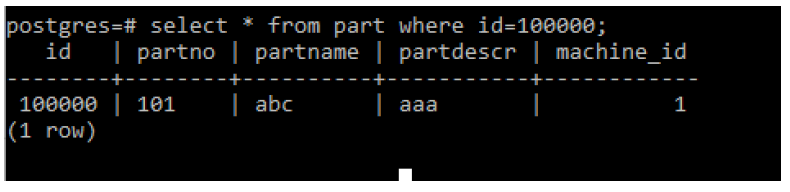
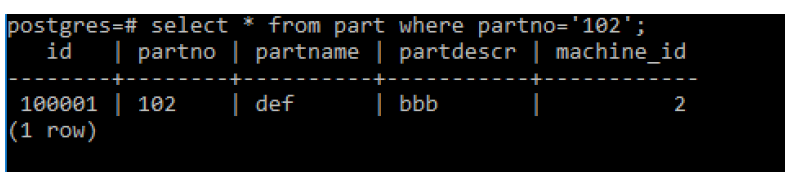
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Hope Foundation’s,**  **Finolex Academy of Management and Technology, Ratnagiri** | | | | | | | | | |
| **Department of Information Technology** | | | | | | | | | |
| Subject name: OLAP LAB | | | | | | | | Subject Code: ITL503 | | | |
| Class | | TE IT | | Semester – V (CBCGS) | | | | Academic year: 2018-19 | | | |
| Name of Student | | **Kazi Jawwad A Rahim** | | | | | **QUIZ Score : 06** | | | | |
| Roll No | | **32** | | | Assignment/Experiment No. | | | | | 03 | |
| Title**: Implementation of query optimization Postgre SQL.** | | | | | | | | | | | |
|  | | | | | | | | | | | |
| 1. **Course objectives applicable:**   **LOB1**- Understand the fundamentals of query- based optimization techniques. | | | | | | | | | | | |
| 1. **Course outcomes applicable:**   **LO1**- Apply the knowledge of query processing and query optimization of database management system | | | | | | | | | | | |
| 1. **Learning Objectives:**  * To be able to apply the knowledge of query optimization techniques. * To understand concept of query processing. | | | | | | | | | | | |
| 1. **Practical applications of the assignment/experiment:**  * Wherever the database management system and data retrieval is required E.g. Company database, banking system, Airline reservation system. | | | | | | | | | | | |
| **5. Prerequisites**: SQL commands, Indexing | | | | | | | | | | | |
| **6. Hardware Requirements**:   1. PC with 4GB RAM, 500GB HDD,   **7. Software Requirements:**  1. Postgre sql | | | | | | | | | | | |
|  | | | | | | | | | | | |
| **8. Quiz Questions (if any): (Online Exam will be taken separately batch wise, attach the certificate/ Marks obtained)**   1. https://goo.gl/BqcH8Q | | | | | | | | | | | |
|  | | | | | | | | | | | |
| **9. Experiment/Assignment Evaluation:** | | | | | | | | | | | |
| **Sr. No.** | **Parameters** | | | | | | | | **Marks obtained** | | **Out of** |
| **1** | Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used - | | | | | | | |  | | 6 |
| **2** | Neatness/presentation | | | | | | | |  | | 2 |
| **3** | Punctuality | | | | | | | |  | | 2 |
| **Date of performance (DOP)** | | |  | | | **Total marks obtained** | | |  | | **10** |
| **Date of checking (DOC)** | | |  | | | **Signature of teacher** | | | | | |

**Results:**

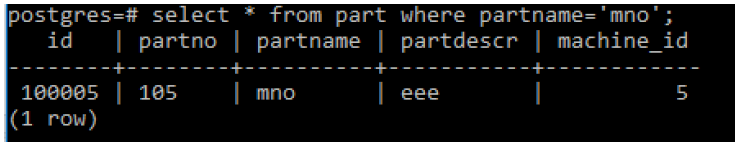
**B-tree Indexes:**

When we define this rather common table, PostgreSQL creates two unique B-tree indexes behind the scenes: part\_pkey and part\_partno\_key. So every unique constraint in PostgreSQL is implemented with a unique INDEX. Now let’s try to do some queries on our table.



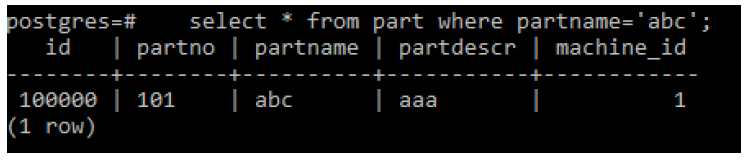


We observe that it takes only fractions of the millisecond to get our results. We expected this since for both columns used in the above queries, we have already defined the appropriate indexes. Now let’s try to query on column partname, for which no index exists.

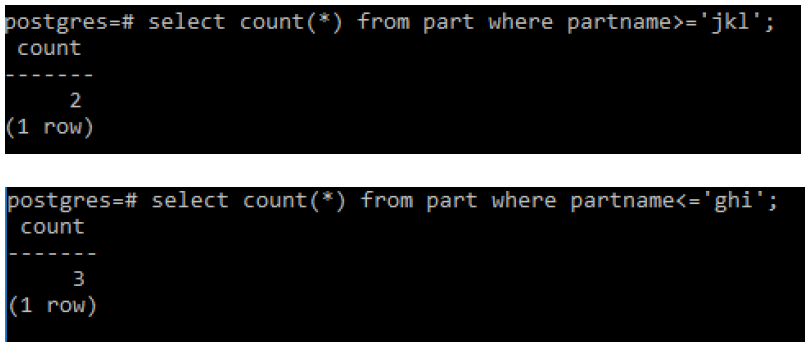


Here we see clearly that for the non indexed column, the performance drops significantly.

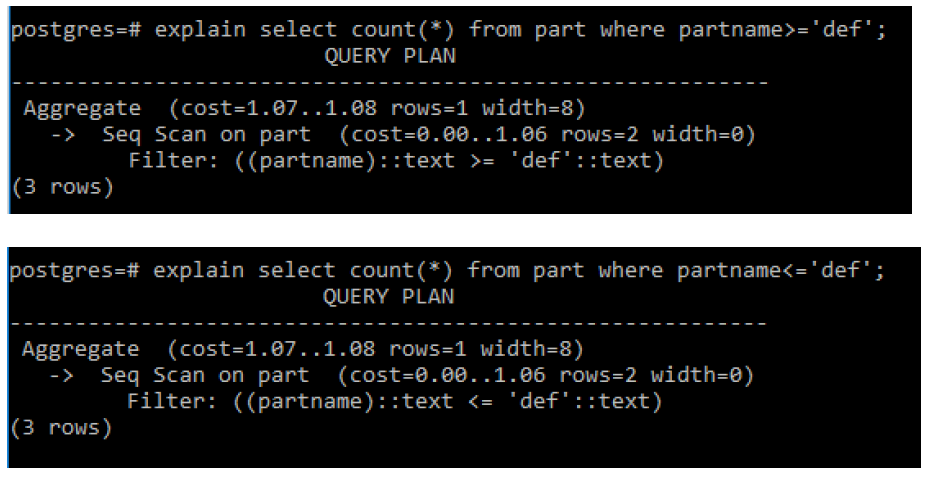
Now let’s create an index on that column, and repeat the query:



Our new index part\_partname\_idx is also a B-tree index (the default). First we note that the index creation on the million rows table took a significant amount of time, about 16 seconds. Then we observe that our query speed was boosted from 89 ms down to 0.525 ms. B-tree indexes, besides checking for equality, can also help with queries involving other operators on ordered types, such as <,<=,>=,>. Lets try with <= and >=



The first query is much faster than the second, by using the EXPLAIN (or EXPLAIN ANALYZE) keywords we can see if the actual index is used or not:

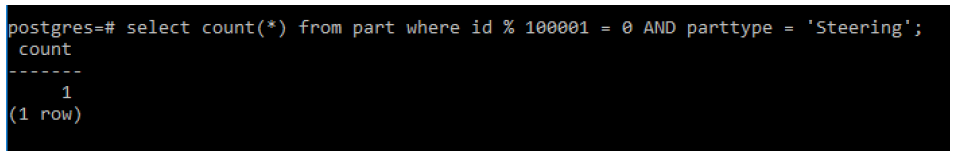


In the first case, the query planner chooses to use the part\_partname\_idx index. We also observe that this will result in an index-only scan which means no access to the data tables at all. In the second case the planner determines that there is no point in using the index as the returned results are a big portion of the table, in which case a sequential scan is thought to be faster.

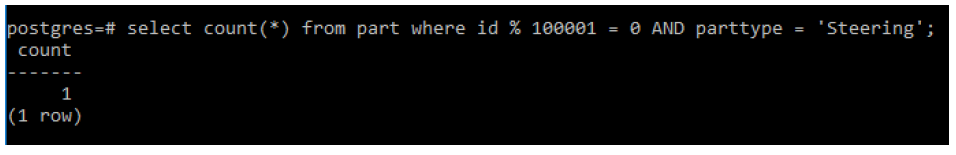
**Hash Indexes:**

Use of hash indexes up to and including PgSQL 9.6 was discouraged due to reasons having to do with lack of WAL writing. As of PgSQL 10.0 those issues were fixed, but still hash indexes made little sense to use. There are efforts in PgSQL 11 to make hash indexes a first class index method along with its bigger brothers (B-tree, GiST, GIN). So, with this in mind, let’s actually try a hash index in action.

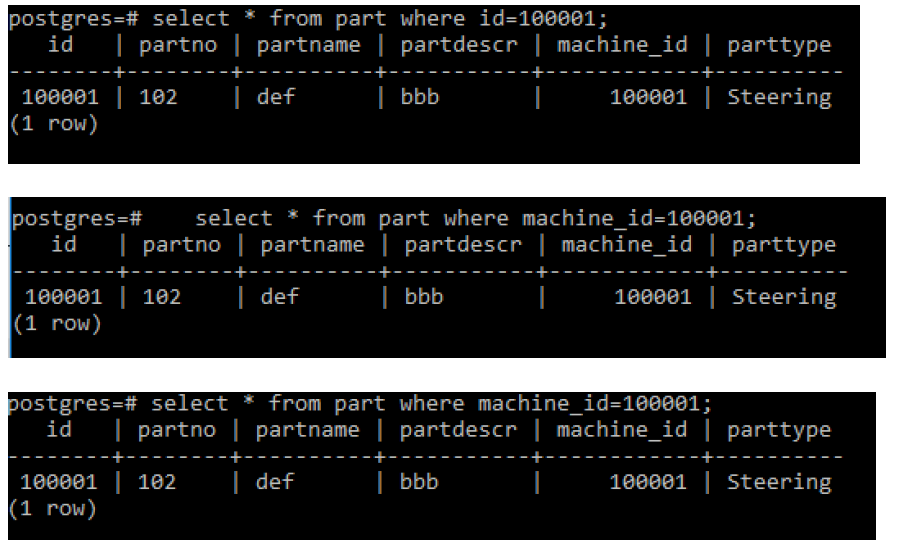
We will enrich our part table with a new column parttype and populate it with values of equal distribution, and then run a query that tests for parttype equal to ‘Steering’:



Now we create a Hash index for this new column, and retry the previous query:



We note the improvement after using the hash index. Now we will compare the performance of a hash index on integers against the equivalent b-tree index.



As we see, with the use of hash indexes, the speed of queries that check for equality is very close to the speed of B-tree indexes. Hash indexes are said to be marginally faster for equality than B-trees, in fact we had to try each query two or three times until hash index gave a better result than the b-tree equivalent.