Create Index using ORDER BY (ASC/DESC)

If we know in which order we are going to SELECT data, we should apply for proper ORDER BY ASC/DESC in the index.

Please check the below full demonstration, and then validate your self:

Create a table with sample records:

```
CREATE TABLE tbl_testIndex(a INTEGER, b INTEGER);

INSERT INTO tbl_testIndex
SELECT x, x FROM generate_series(1, 5000) AS f(x);
```

Create a sample index without any ORDER BY:

```
CREATE INDEX tbl_testIndex_idx ON tbl_testIndex (a, b);
```

Check the execution plan:
Index only scan is working…

```
EXPLAIN
SELECT *FROM tbl_testIndex
ORDER BY a,b;
```

Check the below execution plan:
Sequential Scan of the index which is not good.
Because the index is not available for ORDER BY b DESC.

```
EXPLAIN
SELECT *FROM tbl_testIndex
ORDER BY a,b DESC;
```

Now, drop the old index:

```
DROP INDEX tbl_testIndex_idx;
```

Create a new index with b DESC:

```
CREATE INDEX tbl_testIndex_idx ON tbl_testIndex (a, b DESC);
```

Check the below execution plan:

Index scan is working…

```
EXPLAIN
SELECT *FROM tbl_testIndex
ORDER BY a,b DESC;
```

Introduced BRIN – Block Range Index with Performance Report

PostgreSQL 9.5 introduced the powerful BRIN Index, which is performance much faster than the regular BTREE Index.
The most important two lines of the BRIN are: It stores only minimum and maximum value per block so it does not require more space. For extremely large table It runs faster than any other Indexes.

Below are steps:

First create one sample table:

```
CREATE TABLE tbl_ItemTransactions
 (
     TranID SERIAL
     ,TransactionDate TIMESTAMPTZ
     ,TransactionName TEXT
 );
```

Insert Millions of data to test the performance of BRIN Index:

```
INSERT INTO tbl_ItemTransactions
(TransactionDate, TransactionName)
SELECT x, 'dbrnd'
FROM generate_series('2008-01-01 00:00:00'::timestamptz,
'2016-08-01 00:00:00'::timestamptz,'2 seconds'::interval) a(x);
```

Check the total size of table:

```
SELECT pg_size_pretty(pg_total_relation_size('tbl_ItemTransactions')) AS TableSize;
```

Now Check the performance without any Index:

```
EXPLAIN ANALYSE
SELECT COUNT(1) FROM tbl_ItemTransactions
```

```
WHERE TransactionDate BETWEEN '2012-01-01 00:00:00' and '2014-08-08 08:08:08';
```

Create BRIN index on TransactionDate Column:

```
CREATE INDEX idx_tbl_ItemTransactions_TransactionDate
ON tbl_ItemTransactions
USING BRIN (TransactionDate);
```

Now Check the performance of the same query which has BRIN index:

```
EXPLAIN ANALYSE
SELECT COUNT(1) FROM tbl_ItemTransactions
WHERE TransactionDate BETWEEN '2012-01-01 00:00:00' and '2014-08-08 08:08:08';
```

Create Partial BRIN index on TransactionDate Column:

You can also create Partial BRIN index for your individual range of data. The Partial BRIN index is also faster than normal BRIN index, but we should apply proper filter based on created Partial BRIN index.

```
CREATE INDEX idx_tbl_ItemTransactions_TransactionDate_2012
ON tbl_ItemTransactions
USING BRIN (TransactionDate)
WHERE TransactionDate BETWEEN '2012-01-01' AND '2012-12-31';
```

Script to find Index Size and Index Usage Statistics

In below script, You can also find occupied size of Indexes, Total tuple read and scan by Indexes.

```
SELECT
    pt.tablename AS TableName
    ,t.indexname AS IndexName
    ,pc.reltuples AS TotalRows
    ,pg_size_pretty(pg_relation_size(quote_ident(pt.tablename)::text)) AS TableSize
    ,pg_size_pretty(pg_relation_size(quote_ident(t.indexrelname)::text)) AS IndexSize
    ,t.idx_scan AS TotalNumberOfScan
    ,t.idx_tup_read AS TotalTupleRead
    ,t.idx_tup_fetch AS TotalTupleFetched
FROM pg_tables AS pt
LEFT OUTER JOIN pg_class AS pc
      ON pt.tablename=pc.relname
LEFT OUTER JOIN
(
      SELECT
            pc.relname AS TableName
            ,pc2.relname AS IndexName
            ,psai.idx_scan
            ,psai.idx_tup_read
            ,psai.idx_tup_fetch
            ,psai.indexrelname
      FROM pg_index AS pi
      JOIN pg_class AS pc
            ON pc.oid = pi.indrelid
      JOIN pg_class AS pc2
            ON pc2.oid = pi.indexrelid
      JOIN pg_stat_all_indexes AS psai
            ON pi.indexrelid = psai.indexrelid
)AS T
    ON pt.tablename = T.TableName
WHERE pt.schemaname='public'
ORDER BY 1;
```

Script to find a Missing Indexes of the schema

The full table scanning is always creating a performance issue for any database. On the other hand, Database Administrator may also require a report on missing indexes which they can share with developers and users so that they can modify indexes accordingly.

```
SELECT
       relname AS TableName
       ,seq_scan-idx_scan AS TotalSeqScan
       ,CASE WHEN seq_scan-idx_scan > 0
              THEN 'Missing Index Found'
              ELSE 'Missing Index Not Found'
       END AS MissingIndex
       ,pg_size_pretty(pg_relation_size(relname::regclass)) AS TableSize
       ,idx_scan AS TotalIndexScan
FROM pg_stat_all_tables
WHERE schemaname='public'
       AND pg_relation_size(relname::regclass)>100000
ORDER BY 2 DESC;
```

Script to find the unused and duplicate index

The management and maintenance of database index is a day to day exercise for a Database Administrator, and unused index can create a performance issues for the whole database system. Sometimes, I found that duplicate indexes on the same table, e.g. same table, same columns, same order of columns and created with a different name. Internally this will also impact to our database performance.

I am sharing two different scripts for finding the unused and duplicate index in PostgreSQL.

Script to find the unused indexes in PostgreSQL:

```
SELECT
       PSUI.indexrelid::regclass AS IndexName
       ,PSUI.relid::regclass AS TableName
FROM pg_stat_user_indexes AS PSUI
JOIN pg_index AS PI
    ON PSUI.IndexRelid = PI.IndexRelid
WHERE PSUI.idx_scan = 0
       AND PI.indisunique IS FALSE;
```

Script to find the duplicate indexes in PostgreSQL:

```
SELECT
    indrelid::regclass AS TableName
    ,array_agg(indexrelid::regclass) AS Indexes
FROM pg_index
GROUP BY
    indrelid
    ,indkey
HAVING COUNT(*) > 1;
```