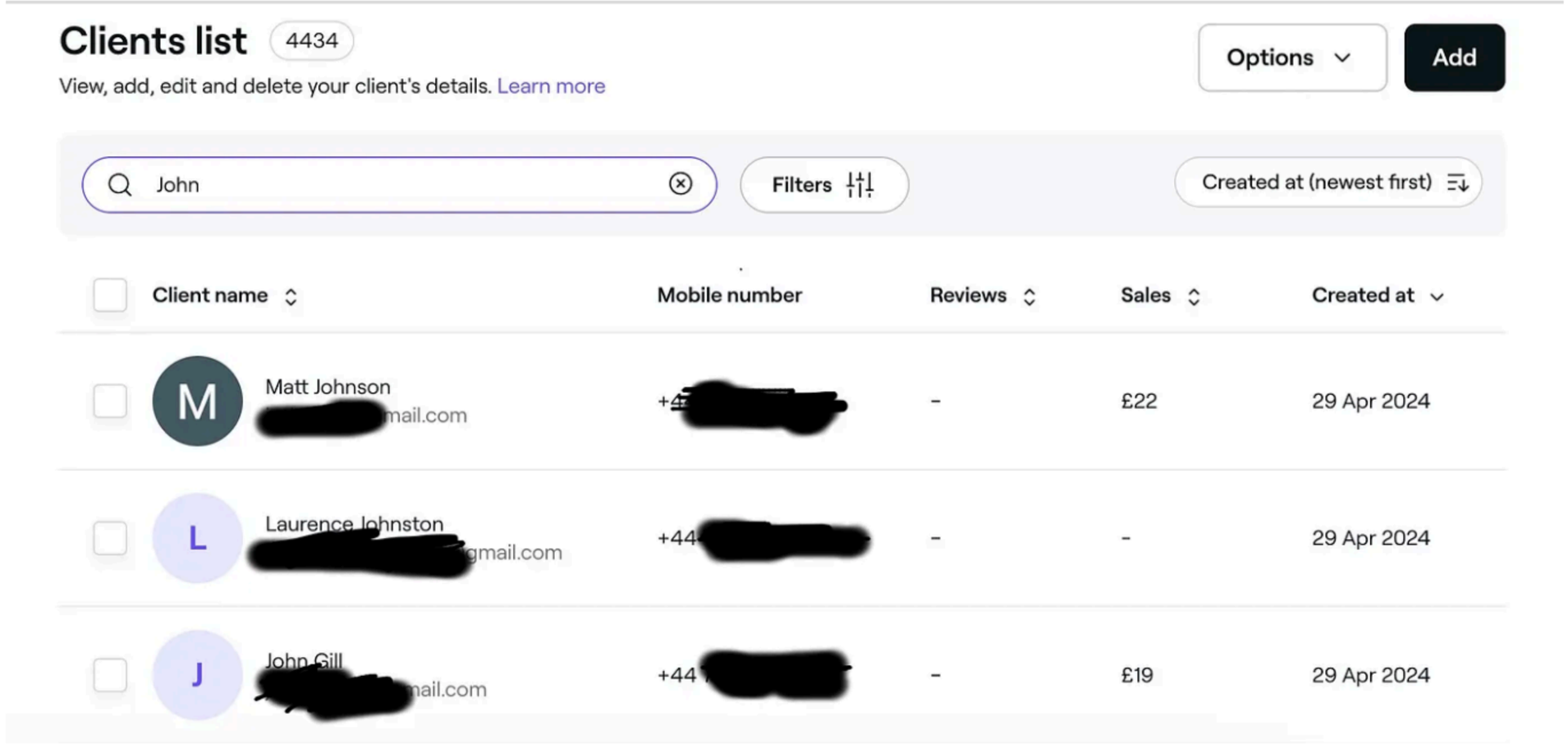
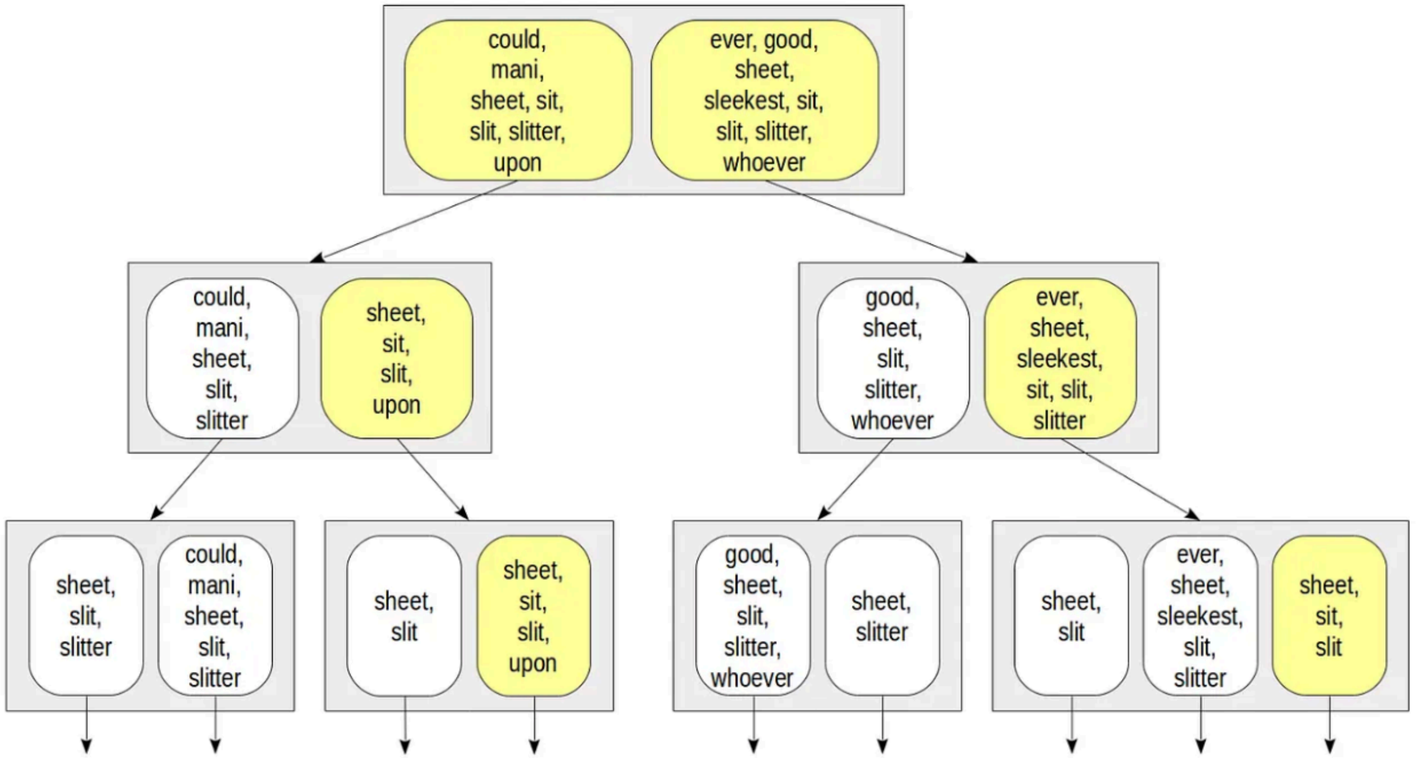
How Bóbr Broke My Postgres Search: ARM vs x86 in 256 Seconds

Anton Borisov

Search As You Go



GIST index



GiST index logical structure. Courtesy of PostgresPro.

Search Bóbr

```
anton.borisov@192 system % psql --host=graviton --port=5432
psql (14.9 (Homebrew), server 13.8)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 25
Type "help" for help.
postgres=> SELECT * FROM customers where name ILIKE '%Bóbr%' limit 10;
     name
-----------------------
 FabM0sWwFø BóbrcOZ5KH
 ö58rbóbRE4 jBRiUcdI9T
 SBóbrhkfEt hBwTjkQ8öS
 TslpZUp6fF öBóBRYS0lT
 Bóbr Ale Bydle
(5 rows)
```

Search Bóbr again

```
anton.borisov@192 system % psql --host=x86 --port=5432
psql (14.9 (Homebrew), server 13.8)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 25
Type "help" for help.
postgres=> SELECT * FROM customers where name ILIKE '%Bóbr%' limit 10;
 name
------
(0 rows)
```

How is it sorted?

```
Master:
postgres=> SELECT show_trgm('Bóbr');
         show_trgm
-----------------------------------------
 {0x89194c," b","br ",0x707c72,0x7f7849}
(1 row)

Replica:
postgres=> SELECT show_trgm('Bóbr');
         show_trgm
-----------------------------------------
 {" b","br ",0x707c72,0x7f7849,0x89194c}
(1 row)
```

CMPCHAR

```
#define CMPCHAR(a,b) ( ((a)==(b)) ? 0 : ( ((a)<(b)) ? -1 : 1 ) )
#define CMPPCHAR(a,b,i)  CMPCHAR( *(((const char*)(a))+i), *(((const char*)(b))+i) )
#define CMPTRGM(a,b) ( CMPPCHAR(a,b,0) ? CMPPCHAR(a,b,0) : ( CMPPCHAR(a,b,1) ? CMPPCHAR(a,b,1) : CMPPCHAR(a,b,2) ) )

#define CPTRGM(a,b) do {    \
 *(((char*)(a))+0) = *(((char*)(b))+0); \
 *(((char*)(a))+1) = *(((char*)(b))+1); \
 *(((char*)(a))+2) = *(((char*)(b))+2); \
} while(0)
```

Aha Moment!

X86 vs ARM

Default Char is Signed

Default Char is Unsigned

Fixed Now!

Commit `dfd8e6c`

MasahikoSawada committed on Feb 21, 2025 · ✓ 7 / 10

```
Fix an issue with index scan using pg_trgm due to char signedness on different architectures.

GIN and GiST indexes utilizing pg_trgm's opclasses store sorted
trigrams within index tuples. When comparing and sorting each trigram,
pg_trgm treats each character as a 'char[3]' type in C. However, the
char type in C can be interpreted as either signed char or unsigned
char, depending on the platform, if the signedness is not explicitly
specified. Consequently, during replication between different CPU
architectures, there was an issue where index scans on standby servers
could not locate matching index tuples due to the differing treatment
of character signedness.

This change introduces comparison functions for trgm that explicitly
handle signed char and unsigned char. The appropriate comparison
function will be dynamically selected based on the character
signedness stored in the control file. Therefore, upgraded clusters
can utilize the indexes without rebuilding, provided the cluster
upgrade occurs on platforms with the same character signedness as the
original cluster initialization.

The default char signedness information was introduced in 44fe30f,
so no backpatch.

Reviewed-by: Noah Misch <noah@leadboat.com>
Discussion: https://postgr.es/m/CB11ADBC-0C3F-4FE0-A678-666EE80CBB07%40amazon.com
```

Still Gotcha!

Commit `44fe30f`

MasahikoSawada committed on Feb 21, 2025 · ✓ 7 / 10

```
Add default_char_signedness field to ControlFileData.

The signedness of the 'char' type in C is
implementation-dependent. For instance, 'signed char' is used by
default on x86 CPUs, while 'unsigned char' is used on aarch
CPUs. Previously, we accidentally let C implementation signedness
affect persistent data. This led to inconsistent results when
comparing char data across different platforms.

This commit introduces a new 'default_char_signedness' field in
ControlFileData to store the signedness of the 'char' type. While this
change does not encourage the use of 'char' without explicitly
specifying its signedness, this field can be used as a hint to ensure
consistent behavior for pre-v18 data files that store data sorted by
the 'char' type on disk (e.g., GIN and GiST indexes), especially in
cross-platform replication scenarios.

Newly created database clusters unconditionally set the default char
signedness to true. pg_upgrade (with an upcoming commit) changes this
flag for clusters if the source database cluster has
signedness=false. As a result, signedness=false setting will become
rare over time. If we had known about the problem during the last
development cycle that forced initdb (v8.3), we would have made all
clusters signed or all clusters unsigned. Making pg_upgrade the only
source of signedness=false will cause the population of database
clusters to converge toward that retrospective ideal.

Bump catalog version (for the catalog changes) and PG_CONTROL_VERSION
(for the additions in ControlFileData).

Reviewed-by: Noah Misch <noah@leadboat.com>
Discussion: https://postgr.es/m/CB11ADBC-0C3F-4FE0-A678-666EE80CBB07%40amazon.com
```

Conclusion

New Clusters have signed chars by default

Old clusters remember the original signedness

Pg_upgrade retains the old cluster's behaviour