the transaction.

The following example shows how we can force the optimizer to choose the plan BIS-cb.

```
assign2=#  begin;
BEGIN
assign2=#  set enable_bitmapscan = on; set enable_seqscan = off;
SET
SET
assign2=#  drop index b_idx;
DROP INDEX
assign2=#  explain analyze select * from r where b=20;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
Bitmap Heap Scan on r (cost=22022.49..37231.65 rows=40253 width=32) (actual time=14.909..99.645 rows=40000 loops=1)
   Recheck Cond:  (b = 20)
   Heap Blocks:  exact=13744
   -> Bitmap Index Scan on cb_idx (cost=0.00..22012.43 rows=40253 width=0) (actual time=12.961..12.962 rows=40000 loops=1)
      Index Cond:  (b = 20)
Planning Time:  2.023 ms
Execution Time:  101.779 ms
(7 rows)
assign2=#  rollback;
ROLLBACK
```

Note that it is not always possible to force the optimizer to choose a specific query plan even if the plan is feasible for the query. For example, if the optimal plan for a query is BIS-cb, it is impossible to force the optimizer to choose the plan ABIS-b-cb.

2 Getting Started

Run the following commands to download the assignment files.

```
$ cd $HOME
$ wget http://www.comp.nus.edu.sg/~cs3223/assign/cs3223_assign2.zip
$ unzip cs3223_assign2.zip
$ cd cs3223_assign2
$ ls
```

The cs3223_assign2 sub-directory created in your home directory contains the following files: (1) a CSV data file named data.csv, (2) 10 bash scripts (install_pg.sh, setup.sh, createdb.sh, size.sh, sample.sh, a.sh, b.sh, c.sh, d.sh, e.sh), (3) two SQL scripts (sample-is-b.sql, sample-bis-b.sql), and (4) dropdbbuffers/ for the dropbuffers extension.

For this assignment, ensure that you are using the original version of PostgreSQL 15.0 (with the CLOCK replacement algorithm) and not the modified version (using LRU algorithm) from the previous assignment. If the directory ~/cs3223_assign1 from your first assignment is available, re-install the original version of PostgreSQL as follows:

```
$ cd ~/cs3223_assign1
$ make clock
```

On the other hand, if the directory ~/cs3223_assign1 from your first assignment is no longer available, re-install PostgreSQL as follows:

```
$ cd ~/cs3223_assign2
$ ./install_pg.sh
```

Next, run the following commands where $nnnn$ denote the port number used to start your PostgreSQL server. For this assignment, you will execute the server with 5000 pages in the buffer pool.

```
$ cd ~/cs3223_assign2
$ ./setup.sh
Modifying /home/alice/pgdata/postgresql.conf ...
Installing pgstattuple extension ...
Installing dropdbbuffers extension ...
$ pg_ctl start -l ~/log.txt -o "-p nnnn -B 5000"
waiting for server to start....  done
server started
$ export PGPORT=nnnn
$ ./createdb.sh
Creating table r & indexes b_idx, c_idx, cb_idx ...
./size.sh
select relpages, reltuples from pg_class where relname = 'r';
```

relpages	reltuples
14706	2e+06

```
(1 row)
create extension if not exists pgstattuple;
CREATE EXTENSION
select tree_level, internal_pages, leaf_pages, index_size from pgstatindex('b_idx'::regclass);
```

tree_level	internal_pages	leaf_pages	index_size
2	10	1686	13901824

```
(1 row)
select tree_level, internal_pages, leaf_pages, index_size from pgstatindex('c_idx'::regclass);
```

tree_level	internal_pages	leaf_pages	index_size
2	10	1685	13893632

```
(1 row)
select tree_level, internal_pages, leaf_pages, index_size from pgstatindex('cb_idx'::regclass);
```

tree_level	internal_pages	leaf_pages	index_size
2	12	1740	14360576

```
(1 row)
```

The script **setup.sh** disables parallel query plans with appropriate changes to your server's configuration file and also installs the two extensions (pgstattuple & dropdbbuffers) required in this assignment.

The script **createdb.sh** creates a database named assign2 which consists of the relation r and the three indexes (*b_idx*, *c_idx*, and *cb_idx*) described in Section 1.2.

The script **size.sh** shows information related to the size of the created table and indexes using PostgreSQL's catalog pg_class and the pgstatindex function from the pg_stattuple extension.

3 What to do

3.1 Required Readings

The following PostgreSQL documentation should be read before you start on this assignment.

1. How to read query plans generated by EXPLAIN command.
   http://www.postgresql.org/docs/current/using-explain.html

2. Manual page for EXPLAIN command.
   http://www.postgresql.org/docs/current/sql-explain.html

3.2 Assignment Tasks

This assignment consists of five parts (A to E). Each part (worth 2 marks) examines the query plans for a specific SQL query; the five queries are shown in Table 2.

Part	SQL Query
A	SELECT * FROM r WHERE c = 10
B	SELECT b FROM r WHERE c > 15
C	SELECT * FROM r WHERE b = 9 AND c = 10
D	SELECT * FROM r WHERE b > 9 AND c = 10
E	SELECT * FROM r WHERE b op_b v_b AND c op_c v_c

Table 2: Selection queries

For each part, you are to do the following: (1) for each specified query plan for a given query, write a SQL script (using appropriate configurations as discussed in Section 1.4) to enable the query to be executed using the specified query plan, (2) execute a bash script to execute each specified query plan and compute its average execution time, and (3) answer one question (on Canvas Assignment 2 Part 1) based on your execution results.

The submissions for this assignment consist of two parts to be submitted on Canvas:

- **Part 1**: Upload a zip file containing all your SQL scripts and output files to Assignment 2 Part 1.
- **Part 2**: Answer five questions on Assignment 2 Part 2.

Part A: Write two SQL scripts for **Query A** to be executed with the query plans IS-cb and BIS-cb. Your SQL scripts should be named **a-is-cb.sql** and **a-bis-cb.sql** for the two query plans IS-cb and BIS-cb, respectively. Run the script a.sh to generate two output files which are named after the SQL script files with the extension ‘‘.txt’’.

Part B: Write three SQL scripts for **Query B** to be executed with the query plans IS-c, BIS-c, and IOS-cb. Your SQL scripts should be named **b-is-c.sql**, **b-bis-c.sql**, and **b-ios-cb.sql** for the three query plans IS-c, BIS-c and IOS-cb, respectively. Run the script b.sh to generate three output files which are named after the SQL script files with the extension ‘‘.txt’’.

Part C: Write two SQL scripts for **Query C** to be executed with the query plans BIS-cb and ABIS-b-c. Your SQL scripts should be named **c-bis-cb.sql** and **c-abis-b-c.sql** for the two

query plans BIS-cb and ABIS-b-c, respectively. Run the script c.sh to generate two output files which are named after the SQL script files with the extension ‘‘.txt’’.

Part D: Write eight SQL scripts for **Query D** to be executed with the query plans SS, IS-b, IS-c, IS-cb, BIS-b, BIS-c, BIS-cb, and ABIS-b-c. Your SQL scripts should be named **d-ss.sql**, **d-is-b.sql**, **d-is-c.sql**, **d-is-cb.sql**, **d-bis-b.sql**, **d-bis-c.sql**, **d-bis-cb.sql**, and **d-abis-b-c.sql** for the eight query plans SS, IS-b, IS-c, IS-cb, BIS-b, BIS-c, BIS-cb and ABIS-b-c, respectively. Run the script d.sh to generate eight output files which are named after the SQL script files with the extension ‘‘.txt’’.

Part E: Write an instance of **Query E**, SELECT * FROM r WHERE b op_b v_b AND c op_c v_c, with appropriate comparison operators (for op_b and op_c) and integer constants (for v_b and v_c) such that the plan IS-b outperforms the plan BIS-cb in terms of the average execution time (over five exceutions) for your query instance.

Write two SQL scripts for your **Query E** instance to be executed with the query plans IS-b and BIS-cb. Your SQL scripts should be named **e-is-b.sql** and **e-bis-cb.sql** for the two query plans IS-b and BIS-cb, respectively. Run the script e.sh to generate two output files which are named after the SQL script files with the extension ‘‘.txt’’.

3.3 Sample Scripts

To help you with writing the SQL scripts, refer to the provided scripts named sample-is-b.sql and sample-bis-b.sql which specify the query plans IS-B and BIS-B, respectively, for the query SELECT * FROM r WHERE b=9. You can execute these query plans and compute their average execution times by running the script sample.sh.

4 What & How to Submit

The submissions for this assignment consists of two parts.

For the first part, create a zip file named e0123456.zip, where e0123456 is your NUS email ID. The zipped directory should contain 34 files (the 17 SQL script files that you have written and the 17 output text files produced by the execution of these 17 query plans). Upload the zip file to **Assignment 2 Part 1** on Canvas.

```
$ cd ~/cs3223_assign2
$ mkdir e0123456
$ cp [a-e]*sql* e0123456/
$ zip -r e0123456.zip e0123456
```

For the second part, answer five questions for **Assignment 2 Part 2** on Canvas.

5 References

- PostgreSQL Documentation https://www.postgresql.org/docs/current/index.html
- EXPLAIN command
  - Guide http://www.postgresql.org/docs/current/using-explain.html
  - Manpage http://www.postgresql.org/docs/current/sql-explain.html
- Query Plan Configuration http://www.postgresql.org/docs/current/runtime-config-query.html
- pg_statio_user_tables
  https://www.postgresql.org/docs/current/monitoring-stats.html#MONITORING-PG-STATIO-ALL-TABLES-VIEW
- pg_stat_user_tables
  https://www.postgresql.org/docs/current/monitoring-stats.html#MONITORING-PG-STAT-ALL-TABLES-VIEW
- pg_statio_user_indexes
  https://www.postgresql.org/docs/current/monitoring-stats.html#MONITORING-PG-STATIO-ALL-INDEXES-VIEW
- pg_stat_user_indexes
  https://www.postgresql.org/docs/current/monitoring-stats.html#MONITORING-PG-STAT-ALL-INDEXES-VIEW
- pg_class catalog https://www.postgresql.org/docs/current/catalog-pg-class.html
- pgstattuple extension https://www.postgresql.org/docs/current/pgstattuple.html
- psql
  - Guide https://www.postgresql.org/docs/current/tutorial-accessdb.html
  - Manpage https://www.postgresql.org/docs/current/app-psql.html
- SQL commands
  - BEGIN https://www.postgresql.org/docs/current/sql-begin.html
  - DROP INDEX https://www.postgresql.org/docs/current/sql-dropindex.html
  - ROLLBACK https://www.postgresql.org/docs/current/sql-rollback.html
  - SET https://www.postgresql.org/docs/current/sql-set.html