Assignment 3

Index Tuning

Database Tuning

Group Name (e.g. A1, B5, B3)

Balint Peter, 12213073

Günter Lukas, 12125639

Ottino David, 51841010

April 15, 2025

Database system and version: *PostgreSQL 17.4-1*

1 Index Data Structures

Which index data structures (e.g., B^+ tree index) are supported?

1.1 B-Tree

Postgres uses B-Trees as it's standard data structure for storing indexes. Whenever the `CREATE INDEX` command is used without additional parameters, a B-Tree is created. The main difference between a B-tree and a B^+ tree is that with a B-tree the data is stored in either internal or leaf nodes while a B^+ tree stores the data in leaf nodes only. Intuitively, this may seem more efficient , but this also leads to more storage being used for one internal node which decreases the amount of pointers you can pack into into it. This leads to a smaller fanout compared to a B^+ tree. Additionally, the B^+ tree also uses a linked list for data stored in leaf nodes which is not possible for a B tree. Generally speaking B trees are more efficient when it comes to single access of random data while B^+ trees are prefered when range-queries have to be executed.

1.2 Hash

Hash indexes in Postgres store a 32-bit hash code of the indexed column which is then used to find and access data. Due to its nature hash indexes are considered when the indexed column is involved in a comparison using the "-"-Operator.

1.3 GiST

GiST stands for Generalized Search Tree. It doesn't describe a specific index type but is more of a framework which allows Postgres-Users to implement their own arbitrary indexing schemes, for example B^+ trees. However, it comes with some built-in standard functionality.

1.4 SP-GiST

The "SP" in this index type stands for space-partitioned GiST, which means that the search space is repeatedly divided into different partitions of potentially different sizes. Searches that are well matched to the partitioning rule can be very efficient. Similar to GiST, SP-GiST allows users to develop their own custom data types and access methods.

1.5 GIN

GIN stands for "Generalized Inverted Index" and is used for complex data values that consist of different components, such as arrays or texts, and the expected queries search for different elements within this data. GIN stores each of these components, so called keys, within a separate index. Each key references to the tupels it is contained in. It is also possible to develop custom data types and access methods with this index type.

1.6 BRIN

BRIN, which is shorthand for "Block Range Indexes", is mainly used for very large tables in which attributes can naturally be ordered in some way and this order is resembled by their location in physical memory. BRIN stores a summary of all the data present in one or multiple pages that are besides each other in memory. This means that the index itself is very small and therefore fast to traverse. However, BRIN is a lossy type of index, which means that after traversing the index, the returned tupels will still have to be checked for the actual selection criteria.

2 Clustering Indexes

Discuss how the system supports clustering indexes, in particular:

a) How do you create a clustering index on `ssnum`? Show the query.[1]

PostgreSQL does not directly provide clustered indexes. However, the command `CLUSTER` clusters a given table based on an already existing index, but this doesn't work on a hash index.

```
CREATE INDEX idx_employee_ssnum ON Employee(ssnum);
CLUSTER Employee USING idx_employee_ssnum;
```

b) Are clustering indexes on non-key attributes supported, e.g., on `name`? Show the query.

Yes, it is possible to create clustering indexes on all attributes using the aforementioned method.

```
CREATE INDEX IDX_EMPLOYEE_NAME ON EMPLOYEE (NAME);
CLUSTER EMPLOYEE USING IDX_EMPLOYEE_NAME;
```

[1]Give the queries for creating a hash index *and* a B^+ tree index if both of them are supported.

c) Is the clustering index dense or sparse?

The PostgreSQL-documentation does not outright state if the created indexes are dense or sparse. However, it is possible to view the number of entries of a given index which allows us to conclude if the index is dense or sparse.

```
SELECT COUNT(DISTINCT MANAGER)FROM EMPLOYEE;
CREATE INDEX IDX_EMPLOYEE_MANAGER ON EMPLOYEE (MANAGER);
SELECT reltuples FROM PG_CLASS
WHERE RELNAME = 'idx_employee_manager';
```

We know from the first query that there are 49.825 managers. But the total number of tuples in the index is 100.000, which is the total amount of rows. Therefore we can conclude that the index is dense.

d) How does the system deal with overflows in clustering indexes? How is the fill factor controlled?

Since the clustering index in PostgreSQL is not really a clustered index in the traditional sense, overflows are not really an issue since new data is just inserted whereever there is an open space and a repeated clustering based on the index leads to the whole data structure being reorganized. The fill factor can be set by the user for each table or index individually with the following statements:

```
ALTER TABLE EMPLOYEE SET (fillfactor = 80);
CREATE INDEX IDX_EMPLOYEE_MANAGER ON EMPLOYEE (MANAGER) WITH (fillfactor = 80);
```

The specified fill factor is stored in `pg_class` in the `reloptions`-attribute.

e) If new data is inserted into the table, the additions won't be clustered automatically and a new `CLUSTER`-operation has to be executed. Clustering took about 1,5 seconds in our case.

```
CLUSTER Employee;
```

The `CLUSTER`-operation without any arguemnts will cluster according to the last index used when clustering.

[Your answer goes here . . .]

3 Non-Clustering Indexes

Discuss how the system supports non-clustering indexes, in particular:

a) How do you create a combined, non-clustering index on `(dept,salary)`? Show the query.[1]

[Your answer goes here . . .]

```
[Your SQL query goes here ...]
```

b) Can the system take advantage of covering indexes? What if the index covers the query, but the condition is not a prefix of the attribute sequence `(dept,salary)`?

[Your answer goes here . . .]

c) Discuss any further characteristics of the system related to non-clustering indexes that are relevant to a database tuner.

[Your answer goes here . . .]

4 Key Compression and Page Size

If your system supports B^+ trees, what kind of key compression (if any) is supported? How large is the default disk page? Can it be changed?

[Your answer goes here . . .]

Time Spent on this Assignment

Time in hours per person: **XXX**

References

Important: Reference your information sources!

https://www.postgresql.org/docs/current/indexes-types.html

https://www.shiksha.com/online-courses/articles/difference-between-b-tree-and-b-plus-tree-blogId-15590

https://www.postgresql.org/docs/current/gin.html

https://www.postgresql.org/docs/8.1/gist.html

https://www.postgresql.org/docs/16/brin-intro.html

https://www.postgresql.org/docs/current/sql-cluster.html

https://www.cybertec-postgresql.com/en/what-is-fillfactor-and-how-does-it-affect-postgresql-performanc