Advanced Databases (CO572), Coursework 1

Indexing and Querying

1 Introduction

The objective of this coursework is to practice the complex interplay between data storage and processing. Depending on the available indices, processing can be implemented using different algorithms with various optimizations.

You will work with a database of the following schema

```
CREATE TABLE Items (salesdate INT, employee INT, price INT);
CREATE TABLE Orders (
  salesdate INT,
  employee INT,
  employeemanagerid INT,
  discount INT
);
CREATE TABLE Stores (
  managerid INT,
  latitude INT,
  longitude INT,
  countryid INT
);
```

2 Getting started

To get started log in to a lab machine and run the following sequence of commands:

```
git clone $YOUR_REPOSITORY_URL
cd  $YOUR_REPOSITORY_DIRECTORY
```

You may want to set up two separate build directories for the code, one for debugging and one for benchmarking. Here is how you could do that:

```
mkdir Debug
cd Debug
cmake -DCMAKE_BUILD_TYPE=Debug ..
cd ..
mkdir Release
cd Release
cmake -DCMAKE_BUILD_TYPE=Release ..
cd ..
```

You can compile each by (respectively) typing:

```
cmake --build Debug
```

or

```
cmake --build Release
```

Note that the first time you build each of these will take a long time since it also builds dependencies.

2.1 Testing

To run the tests (see file `tests.cpp`), simply run

```
./Debug/tests
```

a successful run output should look like this (pass -? for more options)

```
===============================================================================
All tests passed (30 assertions in 3 test cases)
```

2.2 Benchmarking

To run the microbenchmarks (see file `microbenchmarks.cpp`), simply run

```
./Release/microbenchmarks
```

a semi-naive implementation (building no indices) would produce output like this

```
Running ./Release/microbenchmarks
Run on (4 X 1200 MHz CPU s)
Load Average: 0.31, 0.81, 0.74
-------------------------------------------------------------------------
Benchmark                               Time             CPU   Iterations
-------------------------------------------------------------------------
CreateIndicesBenchmark/1024          2.64 us         2.66 us       262929
CreateIndicesBenchmark/4096          2.65 us         2.66 us       262897
CreateIndicesBenchmark/32768         2.93 us         2.95 us       238952
CreateIndicesBenchmark/262144        2.93 us         2.94 us       237840
CreateIndicesBenchmark/1048576       2.93 us         2.94 us       237703
Query1Benchmark/1024                 95.9 us         95.9 us         7192
Query1Benchmark/4096                 1010 us         1010 us          704
Query1Benchmark/32768               46032 us        46030 us           15
Query1Benchmark/262144            2038912 us      2038335 us            1
Query1Benchmark/1048576          31529341 us     31528313 us            1
Query2Benchmark/1024                   301 us          301 us         2328
Query2Benchmark/4096                  5690 us         5690 us          117
Query2Benchmark/32768               120979 us       120959 us            6
Query2Benchmark/262144             5047080 us      5046606 us            1
Query2Benchmark/524288            10067662 us     10067067 us            1
Query3Benchmark/1024                  1117 us         1117 us          629
Query3Benchmark/4096                 11162 us        11159 us           67
Query3Benchmark/32768               565663 us       565543 us            1
Query3Benchmark/262144            38888371 us     38886797 us            1
```

a good solution (making use of appropriate indices) produces output like this (same hardware):

```
Running ./Release/microbenchmarks
Run on (4 X 1200 MHz CPU s)
Load Average: 0.24, 0.49, 0.63
-------------------------------------------------------------------------
Benchmark                               Time             CPU   Iterations
-------------------------------------------------------------------------
CreateIndicesBenchmark/1024          1364 us         1364 us          515
CreateIndicesBenchmark/4096          5496 us         5497 us          127
CreateIndicesBenchmark/32768        52534 us        52535 us           13
CreateIndicesBenchmark/262144      504598 us       504595 us            1
CreateIndicesBenchmark/1048576    2237887 us      2237300 us            1
Query1Benchmark/1024                  249 us          249 us         2808
Query1Benchmark/4096                 1460 us         1460 us          480
Query1Benchmark/32768               11800 us        11796 us           60
Query1Benchmark/262144              99317 us        99310 us            7
Query1Benchmark/1048576            397303 us       397276 us            2
```

```
Query2Benchmark/1024          139 us          139 us         5034
Query2Benchmark/4096         2126 us         2126 us          329
Query2Benchmark/32768       17721 us        17720 us           37
Query2Benchmark/262144      22913 us        22913 us           28
Query2Benchmark/524288      28612 us        28611 us           18
Query3Benchmark/1024          643 us          643 us         1090
Query3Benchmark/4096         4174 us         4173 us          168
Query3Benchmark/32768       38987 us        38984 us           18
Query3Benchmark/262144     346434 us       346411 us            2
```

3 Your task

Your task is to implement three queries using the techniques, algorithms and data structures you have learned about in class. You shall also implement one or more indices to accelerate the queries. Your are free to implement index structures of your choosing but you need to justify your choice.

The file `solution.c` contains stubs for four functions: three of them need to be filled with the implementation of the queries and one is a preparation function you can use to build your index.

3.1 Q1

```
SELECT
  COUNT(*)
FROM
  Items,
  Orders
WHERE
  Items.price < $1
  AND Orders.employeeManagerID = $2
  AND Items.salesDate = Orders.salesDate
  AND Items.employee = Orders.employee
```

3.2 Q2:

```
SELECT
  COUNT(*)
FROM
  Items,
  Orders
WHERE
  Orders.discount = $1
  AND Items.salesDate <= Orders.salesDate
  AND Orders.salesDate <= Items.salesDate + $2
```

4 Q3:

```
SELECT
  COUNT(*)
FROM
  Items,
  Orders,
  Stores
WHERE
  Stores.managerID = Orders.employeeManagerID
  AND Items.salesDate = Orders.salesDate
  AND Items.employee = Orders.employee
  and store.countryid = $1;
```

5 Solution and Marking

Coursework shall be handed in teams of two. Form teams and designate one of you as lead. When handing in your solution, place a file named `partner.txt` in the root of the repository. When marking the solutions, we will read the first line of this file and attribute the same marks to the login mentioned there.

The marks are distributed as follows:

- Correct implementation of the queries/passing tests: 40%
- Correct (and leak-free!) implementation of the queries/passing tests: 40%
- Correct (and leak-free!) implementation of at least one indexing structure and use of the index to accelerate query processing: 30%
- Justification of the decision to implement this index structure: 20%
- Something extra: 10%

The "extra" can be anything: a detailed performance analysis, hybrid index structures, adaptive indexing, exploitation of hardware features, etc. If you are unsure if your "extra" is enough to get full marks, raise the matter after class.

6 Submission

The coursework will be submitted using LabTS. Important are three files that shall be in the root directory of the repository:

- `solution.c`, containing **all** code pertaining to your solution. No other files shall be modified!
- `explanation.txt`, explaining the solution: what indices were implemented and why? Also use this to justify your "extra". What is cool about your solution and why do you deserve 10% extra marks
- `partner.txt`, the file containing the name of your teammate

7 Competition

In addition, but completely unrelated, to the coursework, we are having a competition. For that, we will run your queries in a "macrobenchmark", i.e., in a sequence simulating the workload of a real data management system. You can run the macrobenchmarks yourself:

```
./Release/macrobenchmark
```

. You will also find the implementation in `macrobenchmark.cpp`

To make things interesting, we are running your solution in a highly resource-constrained environment: a raspberry pi (Model 3B). Every time you trigger a test on labts, the your solution is tested and uploaded to the leaderboard. The leaderboard can be accessed at `http://dbtitans.lsds.uk`.