# AUTOMATIC CREATION OF INDICES

*Report*

**Ankita Singh (19305R002)**

**Aarushi Aiyyar (203050045)**

14.12.2020

CS-631

## GOALS

In this project, our aim was to modify the access layer of PostgreSQL to keep track of relation scans that could have benefitted from an index, and if there are many such scans for a particular index, create the index automatically.

## SUMMARY OF THE IMPLEMENTATION PLANS

First, we check if there is a need for index creation. Then, we check whether the ratio of the number of tuples satisfying the where clause and the total number of tuples in the table is less than a certain threshold.We do that only if the execution process undergoes sequential scan. If so, the index is created.

## IMPLEMENTATION AND MODIFIED FILES

The files that have been modified are :

***a. nodeSeqscan.c:*** (Function in which changes are made: *ExecEndSeqScan()*)

1. ***pg_indices_stats*** is a relation that stores the name of the relation and the corresponding attribute for which the index needs to be created. This table is created in *nodeSeqscan.c*
2. Firstly, an index is created only if the size of the relation exceeds 500 tuples. *(TABLE_SIZE_THRESHOLD)*
3. Then, the total number of tuples of the relation (*tot_scans)* and the number of tuples satisfying the where clause in the given query *(tuples_returned)* are found out.
4. The threshold is set to SINGLE_ATTR_THRESHOLD (0.4 for demo purposes) and the ratio of t*uples_returned/tot_scans* is compared to see if it is less than the threshold. If that is the case, the required attribute and relation name are added in *pg_indices_stats* table. (Actual creation will be done after the execution cycle is complete in *postgres.c*).

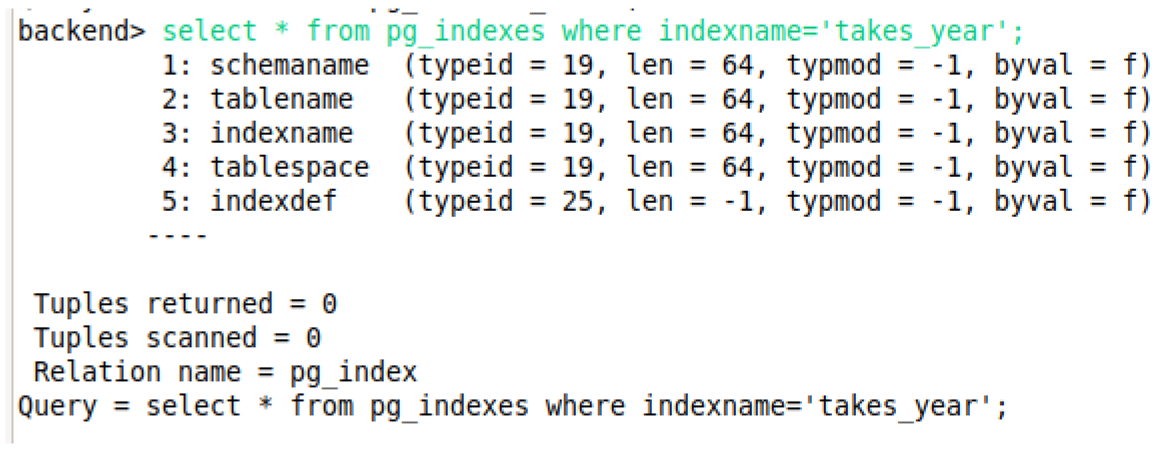
***b. nodeSeqscan.c:***(Function in which changes are made: exec_simple_query())

1. We consider only one index query would be stored in *pg_indices_stats* table. Otherwise, we throw an error and skip processing. Then, we retrieve the relation name and the attribute name and delete that entry from *pg_indices_stats.*

2. We then create an index of the form: ***relation-name_attribute-name*** at the end of the execution cycle (Because when we tried to do it before that, we were not able to access the table as there was a lock on it)

**Example 1:** Consider the **takes** table that consists of 30,000 tuples and for the query: select * from takes where year = 2010 returns 3194 tuples. As the size of the table exceeds 500 tuples, an index should be created.

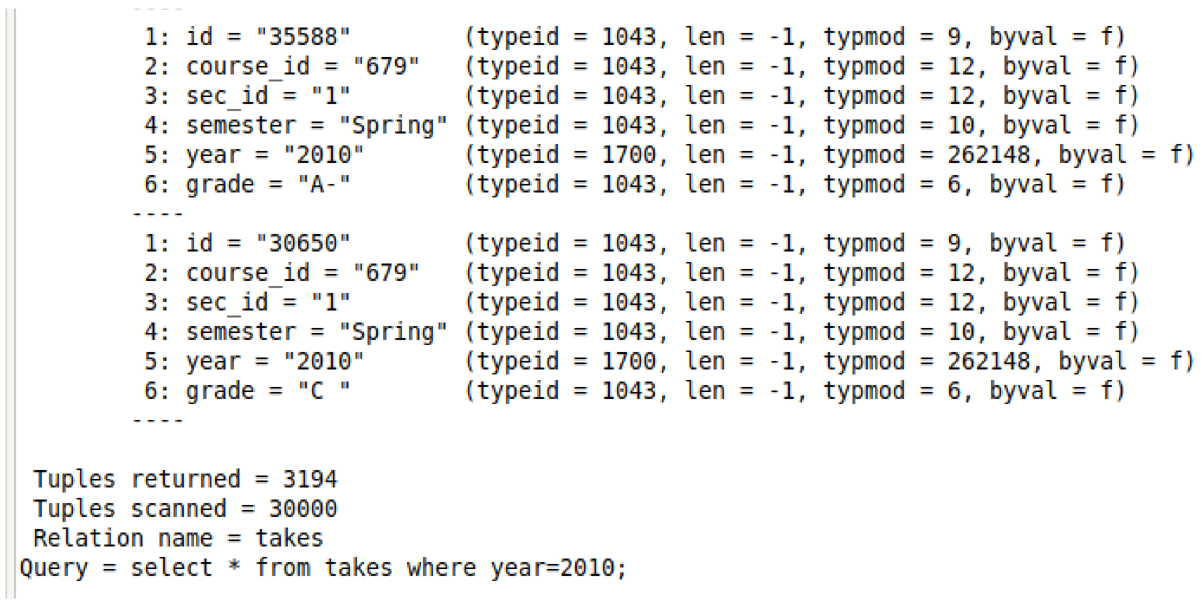
1. **pg_indexes** table stores information about all indices. Here, note that the index on **takes_year** does not exist initially. (**select * from pg_indexes where indexname='takes_year'**)



```
backend> select * from pg_indexes where indexname='takes_year';
         1: schemaname  (typeid = 19, len = 64, typmod = -1, byval = f)
         2: tablename   (typeid = 19, len = 64, typmod = -1, byval = f)
         3: indexname   (typeid = 19, len = 64, typmod = -1, byval = f)
         4: tablespace  (typeid = 19, len = 64, typmod = -1, byval = f)
         5: indexdef    (typeid = 25, len = -1, typmod = -1, byval = f)
        ----

 Tuples returned = 0
 Tuples scanned = 0
 Relation name = pg_index
Query = select * from pg_indexes where indexname='takes_year';
```

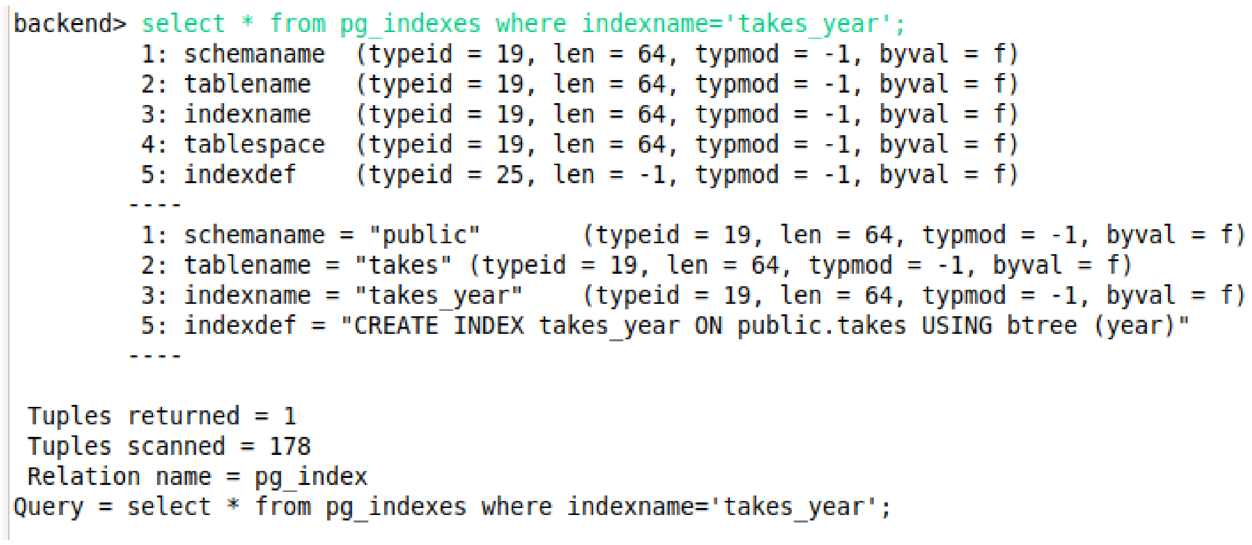
2. Execution of the query **select * from takes where year=2010**. This yields a result of 3194 tuples. But we are showing 2 of them in the screenshot:



```
        ----
         1: id = "35588"        (typeid = 1043, len = -1, typmod = 9, byval = f)
         2: course_id = "679"   (typeid = 1043, len = -1, typmod = 12, byval = f)
         3: sec_id = "1"        (typeid = 1043, len = -1, typmod = 12, byval = f)
         4: semester = "Spring" (typeid = 1043, len = -1, typmod = 10, byval = f)
         5: year = "2010"       (typeid = 1700, len = -1, typmod = 262148, byval = f)
         6: grade = "A-"        (typeid = 1043, len = -1, typmod = 6, byval = f)
        ----
         1: id = "30650"        (typeid = 1043, len = -1, typmod = 9, byval = f)
         2: course_id = "679"   (typeid = 1043, len = -1, typmod = 12, byval = f)
         3: sec_id = "1"        (typeid = 1043, len = -1, typmod = 12, byval = f)
         4: semester = "Spring" (typeid = 1043, len = -1, typmod = 10, byval = f)
         5: year = "2010"       (typeid = 1700, len = -1, typmod = 262148, byval = f)
         6: grade = "C "        (typeid = 1043, len = -1, typmod = 6, byval = f)
        ----

 Tuples returned = 3194
 Tuples scanned = 30000
 Relation name = takes
Query = select * from takes where year=2010;
```

3. **pg_indexes** table now contains an index on **takes_year** after the execution of the above query:

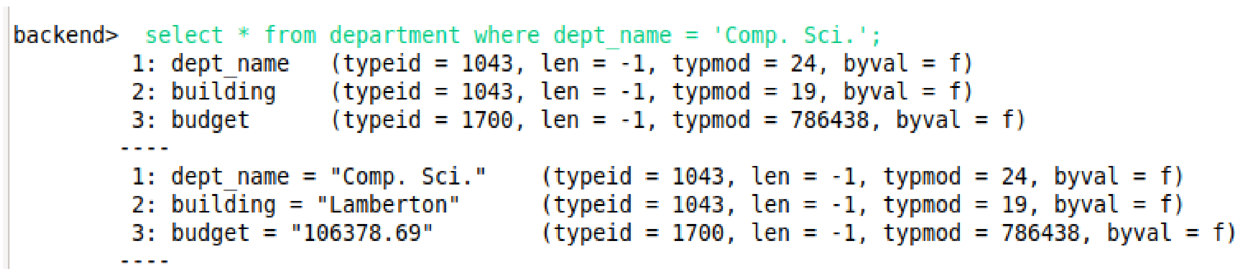


```
backend> select * from pg_indexes where indexname='takes_year';
         1: schemaname  (typeid = 19, len = 64, typmod = -1, byval = f)
         2: tablename   (typeid = 19, len = 64, typmod = -1, byval = f)
         3: indexname   (typeid = 19, len = 64, typmod = -1, byval = f)
         4: tablespace  (typeid = 19, len = 64, typmod = -1, byval = f)
         5: indexdef    (typeid = 25, len = -1, typmod = -1, byval = f)
        ----
         1: schemaname = "public"      (typeid = 19, len = 64, typmod = -1, byval = f)
         2: tablename = "takes" (typeid = 19, len = 64, typmod = -1, byval = f)
         3: indexname = "takes_year"   (typeid = 19, len = 64, typmod = -1, byval = f)
         5: indexdef = "CREATE INDEX takes_year ON public.takes USING btree (year)"
        ----

 Tuples returned = 1
 Tuples scanned = 178
 Relation name = pg_index
Query = select * from pg_indexes where indexname='takes_year';
```

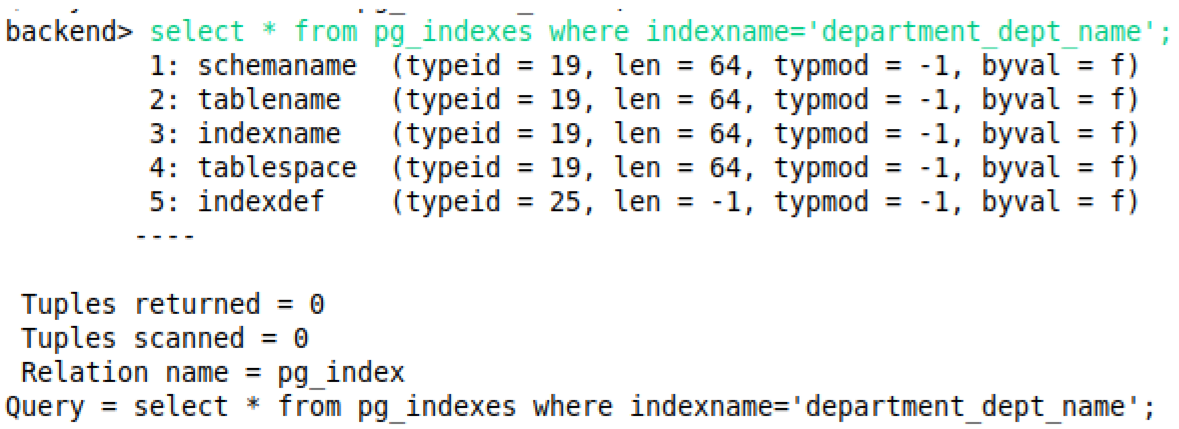
**Example 2:** Consider the **department** table that consists of 20 tuples and for the query: **select * from department where  dept_name = 'Comp. Sci'.** returns 1 tuple. As the size of the table eis less than 500 tuples, an index should not be created.

1. **select * from department where  dept_name = 'Comp. Sci.'** This returns 1 tuple.



```
backend>  select * from department where dept_name = 'Comp. Sci.';
         1: dept_name    (typeid = 1043, len = -1, typmod = 24, byval = f)
         2: building     (typeid = 1043, len = -1, typmod = 19, byval = f)
         3: budget       (typeid = 1700, len = -1, typmod = 786438, byval = f)
        ----
         1: dept_name = "Comp. Sci."     (typeid = 1043, len = -1, typmod = 24, byval = f)
         2: building = "Lamberton"       (typeid = 1043, len = -1, typmod = 19, byval = f)
         3: budget = "106378.69"         (typeid = 1700, len = -1, typmod = 786438, byval = f)
        ----
```

2. This is the **pg_indexes** table. As you can see, no index on department_dept_name was created.



```
backend> select * from pg_indexes where indexname='department_dept_name';
         1: schemaname  (typeid = 19, len = 64, typmod = -1, byval = f)
         2: tablename   (typeid = 19, len = 64, typmod = -1, byval = f)
         3: indexname   (typeid = 19, len = 64, typmod = -1, byval = f)
         4: tablespace  (typeid = 19, len = 64, typmod = -1, byval = f)
         5: indexdef    (typeid = 25, len = -1, typmod = -1, byval = f)
        ----

 Tuples returned = 0
 Tuples scanned = 0
 Relation name = pg_index
Query = select * from pg_indexes where indexname='department_dept_name';
```

## WHAT WE WOULD HAVE LIKED TO IMPLEMENT

1. We also tried modifying costSize.c to put the index creation code before the execution phase and create index-scan if cost criteria matches our threshold. The index creation was not allowed because of conflicting locks. This can be explored more.
2. We obtained the list of attributes in the 'where' clause but considered index creation only when it is a one attribute query. We can put a different threshold for composite query and build the index on the same as well.
3. TEMP tables and 'CONCURRENTLY' clause might be explored for the purpose of creating index table. (Query needs to be modified accordingly).
4. We can modify our ***pg_indices_stats*** table to take into account the number of times the query executed and then create an index using total cost assumption rather than only tuple threshold assumption. Also we can explore on index creation cost for that table and add it to ***pg_indices_stats*** **.** If total cost exceeds the index creation cost then we can add the index.

## NAMES OF THE MODIFIED FILES

| FILENAME | MODIFIED FUNCTIONS | FUNCTIONALITIES |
|---|---|---|
| postgres.c | exec_simple_query() | Checks if there is a need for creating an index and creates one if it doesn't already exist. |
| nodeSeqscan.c | ExecEndSeqScan() | 1.Creates a temporary table to store values of attribute & relation on which index needs to be created.<br><br>2.Add the required attribute and relation name in ***pg_indices_stats*** table. Actual creation will be done after the execution cycle is complete in ***postgres.c*** |

## WHAT WE LEARNT VIA THE PROJECT

1. How to navigate through PostgreSQL internals : how the query string is converted to a parse tree, plan is estimated, execution is done and how postgres tracks all the variables used.
2. How to use the  debugger appropriately: Putting breakpoints whenever we see something interesting, stepping up,out and into functions as and when required.
3. How structures are organized in C code in PostgreSQL: The code structure is so well formed that even at the end of the execution we can navigate through all the attributes/plan/costs etc. We learnt a lot about code organisation from this.
4. We learnt how to extract the total number of tuples in a relation and the number of tuples that satisfy a particular predicate. We learnt about SPI commands and how we can use them to fetch count and values of tuples even when query is executing,
5. We explored a lot of code material ( nearly went into 60+ files) and got a brief idea of the entire framework. A lot of our SPI code and some other code material was extracted from pre-written code.

## CONCLUSION

Thus, we have created indices automatically wherever their creation could help speed up query execution. A lot of modifications could be done on the existing code for better performances and handling further use cases.

## REFERENCES

1. https://doxygen.postgresql.org/
2. https://www.cse.iitb.ac.in/infolab/Data/Courses/CS631/PostgreSQL-Resources/