National University of Singapore
School of Computing

CS3223: Database Systems Implementation
Semester 2, 2015/16

Assignment 2: Evaluation of Selection Queries (12 marks)

Due: 11:59pm, March 23, 2016 (Wednesday)

1 Introduction

In this assignment, we examine the evaluation of selection queries. As in the previous assignment, this assignment is to be done using your assigned compg compute node.
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 your team will receive 0 credit 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. 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. 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, we will examine the query plans for selection queries on a relation $r(\underline{a}, b, c, d)$ which has three B^+-tree indexes:

- *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.

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
OBIS-b-c	BitmapOR index scan on b_idx & c_idx

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

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=428.63..8065.83 rows=22736 width=32)
        Recheck Cond: (b = 20)
        -> Bitmap Index Scan on b_idx (cost=0.00..422.94 rows=22736 width=0)
             Index Cond: (b = 20)
(4 rows)
```

The result indicates that the cheapest query plan for Q is `BIS-b` with an estimated evaluation cost of $8,065.83$ and an estimated number of $22,736$ 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 runing time along with other statistics.

```
assign2=# explain analyze select * from r where b=20;
                            QUERY PLAN
-------------------------------------------------------------------
Bitmap Heap Scan on r (cost=428.63..8065.83 rows=22736 width=32) (actual time=6.886..24.649 rows=20000 loops=1)
        Recheck Cond: (b = 20)
        Heap Blocks: exact=6900
        -> Bitmap Index Scan on b_idx (cost=0.00..422.94 rows=22736 width=0) (actual time=5.301..5.301
        rows=20000 loops=1)
             Index Cond: (b = 20)
Planning time: 0.168 ms
Execution time: 25.562 ms
(7 rows)
```

The above output shows that the total execution time for the query is 25.562 ms with $20,000$ 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 view named `pg_statio_all_tables`.

```
assign2=# select pg_stat_reset();
assign2=# select dropdbbuffers('assign2');
assign2=# explain analyze select * from r where b=20;
.........
assign2=# select relname, heap_blks_read, heap_blks_hit, idx_blks_read, idx_blks_hit
from pg_statio_all_tables where relname = 'r';
 relname | heap_blks_read | heap_blks_hit | idx_blks_read | idx_blks_hit
---------+----------------+---------------+---------------+--------------
 r       | 6900           | 0             | 62            | 0
(1 row)
```

[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.

The statement "select pg_stat_reset()" is used to reset all the statistics counters before the query is run, and the statement "select dropdbbuffers('assign2')" is used to clear all disk blocks belonging to the database 'assign2' from the buffer pool. In the view `pg_statio_all_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" are defined similarly for accessed index blocks.

Besides examining the statistics maintained by `pg_statio_all_tables`, another approach to obtain a summary of the accessed index/data blocks is to use the `BUFFERS` option for the `EXPLAIN` command:

```
assign2=# explain (analyze, buffers) select * from r where b=20;
                              QUERY PLAN
-----------------------------------------------------------------------
Bitmap Heap Scan on r (cost=428.63..8065.83 rows=22736 width=32) (actual time=6.886..24.649 rows=20000 loops=1)
        Recheck Cond: (b = 20)
        Heap Blocks: exact=6900
        Buffers: shared hit=3 read=6957
        -> Bitmap Index Scan on b_idx (cost=0.00..422.94 rows=22736 width=0) (actual time=5.301..5.301
        rows=20000 loops=1)
              Index Cond: (b = 20)
        Buffers: shared hit=3 read=57
Planning time: 0.168 ms
Execution time: 25.562 ms
(7 rows)
```

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..19853.00 rows=22736 width=32) (actual time=0.018..132.796 rows=20000 loops=1)
        Filter: (b = 20)
        Rows Removed by Filter: 980000
Planning time: 0.082 ms
Execution time: 133.506 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_seqscan = off;
SET
assign2=# explain analyze select * from r where b=20;
                              QUERY PLAN
-----------------------------------------------------------------------
Index Scan using b_idx on r (cost=0.42..29888.94 rows=22736 width=32) (actual time=0.024..9.531 rows=20000
loops=1)
        Index Cond: (b = 20)
Planning time: 0.081 ms
Execution time: 10.207 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_bitmapscan
-------------------
off
(1 row)

enable_seqscan
-----------------
off
(1 row)
```

To display the values of all 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`. Recall that the sequential scan plan has already been disabled by an earlier command.

```
assign2=# set enable_bitmapscan = on;
SET
assign2=# begin;
BEGIN
assign2=# drop index b_idx;
DROP INDEX
assign2=# explain analyze select * from r where b=20;
                              QUERY PLAN
----------------------------------------------------------------------------------------------
Bitmap Heap Scan on r (cost=22918.11..30555.31 rows=22736 width=32) (actual time=35.797..46.149 rows=20000
loops=1)
        Recheck Cond: (b = 20)
        Heap Blocks: exact=6900
        -> Bitmap Index Scan on cb_idx (cost=0.00..22912.42 rows=22736 width=0) (actual time=34.448..34.448
rows=20000 loops=1)
            Index Cond: (b = 20)
Planning time: 0.126 ms
Execution time: 46.897 ms
(7 rows)
assign2=# rollback;
ROLLBACK
```

Note in 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`.

The cost model used by `PostgreSQL`'s query optimizer involves several cost constants (refer to `http://www.postgresql.org/docs/9.4/static/runtime-config-query.html`) that could be instrumented for the database server hardware. However, tuning of the query optimizer is not the focus of this assignment.

2 Getting Started

For this assignment, ensure that you are using the original version of `PostgreSQL` 9.4.5 and not a modified version from the previous assignment. Perform the following to re-install the original version of `PostgreSQL` and setup for assignment 2:

```
$ cd $HOME
$ wget http://www.comp.nus.edu.sg/~cs3223/assign/assign2.zip
$ unzip assign2
$ cd assign2
$ bash restore-pgsql-linux.sh
$ .  ~/.bash_profile
$ pg_ctl start
$ createdb assign2
$ bash setup.sh
```

The `assign2` sub-directory created in your home directory contains the following files:

- `restore-pgsql-linux.sh`: This script re-installs `PostgreSQL`.
- `setup.sh`: The `setup.sh` script creates and loads data into a relation r, and also creates three indexes on r. The "ALTER TABLE" command is used to configure the amount of space allocated for collecting statistics for each attribute, and the "VACUUM ANALYZE" command is used to perform the statistics collection. The script also creates and installs an SQL extension named `dropdbbuffers`. With this extension, you can use the SQL command "select dropdbbuffers('assign2')" to clear all disk blocks belonging to the `assign2` database from the buffer pool.
- `data.txt`: Data for relation r that is used by `setup.sh`.
- `dropdbbuffers/`: This is a directory containing the `dropdbbuffers` extension to be installed by `setup.sh`.
- `sample-scripts/`: This is a directory containing a sample script and query files for Question 1.
- `solution-a0123456x.txt`: This is a solution template file for you to complete and submit. You should rename this file by replacing "a0123456x" with the student number of a team member.

2.1 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/9.4/static/using-explain.html`
2. Manual page for `EXPLAIN` command.
   `http://www.postgresql.org/docs/9.4/static/sql-explain.html`
3. Details of run-time configuration parameters to influence `PostgreSQL`'s selected query plans.
   `http://www.postgresql.org/docs/9.4/static/runtime-config-query.html`
4. Description of statistics collector for monitoring database activity.
   `http://www.postgresql.org/docs/9.4/static/monitoring-stats.html`

3 Questions

This assignment consists of six questions with five of the SQL queries referred to shown below.

Query	Description
A	SELECT * FROM r WHERE b = 9
B	SELECT * FROM r WHERE c = 10
C	SELECT b FROM r WHERE c > 15
D	SELECT * FROM r WHERE b = 9 AND c = 10
E	SELECT * FROM r WHERE b > 9 AND c = 10

Table 2: Selection queries

Question 1: Compare the actual execution times for the plans `IS-b` and `BIS-b` for Query A. Explain the performance of these plans.

Question 2: Compare the actual execution times for the plans `IS-cb` and `BIS-cb` for Query B. Explain the performance of these plans.

Question 3: Compare the actual execution times for the plans `IS-c`, `BIS-c`, and `IOS-cb` for Query C. Explain the performance of these plans.

Question 4: Compare the actual execution times for the plans `BIS-cb` and `ABIS-b-c` for Query D. Explain the performance of these plans.

Question 5: Write a selection query on relation r of the form

$$\text{SELECT * FROM r WHERE (b } op_1 \ v_1) \text{ AND (c } op_2 \ v_2)$$

where each op_i is a comparison operation and each v_i is a constant such that the plan `IS-b` outperforms the plan `BIS-cb` in terms of actual execution time for the query.

Question 6: Which is the best query plan for Query E in terms of execution time? Which is the worst query plan for Query E in terms of execution time? For this question, the query plan search space refers to all the query plans that could be configured for execution.

The directory `~/assign2/sample-scripts` contains a sample script and query files for Question 1. For each query, it is recommended that you execute it multiple times to derive its average execution time.

4 What & How to Submit

For this assignment, you just need to submit your answers in the provided template file solution-a0123456x.txt. Remember to rename this file by replacing "a0123456x" with the student number of one of the team members before uploading it to IVLE's `Submission-Assignment-2` workbin. Each team should upload only one submission.