Univ.-Prof. Dr.-Ing. Matthias Boehm
Graz University of Technology
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMVIT endowed chair for Data Management

3 Database Management SS 2020: Exercise 03 – Tuning and TXs

Published: April 28, 2020
Deadline: May 19, 2020, 11:59pm

This exercise on tuning and transactions aims to provide practical experience with physical design tuning (such as indexing), query rewriting and processing, as well as transaction processing. The expected result is a zip archive named `DBExercise03_<studentID>.zip`, submitted in TeachCenter. If all tasks (excluding extra credit) are submitted, it should contain `Tuning.sql`, `BTree.pdf`, `Transactions.sql`, and a folder called `IteratorModel`.

3.1 Query Rewriting and Tuning (6/25 points)

In order to obtain a better understanding of query rewriting, optimization and the use of index structures, this task aims to compare resulting plans before and after manual tuning. For each sub-task, provide the requested SQL, and a short explanation in comments how the query plans differ. The plans can be obtained via text or visual `EXPLAIN` and don't need to be included.

(a) **Query Unnesting:** Rewrite **Q09** below into a semantically equivalent SQL query without any sub queries (i.e., please, unnest the sub-query of the `IN`-clause).

```
-- Q09:
SELECT I.Name FROM Institutions I WHERE I.CoKey IN(
  SELECT CoKey FROM Countries C WHERE C.Name='Germany' OR C.Name='Austria')
```

(b) **Query Rewriting:** Rewrite **Q10** below into a semantically equivalent SQL query without any intersection or difference operations.

```
-- Q10:
(SELECT P.Name FROM Persons P, Theses T
  WHERE P.Akey = T.Akey AND T.Year < 2020)
INTERSECT
(SELECT P.Name FROM Persons P, Theses T
  WHERE P.Akey = T.Akey AND T.Year >= 2018)
```

(c) **Indexing:** Add a secondary (non-clustered) index on an attribute of your choosing to speedup the original or rewritten query **Q10**. Provide the SQL statement for index creation, and again compare the resulting plans.

Partial Results: SQL script `Tuning.sql` with DML and DDL SQL statements for queries and index creation, including comments explaining the plan differences.

3.2 B-Tree Insertion and Deletion (6/25 points)

As a preparation step, let `x = 0.2<studentID>` and generate a sequence of 20 numbers via

```
SET seed TO <x>; SELECT * FROM generate_series(1,20) ORDER BY random();
```

Now, assume an empty B-tree with $k = 2$ (max $2k = 4$ keys, $2k + 1 = 5$ pointers), sequentially insert the sequence of numbers in the obtained order, and draw the resulting B-tree. Subsequently, delete all keys in the range [8, 14) (lower inclusive, upper exclusive) in order of keys (i.e., del 8, del 9, ..., del 13), and draw again the resulting B-tree. Along the two required B-trees, we recommend to document selected intermediate steps (e.g., whenever a B-tree node is added or removed) to convey your thought process and thus, minimize risk of partial mistakes. Furthermore, please provide your generated sequence of numbers in the obtained order.

Partial Results: A PDF document `BTree.pdf` with the required B-trees and optional documentation (intermediate B-trees or explanations).

3.3 Iterator Model and Operator Implementation (9/25 points)

For a deeper understanding of the iterator model, individual operators, and query processing in general, implement an in-memory **table scan**, a **selection** operator, a **hash join**, as well as a **hash group-by** in a programming language of your choosing (e.g. Python, Java, C# or C++). All operators should implement the `open()`, `next()`, `close()` iterator model (e.g., via an iterator base class and derived operators classes). For testing, you could construct yourself an example query $\gamma_{a,\text{sum(b)}}(\sigma_{c=7}(R) \bowtie_{d=e} S)$ as follows, where `cR` and `cS` are in-memory collections (e.g., lists) of tuples or arrays, and a-e are tuple positions:

```
Q = new HashGroupBy(gcol=a, afun=SUM, acol=b,
        new HashJoin(jcols=(d,e),
            new Selection(pred='c=7', new TblScan(cR)),
            new TblScan(cS)))
Q.open()
while((t=Q.next()) != null)
    print(t)
Q.close()
```

Your implementation should support single-attribute equality selection predicates, single-attribute many-to-many equality inner joins, and single-attribute grouping and aggregation (with aggregation functions sum and count, which are both incrementally maintainable).

Partial Results: A folder `IteratorModel` including the source code of all required operators, and any custom `Tuple` implementations used (no build/run scripts necessary).

3.4 Transaction Processing (4/25 points)

(a) For a basic understanding of transaction processing, create two tables `R(a INT, b INT)`, `S(a INT, b INT)` and insert the following tuples in one atomic transaction:

```
R := {(2, 4) (3, 5) (6, 8) (7, 9)}
S := {(4, 20) (5, 21) (6, 80)}
```

(b) Then create two SQL transactions that can be executed interactively (e.g., in psql terminals) in order to create the anomaly **Phantom Read** under the lowest possible isolation level that prevents it and explain how it is prevented. Afterward, execute the same transactions under the highest isolation level that does not prevent the anomaly and explain why. You should hand in your transactions for creating the anomaly including a description how they should be interleaved. Note that the PostgreSQL isolation levels are stricter than the standard requires (please, have a look into the PostgreSQL documentation for details).

Partial Results: SQL script `Transactions.sql` including the explanations as comments.

3.5 Extra Credit (5 points)

With the implementation of a hash group-by in mind, explain how a specialized group-by operator implementation could exploit the structure of query **Q11** below for improving its latency (time until first result tuple) and total query execution time.

```
-- Q11:
SELECT Year FROM Theses
  GROUP BY Year
  HAVING count(*) > 8
  LIMIT 3
```

Feel free to provide—as an alternative to the required explanation—such an implementation of a specialized group-by operator.

Partial Results: Text file `TuningX.txt` containing your explanation, and/or referencing the source file of the specialized operator in folder `IteratorModel`.