PostgreSQL Performance Tuning

Bruce Momjian

PostgreSQL is an open-source, full-featured relational database. This presentation gives an overview of PostgreSQL performance tuning.

https://momjian.us/presentations
Outline

1. Caching
2. Internals
3. Storage
Caching

https://www.flickr.com/photos/storm-crypt/
## Cache Sizes

<table>
<thead>
<tr>
<th>Storage Area</th>
<th>Measured in</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU registers</td>
<td>bytes</td>
</tr>
<tr>
<td>CPU cache</td>
<td>megabytes</td>
</tr>
<tr>
<td>RAM</td>
<td>gigabytes</td>
</tr>
<tr>
<td>disk drives</td>
<td>terabytes</td>
</tr>
</tbody>
</table>
Checkpoints and WAL Files

Query and Checkpoint Operations

Transaction Durability

PostgreSQL Shared Buffer Cache

Write-Ahead Log

Disk Blocks

Postgres Backend

Postgres Backend

Postgres Backend

Kernel Disk Buffer Cache

fsync

fsync

Recovery
Buffer / Disk Interaction

PostgreSQL Shared Buffer Cache

Begin 1

Write-Ahead Log

End 1

Rotate
Memory Usage

<table>
<thead>
<tr>
<th>RAM</th>
<th>Page In (bad)</th>
<th>Page Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel Disk Buffer Cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared Buffer Cache (shared_buffers)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Postgres Session (work_mem)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Postgres Session (work_mem)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Postgres Session (work_mem)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Postgres Session (work_mem)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Swap
sharedBuffers = 32MB

tempBuffers = 8MB

workMem = 1MB
maintenanceWorkMem = 16MB

effectiveCacheSize = 128MB

Kernel changes often required.
The Anatomy Lesson of Dr. Nicolaes Tulp, Rembrandt van Rijn
SELECT firstname
FROM friend
WHERE age = 33;
test=> SELECT firstname
          test-> FROM friend
          test-> WHERE age = 33;

          firstname
          ---------------
            Sandy
            (1 row)
SELECT firstname
FROM friend
WHERE age = 33;

[ query is processed ]

firstname

Sandy
(1 row)
test=> SELECT firstname
    test->  FROM friend
    test->  WHERE age = 33;

Breakpoint 1, PQexec (conn=0x807a000,
    query=0x8081200  "SELECT firstname\n           FROM friend\n           WHERE age = 33;")
    at fe-exec.c:1195
Libpq

User Terminal

Application Code

Libpq

PostgreSQL Database Server

Queries

Results
ack 61 win 8760 <nop,nop,timestamp 137847 7276138> (DF)

0000: 00 d0 b7 b9 b6 c8 00 02   b3 04 09 dd 08 00 45 00  
0010: 00 62 45 31 40 00 40 06   b1 fe ac 14 00 02 a2 21  
0020: f5 2e c0 0d 15 38 1c af   94 34 a8 1a 1e 39 80 18  
0030: 22 38 19 d5 00 00 01 01   08 0a 00 02 1a 77 00 6f  
0040: 06 6a 51 53 45 4c 45 43   54 20 6f 72 73 74 6e 61  
0050: 6d 0a 46 52 4f 4d 20 6f  
0060: 6e 64 0a 57 48 45 52 45   20 61 67 65 20 3d 20 33  
0070: 33 3b 00 00 62 45 31 40   00 40 06 08 0a 00 02 a2  
0080: 21 f5 2e c0 0d 15 38 1c  
0090: af 94 34 a8 1a 1e 39 80  
00a0: 18 22 38 19 d5 00 00 01  
00b0: 01 08 0a 00 02 1a 77 00  
00c0: 6f 06 6a 51 53 45 4c 45  
00d0: 43 54 20 6f 72 73 74 6e  
00e0: 61 6d 0a 46 52 4f 4d 20  
00f0: 6f 6e 64 0a 57 48 45 52  
0100: 45 20 61 67 65 20 3d 20  
0110: 33 33 3b 00
FindExec: found "/var/local/postgres=./bin/postgres" using argv[0]
DEBUG: connection: host=[local] user=postgres database=test
DEBUG: InitPostgres
DEBUG: StartTransactionCommand
DEBUG: query: SELECT firstname FROM friend WHERE age = 33;

[ query is processed ]
DEBUG: ProcessQuery
DEBUG: CommitTransactionCommand
DEBUG: proc_exit(0)
DEBUG: shmem_exit(0)
DEBUG: exit(0)
Backend Flowchart

Main

Postmaster

Postgres

Libpq

Parse Statement

Traffic Cop

Utility

Command

Rewrite Query

Generate Paths

Optimal Path

Generate Plan

Execute Plan

SELECT, INSERT, UPDATE, DELETE, MERGE

CREATE, CREATE TABLE, COPY

Utilities

Catalog

Storage Managers

Access Methods

Nodes / Lists
Backend Flowchart — Magnified

1. Parse Statement
2. Traffic Cop
3. Rewrite Query
4. Generate Paths
5. Optimal Path
6. Generate Plan
7. Execute Plan

Utility

Command

SELECT, INSERT, UPDATE, DELETE, MERGE
e.g. CREATE TABLE, COPY
Statistics — Part 1

PARSER STATISTICS
system usage stats:
- Elapsed time: 0.000002 seconds
- User time: 0.000001 seconds
- System time: 0.000000 seconds

Filesystem blocks in/out:
- 0/0 [0/1]

Page faults/reclaims:
- 0 [0]

Swaps:
- 0 [0]

Signals received:
- 0 [0]

Messages received/sent:
- 0/0 [2/6]

Voluntary/involuntary context switches:
- 0/0 [2/2]

Postgres usage stats:
- Shared blocks: 0 read, 0 written, buffer hit rate = 0.00%
- Local blocks: 0 read, 0 written, buffer hit rate = 0.00%
- Direct blocks: 0 read, 0 written

PARSE ANALYSIS STATISTICS
system usage stats:
- Elapsed time: 0.000002 seconds
- User time: 0.000001 seconds
- System time: 0.000000 seconds

Filesystem blocks in/out:
- 0/0 [0/1]

Page faults/reclaims:
- 0 [0]

Swaps:
- 0 [0]

Signals received:
- 0 [0]

Messages received/sent:
- 0/0 [2/6]

Voluntary/involuntary context switches:
- 0/0 [2/2]

Postgres usage stats:
- Shared blocks: 1 read, 0 written, buffer hit rate = 96.88%
- Local blocks: 0 read, 0 written, buffer hit rate = 0.00%
- Direct blocks: 0 read, 0 written
Statistics — Part 2

**Rewriter Statistics**

```plaintext
system usage stats:
  0.000002 elapsed 0.000000 user 0.000002 system sec
[0.000000 user 0.049968 sys total]
  0/0 [0/1] filesystem blocks in/out
  0/0 [0/0] page faults/reclaims, 0 [0] swaps
  0 [0] signals rcvd, 0/0 [2/2] messages rcvd/sent
  0/0 [2/6] voluntary/involuntary context switches
```

**Postgres Usage Stats:**

- **Shared blocks**: 0 read, 0 written, buffer hit rate = 0.00%
- **Local blocks**: 0 read, 0 written, buffer hit rate = 0.00%
- **Direct blocks**: 0 read, 0 written

**Planner Statistics**

```plaintext
system usage stats:
  0.009974 elapsed 0.009985 user -1.999985 system sec
[0.019982 user 0.049955 sys total]
  0/0 [0/1] filesystem blocks in/out
  0/0 [0/0] page faults/reclaims, 0 [0] swaps
  0 [0] signals rcvd, 0/0 [2/2] messages rcvd/sent
  0/0 [2/6] voluntary/involuntary context switches
```

**Postgres Usage Stats:**

- **Shared blocks**: 0 read, 0 written, buffer hit rate = 0.00%
- **Local blocks**: 0 read, 0 written, buffer hit rate = 0.00%
- **Direct blocks**: 0 read, 0 written

**Executor Statistics**

```plaintext
system usage stats:
  0.040004 elapsed 0.039982 user 0.000013 system sec
[0.059964 user 0.049970 sys total]
  0/0 [0/1] filesystem blocks in/out
  0/0 [0/0] page faults/reclaims, 0 [0] swaps
  0 [0] signals rcvd, 0/2 [2/4] messages rcvd/sent
  2/2 [4/8] voluntary/involuntary context switches
```

**Postgres Usage Stats:**

- **Shared blocks**: 2 read, 0 written, buffer hit rate = 83.33%
- **Local blocks**: 0 read, 0 written, buffer hit rate = 0.00%
- **Direct blocks**: 0 read, 0 written

---

**Note:** The statistics are system and PostgreSQL usage details, showing various metrics like elapsed time, user/system usage, filesystem block operations, page faults, signals received, and messages exchanged.
Optimizer

- Scan Methods
- Join Methods
- Join Order
Scan Methods

- Sequential Scan
- Index Scan
- Bitmap Scan
Sequential Scan

Heap

8K
BTree Index Scan

Index

Heap

< Key = >

< Key = >

< Key = >
Bitmap Scan

Index 1  Index 2  Combined
col1 = 'A'  col2 = 'NS'

0 0 0
1 1 1
0 1 0
1 0 0

& =

Table

'A' AND 'NS'
Join Methods

- Nested Loop
  - With Inner Sequential Scan
  - With Inner Index Scan
- Hash Join
- Merge Join
Nested Loop Join with Inner Sequential Scan

Outer

| aag | aay | aar | aai |

Inner

| aai | aag | aas | aar | aay | aaa | aag |

No Setup Required

Used For Small Tables
Nested Loop Join with Inner Index Scan

- **Outer**
  - aag
  - aay
  - aar
  - aai

- **Inner**
  - aai
  - aag
  - aas
  - aar
  - aay
  - aaa
  - aag

- **Index Lookup**

- No Setup Required
- Index Must Already Exist
Hash Join

Must fit in Main Memory
Merge Join

Ideal for Large Tables
An Index Can Be Used to Eliminate the Sort
SELECT part.price 
FROM customer, salesorder, part 
WHERE customer.customer_id = salesorder.customer_id AND salesorder.part = part.part_id
Three-Table Join, Pass 1, Part 1

(2 3 ): $\text{rows}=575$ width=76
path list:
HashJoin $\text{rows}=575$ cost=3.57..41.90
  clauses=(salesorder.part_id = part.part_id)
    SeqScan(2) $\text{rows}=575$ cost=0.00..13.75
    SeqScan(3) $\text{rows}=126$ cost=0.00..3.26
Nestloop $\text{rows}=575$ cost=0.00..1178.70
  SeqScan(2) $\text{rows}=575$ cost=0.00..13.75
  IdxScan(3) $\text{rows}=126$ cost=0.00..2.01
Nestloop $\text{rows}=575$ cost=0.00..1210.28
  pathkeys=((salesorder.customer_id, customer.customer_id) )
    IdxScan(2) $\text{rows}=575$ cost=0.00..45.33
      pathkeys=((salesorder.customer_id, customer.customer_id) )
      IdxScan(3) $\text{rows}=126$ cost=0.00..2.01

cheapest startup path:
Nestloop $\text{rows}=575$ cost=0.00..1178.70
  SeqScan(2) $\text{rows}=575$ cost=0.00..13.75
  IdxScan(3) $\text{rows}=126$ cost=0.00..2.01

cheapest total path:
HashJoin $\text{rows}=575$ cost=3.57..41.90
  clauses=(salesorder.part_id = part.part_id)
    SeqScan(2) $\text{rows}=575$ cost=0.00..13.75
    SeqScan(3) $\text{rows}=126$ cost=0.00..3.26
(1 2):

- **rows** = 575, width = 76
- **HashJoin**
  - **rows** = 575, cost = 3.00 .. 40.75
  - clauses = (salesorder.customer_id = customer.customer_id)
  - SeqScan (2) **rows** = 575, cost = 0.00 .. 13.75
  - SeqScan (1) **rows** = 80, cost = 0.00 .. 2.80

- **MergeJoin**
  - **rows** = 575, cost = 0.00 .. 64.39
  - clauses = (salesorder.customer_id = customer.customer_id)
  - IdxScan (1) **rows** = 80, cost = 0.00 .. 10.88
    - pathkeys = ((salesorder.customer_id, customer.customer_id))
  - IdxScan (2) **rows** = 575, cost = 0.00 .. 45.33
    - pathkeys = ((salesorder.customer_id, customer.customer_id))

**cheapest startup path:**
- **MergeJoin**
  - **rows** = 575, cost = 0.00 .. 64.39
  - clauses = (salesorder.customer_id = customer.customer_id)
  - IdxScan (1) **rows** = 80, cost = 0.00 .. 10.88
    - pathkeys = ((salesorder.customer_id, customer.customer_id))
  - IdxScan (2) **rows** = 575, cost = 0.00 .. 45.33
    - pathkeys = ((salesorder.customer_id, customer.customer_id))

**cheapest total path:**
- **HashJoin**
  - **rows** = 575, cost = 3.00 .. 40.75
  - clauses = (salesorder.customer_id = customer.customer_id)
  - SeqScan (2) **rows** = 575, cost = 0.00 .. 13.75
  - SeqScan (1) **rows** = 80, cost = 0.00 .. 2.80
Three-Table Join, Pass 2, Part 1

\((2\ 3\ 1)\): \texttt{rows}=575 \quad \text{width}=112

\text{path list:}

\text{HashJoin } \texttt{rows}=575 \quad \text{cost}=6.58..68.90
\begin{align*}
\text{clauses} &= (\text{salesorder}.\text{customer_id} = \text{customer}.\text{customer_id}) \\
\text{HashJoin } \texttt{rows}=575 \quad \text{cost}=3.57..41.90
\end{align*}
\begin{align*}
\text{clauses} &= (\text{salesorder}.\text{part_id} = \text{part}.\text{part_id}) \\
\text{SeqScan(2)} &\quad \texttt{rows}=575 \quad \text{cost}=0.00..13.75 \\
\text{SeqScan(3)} &\quad \texttt{rows}=126 \quad \text{cost}=0.00..3.26 \\
\text{SeqScan(1)} &\quad \texttt{rows}=80 \quad \text{cost}=0.00..2.80
\end{align*}
\text{HashJoin } \texttt{rows}=575 \quad \text{cost}=3.57..92.54
\begin{align*}
\text{clauses} &= (\text{salesorder}.\text{part_id} = \text{part}.\text{part_id}) \\
\text{MergeJoin } \texttt{rows}=575 \quad \text{cost}=0.00..64.39
\end{align*}
\begin{align*}
\text{clauses} &= (\text{salesorder}.\text{customer_id} = \text{customer}.\text{customer_id}) \\
\text{IdxScan(1)} &\quad \texttt{rows}=80 \quad \text{cost}=0.00..10.88 \\
\text{pathkeys} &= ((\text{salesorder}.\text{customer_id}, \text{customer}.\text{customer_id}) ) \\
\text{IdxScan(2)} &\quad \texttt{rows}=575 \quad \text{cost}=0.00..45.33 \\
\text{pathkeys} &= ((\text{salesorder}.\text{customer_id}, \text{customer}.\text{customer_id}) ) \\
\text{SeqScan(3)} &\quad \texttt{rows}=126 \quad \text{cost}=0.00..3.26
\end{align*}
\text{HashJoin } \texttt{rows}=575 \quad \text{cost}=3.00..1205.70
\begin{align*}
\text{clauses} &= (\text{salesorder}.\text{customer_id} = \text{customer}.\text{customer_id}) \\
\text{Nestloop } \texttt{rows}=575 \quad \text{cost}=0.00..1178.70
\end{align*}
\begin{align*}
\text{SeqScan(2)} &\quad \texttt{rows}=575 \quad \text{cost}=0.00..13.75 \\
\text{IdxScan(3)} &\quad \texttt{rows}=126 \quad \text{cost}=0.00..2.01 \\
\text{SeqScan(1)} &\quad \texttt{rows}=80 \quad \text{cost}=0.00..2.80
\end{align*}
Three-Table Join, Pass 2, Part 2

MergeJoin $rows=575$ cost=0.00..1229.35
  clauses=(salesorder.customer_id = customer.customer_id)
Nestloop $rows=575$ cost=0.00..1210.28
  pathkeys=((salesorder.customer_id, customer.customer_id) )
    IdxScan(2) $rows=575$ cost=0.00..45.33
      pathkeys=((salesorder.customer_id, customer.customer_id) )
    IdxScan(3) $rows=126$ cost=0.00..2.01
IdxScan(1) $rows=80$ cost=0.00..10.88
  pathkeys=((salesorder.customer_id, customer.customer_id) )

cheapest startup path:
MergeJoin $rows=575$ cost=0.00..1229.35
  clauses=(salesorder.customer_id = customer.customer_id)
Nestloop $rows=575$ cost=0.00..1210.28
  pathkeys=((salesorder.customer_id, customer.customer_id) )
    IdxScan(2) $rows=575$ cost=0.00..45.33
      pathkeys=((salesorder.customer_id, customer.customer_id) )
    IdxScan(3) $rows=126$ cost=0.00..2.01
IdxScan(1) $rows=80$ cost=0.00..10.88
  pathkeys=((salesorder.customer_id, customer.customer_id) )

cheapest total path:
HashJoin $rows=575$ cost=6.58..68.90
  clauses=(salesorder.customer_id = customer.customer_id)
HashJoin $rows=575$ cost=3.57..41.90
  clauses=(salesorder.part_id = part.part_id)
    SeqScan(2) $rows=575$ cost=0.00..13.75
    SeqScan(3) $rows=126$ cost=0.00..3.26
    SeqScan(1) $rows=80$ cost=0.00..2.80
Result Returned

test=> SELECT firstname
    test-> FROM friend
    test-> WHERE age = 33;

    1: firstname (typeid = 1042, len = -1, typmod = 19, byval = f)
    ----
    1: firstname = "Sandy" (typeid = 1042, len = -1, typmod = 19, byval = f)
    ----

    firstname
    -------------
    Sandy
    (1 row)
VACUUM ANALYZE VERBOSE customer;
INFO: vacuuming "pg_catalog.pg_depend"
INFO: index "pg_depend_depender_index" now contains 3616 row versions in 19 pages
DETAIL: 0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO: index "pg_depend_reference_index" now contains 3616 row versions in 23 pages
DETAIL: 0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO: "pg_depend": found 0 removable, 3616 nonremovable row versions in 25 pages
DETAIL: 0 dead row versions cannot be removed yet.
There were 9 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/-1.99u sec elapsed 0.00 sec.
INFO: analyzing "pg_catalog.pg_depend"
INFO: "pg_depend": 25 pages, 3000 rows sampled, 3625 estimated total rows
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>starelid</td>
<td>16416</td>
</tr>
<tr>
<td>staattnum</td>
<td>4</td>
</tr>
<tr>
<td>stanullfrac</td>
<td>0</td>
</tr>
<tr>
<td>stawidth</td>
<td>22</td>
</tr>
<tr>
<td>stadistinct</td>
<td>-0.4244</td>
</tr>
<tr>
<td>stakind1</td>
<td>1</td>
</tr>
<tr>
<td>stakind2</td>
<td>2</td>
</tr>
<tr>
<td>stakind3</td>
<td>3</td>
</tr>
<tr>
<td>stakind4</td>
<td>0</td>
</tr>
<tr>
<td>staop1</td>
<td>98</td>
</tr>
<tr>
<td>staop2</td>
<td>664</td>
</tr>
<tr>
<td>staop3</td>
<td>664</td>
</tr>
<tr>
<td>staop4</td>
<td>0</td>
</tr>
<tr>
<td>stanumbers1</td>
<td><code>{0.146658, 0.027904, 0.023625, 0.04375, 0.027125, 0.027125, 0.027125, 0.014925, 0.0142764, 0.0123297}</code></td>
</tr>
<tr>
<td>stanumbers2</td>
<td></td>
</tr>
<tr>
<td>stanumbers3</td>
<td><code>{0.145569}</code></td>
</tr>
<tr>
<td>stanumbers4</td>
<td></td>
</tr>
<tr>
<td>stavalues1</td>
<td><code>{I/O, equal, &quot;not equal&quot;, less-than, greater-than, greater-than-or-equal, less-than-or-equal, subtract, multiply, add}</code></td>
</tr>
<tr>
<td>stavalues2</td>
<td><code>&quot;(Block, offset), physical location of tuple&quot;, &quot;absolute value&quot;, &quot;btree less-equal-greater&quot;, &quot;convert int2 to float4&quot;, &quot;deparse an encoded expression&quot;, &quot;format int8 to text&quot;, &quot;is opclass visible in search path?&quot;, &quot;matches LIKE expression&quot;, &quot;print type names of oidvector field&quot;, &quot;sine&quot;, &quot;18 digit integer, 8-byte storage&quot;</code></td>
</tr>
<tr>
<td>stavalues3</td>
<td></td>
</tr>
<tr>
<td>stavalues4</td>
<td></td>
</tr>
</tbody>
</table>
EXPLAIN

EXPLAIN SELECT name FROM customer;
NOTICE: QUERY PLAN:

Seq Scan on customer (cost=0.00..225.88 rows=12288 width=34)
EXPLAIN ANALYZE

EXPLAIN ANALYZE SELECT name FROM customer;
NOTICE:  QUERY PLAN:

Seq Scan on customer (cost=0.00..225.88 rows=12288 width=34) (actual time=0.21..205.20 rows=12288 loops=1)
Total runtime: 249.10 msec
EXPLAIN INSERT INTO warehouse_tmp
(uri, expression, n, relevance, spid_measure, size, title, sample)
SELECT d.uri, dn.expression, n.n, dn.relevance, d.spid_measure,
d.size, d.title, dn.sample
FROM document as d
INNER JOIN (document_n_gram AS dn
    INNER JOIN n_gram AS n
    ON (dn.expression = n.expression))
    ON (d.uri = dn.uri)
ORDER BY dn.expression, n.n;
NOTICE: QUERY PLAN:

Subquery Scan *SELECT* (cost=3895109.07..3895109.07 rows=1009271 width=886)
  -> Sort (cost=3895109.07..3895109.07 rows=1009271 width=886)
    -> Hash Join (cost=1155071.81..2115045.12 rows=1009271 width=886)
      -> Merge Join (cost=1154294.92..1170599.85 rows=1009271 width=588)
        -> Sort (cost=1001390.67..1001390.67 rows=1009271 width=439)
          -> Seq Scan on document_n_gram dn

      -> Sort (cost=152904.25..152904.25 rows=466345 width=149)
        -> Seq Scan on n_gram n (cost=0.00..12795.45 rows=466345 width=149)
    -> Hash (cost=767.71..767.71 rows=3671 width=298)
      -> Seq Scan on document d (cost=0.00..767.71 rows=3671 width=298)
EXPLAIN SELECT cs.entity_id as region, r.name, cs.status, count(*)
FROM region r inner join
    (SELECT DISTINCT findregion(entity_id) AS entity_id, status
     FROM current_status
     ORDER BY 1
     ) AS cs on r.region_id = cs.entity_id
GROUP BY region, r.name, cs.status;

NOTICE: QUERY PLAN:
Aggregate (cost=13688.40..14338.40 rows=6500 width=24)
  -> Group (cost=13688.40..14175.90 rows=65000 width=24)
    -> Sort (cost=13688.40..13688.40 rows=65000 width=24)
      -> Merge Join (cost=7522.19..7674.94 rows=65000 width=24)
        -> Index Scan using region_pkey on region r
           (cost=0.00 59.00 rows=1000 width=16)
        -> Sort (cost=7522.19..7522.19 rows=6500 width=8)
          -> Subquery Scan cs (cost=6785.54..7110.54
             rows=65 width=8)
            -> Unique (cost=6785.54..7110.54 rows=6500
              with=8)
            -> Sort (cost=6785.54..6785.54 rows=650
              width=8)
          -> Seq Scan on current_status
             (st=0.00..1065.00 rows=65000 width=8)
# - Planner Method Enabling -

-enable_hashagg = true
-enable_hashjoin = true
-enable_indexscan = true
-enable_mergejoin = true
-enable_nestloop = true
-enable_seqscan = true
-enable_sort = true
-enable_tidscan = true

# - Planner Cost Constants -

-effective_cache_size = 1000  # typically 8KB each
-random_page_cost = 4  # units are one sequential page fetch cost
-cpu_tuple_cost = 0.01  # (same)
-cpu_index_tuple_cost = 0.001  # (same)
-cpu_operator_cost = 0.0025  # (same)
# - Genetic Query Optimizer -

#geqo = true
#geqo_threshold = 11
#geqo_effort = 1
#geqo_generations = 0
#geqo_pool_size = 0         # default based on tables in statement,
                            # range 128-1024
#geqo_selection_bias = 2.0  # range 1.5-2.0

# - Other Planner Options -

#default_statistics_target = 10 # range 1-1000
#from_collapse_limit = 8
#join_collapse_limit = 8       # 1 disables collapsing of explicit JOINs
Storage

https://www.flickr.com/photos/mirandala/
File Structure

8K

Page
Page
Page
Page
Page
Page
Page Structure

8K

Page Header | Item | Item | Item

Tuple | Tuple | Tuple | Special
Index Page Structure

Internal

Leaf

Heap

M C I A G E P K W L

E

L

Special

A C

Special

Page Header Item Item Item

Special

Page Header Item Item Item

>= N

< F

< N

Special

Page Header Item Item Item

Page Header Item Item Item
Cluster
CREATE TABLE customer (id SERIAL, name TEXT);
NOTICE: CREATE TABLE will create implicit sequence 'customer_id_seq' for SERIAL column 'customer.id'
CREATE INDEX customer_id_index ON customer (id);

CLUSTER customer USING customer_id_index;
Index Types (Access Methods)

- BRIN
- BTree
- Hash
- GIN (generalized inverted index)
- GiST (generalized search tree)
- SP-GiST (space-partitioned GiST)
Tablespaces For Database I/O Balancing
Tablespaces For Table and Index I/O Balancing

Disk 1  Disk 2  Disk 3

tab1  tab2  index  constraint
Table I/O Balancing Partitions

Range and list partitioning is also possible.
Caches

- System Cache
- Relation Information Cache
- File Descriptor Cache
Shared Memory

- Proc structure
- Lock structure
- Buffer structure
- Free space map
Query Tips

- COPY vs. INSERT
- LIMIT vs. CURSOR
- TRUNCATE vs. DELETE
- Expression indexes
- Partial indexes
- Prepared queries
- INTERSECT vs. AND (selfjoin)
- UNION vs. OR
Conclusion

https://momjian.us/presentations

https://www.flickr.com/photos/143948408@N03/