

Collations from A to Z

Putting words in order
without losing your mind or your data

Jeff Davis

Jeremy Schneider



PostgreSQL does not include its own string comparison code. It calls external libraries, which were installed & managed separately.

The Backstory, Part 1

The Open Group Base Specifications Issue 7, 2018 edition
IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)
Copyright © 2001-2018 IEEE and The Open Group

NAME

strcoll, strcoll_l - string comparison using collating information

SYNOPSIS

```
#include <string.h>
```

```
int strcoll(const char *s1, const char *s2);
```

```
[CX] ☒ int strcoll_l(const char *s1, const char *s2,  
                   locale_t locale); ☒
```

DESCRIPTION

For *strcoll()*: [CX] ☒ The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard. ☒

The Backstory, Part 2 - Six Years Ago

Widespread encounters:

- Queries giving incorrect results
data appears to be lost
- Inserting records with duplicate primary keys
unique constraints not enforced correctly
- Mysterious crashes
in one case during WAL replay, preventing a DB from doing crash recovery

Caused by changes in sort order

23 Things I Completely Got Wrong

about putting words in order

during 7 years working with Postgres



INCORRECT

1. Putting words in order is simple

compare each character
from beginning to end (memcmp)

Linguistic Collation is Complex

Contractions: two (or more) characters sort as if they were a single base letter. In *Table 4*, *CH* acts like a single letter sorted after *C*.

Expansions: a single character sorts as if it were a sequence of two (or more) characters. In *Table 4*, an *Œ* ligature sorts as if it were the sequence of *O + E*.

Backwards Accent: In row 1 of *Table 5*, the first accent difference is on the *o*, so that is what determines the order. In some French dictionary ordering traditions, however, it is the *last* accent difference that determines the order, as shown in row 2.

Table 5. Backward Accent Ordering

Normal Accent Ordering	cote < coté < cōte < côté
Backward Accent Ordering	cote < côte < cote < côté

Table 4. Context Sensitivity

Contractions	H < Z, <i>but</i> CH > CZ
Expansions	OE < Œ < OF
Both	カー < カア, <i>but</i> キー > キア

<https://www.cybertec-postgresql.com/en/case-insensitive-pattern-matching-in-postgresql/>

The difficult case of German soccer

The ICU documentation details why correct case-insensitive pattern matching is difficult. A good example is the German letter "ß", which traditionally doesn't have an upper-case equivalent. So with *good* German collations (the collation from the GNU C library is not good in that respect), you will get a result like this:

```
1 | SELECT upper('Fußball' COLLATE "de-DE-x-icu");
2 |
3 |      upper
4 | -----
5 |      FUSSBALL
6 | (1 row)
```

Now what would be the correct result for the following query in a case-insensitive collation?

```
1 | SELECT 'Fußball' LIKE 'FUS%';
```

You could argue that it should be `TRUE`, because that's what you'd get for `upper('Fußball') LIKE 'FUS%'`. On the other hand,

```
1 | SELECT lower('FUSSBALL' COLLATE "de-DE-x-icu");
2 |
3 |      lower
4 | -----
5 |      fussball
6 | (1 row)
```

so you could just as well argue that the result should be `FALSE`. The ICU library goes with the second solution for simplicity. Either solution would be difficult to implement in PostgreSQL, so we have given up

<https://www.unicode.org/reports/tr10/>



INCORRECT

2. The way computers and people put words in order doesn't change

Must be a mistake by maintainers of the external library?

"Correct" Ordering Does Change

French (2010)

<https://unicode-org.atlassian.net/browse/CLDR-2905>

Currently we have backwards secondary sorting on for French (and only for French).

However, there is a significant cost to this setting in terms of performance, and no real advantage to users in terms of function.

- There is little reason to believe that the average, even well-educated, francophone is aware or cares about these rules.
- They affect very, very few cases (cote, peche, etc).
- From all evidence, the original research behind the rules was based on a selection of dictionaries where a different selection would have given a different answer.

The plan is to issue a PRI for this change.

Tibetan (2021)

<https://unicode-org.atlassian.net/browse/CLDR-9895>



Élie Roux July 8, 2021 at 12:28 AM

After a discussion with Peter, I realize I should add some context here (mostly duplicate from the presentation Peter pointed to, just for reference):

- the rules have been developed and are documented on [GitHub - eroux/tibetan-collation: Collation algorithm for Tibetan](#)
- they follow peer-reviewed articles (cited in the git repo)
- they are tested against a lot of edge cases (there's a Python test script in the repo)
- they have been adopted by GLibC
- I'm the lead developer of the Buddhist Digital Resource Center ([Home - Buddhist Digital Resource Center](#)), author of this article about the Tibetan syllabic components: [Algorithmic description of the decomposition and checking of a Classical Tibetan syllable](#) and co-author of these articles on Tibetan NLP: [A Optimisation of the Largest Annotated Tibetan Corpus Combining Rule-based, Memory-based, and Deep-learning](#) and <https://aclanthology.org/2020.tlt-1.3.pdf>



Peter Edberg July 7, 2021 at 10:35 AM

Also see this preso about various Tibetan issues/proposals for CLDR & ICU: [Tibetan in CLDR & ICU](#)

wiki.postgresql.org/wiki/Collations



To quote from [Unicode Technical Standard](#):

"Over time, collation order will vary: there may be fixes needed as more information becomes available about languages; there may be new government or industry standards for the language that require changes; and finally, new characters added to the Unicode Standard will interleave with the previously-defined ones. This means that collations must be carefully versioned."

Swedish (2022)

<https://unicode-org.atlassian.net/browse/CLDR-3059>

Projects / [CLDR](#) / [CLDR-3059](#)



Henri Sivonen May 2, 2022 at 12:35 AM

This issue bundles things that can be addressed separately from each other. I file [CLDR-15603: Align Swedish \(sv\) collation naming with other \(non-zh\) languages](#) **DONE** about the Swedish collation renaming.


[CLDR-7088: Swedish collation](#) **ACCEPTED** also mentions the renaming but focuses on w and v but in the opposite way compared to this issue.



Henri Sivonen May 1, 2022 at 11:56 PM

What's the evidence that users expect or want v and w to match in search?

As two completely unscientific (N=1 for Finnish, and N=1 for Swedish) anecdotes: I had lived as a Finnish native-speaker in Finland for about 39 years (and more than half of that having been interested in things of this nature) before I learned, by reading CLDR sources, about the notion of v and w having been formerly primary-equal in a Finnish standard. After learning, again by reading CLDR sources, that CLDR also matches v and w for Swedish search, I asked the first Swede who I could ask about whether they expected this, and they didn't expect this, either.



It's so interesting, first human languages were leading changes on computers. We added rules to computers reflecting how we speak or write.

Now – like in the French example – the rule was decided the way computing is done.

The order of things can change. Now computers are impacting our natural language.

Gülçin Yıldırım Jelínek

Paraphrasing Peter Eisentraut during an interview with him

The Builders: A Postgres Podcast, Episode 1, Dec 12 2023

INCORRECT

3. Changing sort order is rare

Rare Large Change Got Everyone's Attention

← → ↻ 🌐 postgresql.verite.pro/blog/2018/08/27/glibc-upgrade.html ☆ 📄 😊 ⋮

PostgreSQL Notes - Daniel Vérité

About

Beware of your next glibc upgrade

Aug 27, 2018

[GNU libc 2.28](#), released on August 1, 2018, has among its new features a major update of its Unicode locale data with new collation information.

From the [announcement](#):

The localization data for ISO 14651 is updated to match the 2016 Edition 4 release of the standard, this matches data provided by Unicode 9.0.0. This update introduces significant improvements to the collation of Unicode characters. [...] With the update many locales have been updated to take advantage of the new collation information. The new collation information has increased the size of the compiled locale archive or binary locales.

For Postgres databases using language and region-sensitive collations, which tend to be the default nowadays, it means that certain strings might sort differently after this upgrade. A critical consequence is that **indexes** that depend on such collations **must be rebuilt** immediately after the upgrade. Servers in WAL-based/streaming replication setups should also be upgraded together since a standby must run the same libc/locales as its primary.

The risk otherwise is [index corruption issues](#), as mentioned for instance in these two threads from postgresql-general: [“Issues with german locale on GentOS 5 6 7”](#) and [“The dangers of streaming across](#)

2018

Rare Large Change Got Everyone's Attention

DANGER: glibc 2.28 has a scary and major collation change

Even pure ASCII strings change sort order!

- Debian 10 (buster)
- Ubuntu 18.04
- RHEL 8
- SLE15 Service Pack 3

https://wiki.postgresql.org/wiki/Locale_data_changes

Collation Torture Test

github.com/ardentperf/glibc-unicode-sorting

Data to answer the questions:

Is this really a problem?

How common are sort order changes?

- 10 years of historical versions
- Ubuntu and RHEL
- All assigned code points

The screenshot shows the GitHub repository page for 'ardentperf/glibc-unicode-sorting'. The repository is public and has 12 stars, 3 forks, and 3 unwatchers. The main branch is 'main'. The repository contains 36 commits and 11 files. The files listed are: `_rhel`, `_ubuntu-icu`, `_ubuntu`, `.gitignore`, `README.md`, `diff.sh`, `filter.sh`, `run-icu.sh`, `run.sh`, `table.sh`, `test-host-icu.sh`, and `test-host.sh`. The repository also has a README.md file which is partially visible, showing the title 'Collation Changes Across Linux Versions' and sections for 'Methodology' and 'GNU C Library'.

File	Description	Last Commit
<code>_rhel</code>	Generalize scripts and add RHEL support, add two new string patter...	2 years ago
<code>_ubuntu-icu</code>	results of ICU tests, updated comments in ICU scripts	3 months ago
<code>_ubuntu</code>	cosmetic updates: summary locale column before detail; add link to f...	8 months ago
<code>.gitignore</code>	further significant updates: split diff to separate script, use git d...	9 months ago
<code>README.md</code>	results of ICU tests, updated comments in ICU scripts	3 months ago
<code>diff.sh</code>	add ubuntu-icu and bump up unicode version	3 months ago
<code>filter.sh</code>	add a script that can filter to only the pure ISO-8859-1 strings, as ...	8 months ago
<code>run-icu.sh</code>	results of ICU tests, updated comments in ICU scripts	3 months ago
<code>run.sh</code>	bugfix: heart emoji in source code mistakenly included U+FE0F varia...	9 months ago
<code>table.sh</code>	add ubuntu-icu, support MacOS (stat utility args), bump up unicode ...	3 months ago
<code>test-host-icu.sh</code>	results of ICU tests, updated comments in ICU scripts	3 months ago
<code>test-host.sh</code>	few comment updates; use larger instances, search deprecated AMIs	8 months ago

286,654



91



26 million

unicode code points

string patterns

strings



S-199: 🍷	S-300: 🍷BB	S-330: 🍷🍷B	S-400: 🍷🍷BB
S-200: 🍷B	S-301: 🍷00	S-331: 🍷🍷0	S-401: 🍷🍷00
S-201: 🍷0	S-302: 🍷33	S-332: 🍷🍷3	S-402: 🍷🍷33
S-202: 🍷3	S-303: 🍷..	S-333: 🍷🍷.	S-403: 🍷🍷..
S-203: 🍷.	S-304: 🍷	S-334: 🍷🍷.	S-404: 🍷🍷
S-204: 🍷	S-305: 🍷様様	S-335: 🍷🍷様	S-405: 🍷🍷様様
S-205: 🍷様	S-306: 🍷クク	S-336: 🍷🍷ク	S-406: 🍷🍷クク
S-206: 🍷ク	S-310: B🍷B	S-340: 🍷B🍷	S-410: B🍷🍷B
S-210: B🍷	S-311: 0🍷0	S-341: 🍷0🍷	S-411: 0🍷🍷0
S-211: 0🍷	S-312: 3🍷3	S-342: 🍷3🍷	S-412: 3🍷🍷3
S-212: 3🍷	S-313: .🍷.	S-343: 🍷.🍷	S-413: .🍷🍷.
S-213: .🍷	S-314: 🍷	S-344: 🍷🍷	S-414: 🍷🍷
S-214: 🍷	S-315: 🍷🍷様	S-345: 🍷🍷様	S-415: 🍷🍷🍷様
S-215: 🍷様	S-316: 🍷🍷ク	S-346: 🍷🍷ク	S-416: 🍷🍷🍷ク
S-216: 🍷ク	S-320: BB🍷	S-350: B🍷🍷	S-420: BB🍷🍷
S-219: 🍷🍷	S-321: 00🍷	S-351: 0🍷🍷	S-421: 00🍷🍷
	S-322: 33🍷	S-352: 3🍷🍷	S-422: 33🍷🍷
	S-323: ..🍷	S-353: .🍷🍷	S-423: ..🍷🍷
	S-324: 🍷	S-354: 🍷🍷	S-424: 🍷🍷
	S-325: 🍷🍷様	S-355: 🍷🍷様	S-425: 🍷🍷🍷様
	S-326: 🍷🍷ク	S-356: 🍷🍷ク	S-426: 🍷🍷🍷ク
		S-380: 3B🍷	S-480: 3B🍷B
		S-399: 🍷🍷🍷	S-499: 🍷🍷🍷
			S-580: BB🍷🍷 [tab]
			S-581: [tab]BB🍷🍷
			S-582: BB-🍷🍷
			S-583: 🍷🍷🍷🍷🍷
			S-584: 🍷🍷.33
			S-585: 3B-🍷B
			S-599: 🍷🍷🍷🍷

#	CodePoint	UnicodeBlock	PatternID	String	PositionChange
001c97	1C90	S-406	🍷🍷クク	-11431135,46+11444795,87:-199419	
001c97	1C90	S-410	B🍷🍷B	-8465481,9+8479910,13:-8959176,190+8	
001c97	1C90	S-411	0🍷🍷0	-10240551,9+10255120,13:-10734678,190+8	
001c97	1C90	S-412	3🍷🍷3	-5845196,9+5857750,13:-6339374,190+8	
001c97	1C90	S-413	.🍷🍷.	-2375649,9+2377580,13:-2869104,190+8	
001c97	1C90	S-414	🍷🍷	-1134663,190+1137631,6:-641130,9+6	
001c97	1C90	S-415	🍷🍷🍷🍷	-15846114,9+15875242,13:-1691821	
001c97	1C90	S-416	🍷🍷🍷🍷	-12305289,9+12331858,13:-1337769	
001c97	1C90	S-420	BB🍷🍷	-7358088,9+7373833,13:-7851695,190+8	
001c97	1C90	S-421	00🍷🍷	-9684114,263+9699303,357:-9931353	
001c97	1C90	S-422	33🍷🍷	-4133299,263+4146219,357:-4380496	
001c97	1C90	S-423	..🍷🍷	-1780032,263+1781363,357:-2026823	
001c97	1C90	S-424	🍷🍷	-292833,98+294569,6:-46002,263+470	
001c97	1C90	S-425	🍷🍷🍷🍷	-16135223,263+16166487,357:-1638	
001c97	1C90	S-426	🍷🍷🍷🍷	-12417471,263+12445940,357:-1266	
001c97	1C90	S-480	3B🍷B	-5284151,263+5297298,357:-5530914	
001c97	1C90	S-481	3B-🍷	-4711898,263+4725055,357:-4958661	
001c97	1C90	S-499	🍷🍷🍷	-11431135,46+11444795,87:-1994190	
001c97	1C90	S-582	BB-🍷🍷	-7035764,134+7050593,181:-7159145	
001c97	1C90	S-583	🍷🍷🍷🍷🍷	-3707164,134+3713766,181:-38305	
001c97	1C90	S-584	🍷🍷.33	-11431135,46+11444795,87:-1994190	
001c97	1C90	S-585	3B-🍷B	-4711898,263+4725055,357:-4958661	
001c97	1C90	S-599	🍷🍷🍷🍷	-11431135,46+11444795,87:-1994190	
001c98	1C90	S-199	🍷	-11431258,46+11444959,87:-19941990,18	
001c98	1C90	S-200	🍷B	-11431258,46+11444959,87:-19941990,18	
001c98	1C90	S-201	🍷0	-11431258,46+11444959,87:-19941990,18	
001c98	1C90	S-202	🍷3	-11431258,46+11444959,87:-19941990,18	
001c98	1C90	S-203	🍷.	-11431258,46+11444959,87:-19941990,18	
001c98	1C90	S-204	🍷	-11431258,46+11444959,87:-19941990,18	
001c98	1C90	S-205	🍷🍷	-11431258,46+11444959,87:-19941990,18	
001c98	1C90	S-206	🍷B	-11431258,46+11444959,87:-19941990,18	
001c98	1C90	S-210	B🍷	-8465493,9+8479926,13:-8959176,190+8	
001c98	1C90	S-211	0🍷	-10240563,9+10255136,13:-10734678,190+8	
001c98	1C90	S-212	3🍷	-5845208,9+5857766,13:-6339374,190+8	
001c98	1C90	S-213	.🍷	-2375661,9+2377596,13:-2869104,190+8	



**Every single RHEL major and Ubuntu LTS
in the last 10 years has sort order changes
except for Ubuntu 14.04**

Collation Torture Test

```
github.com/ardentperf/glibc-unicode-sorting/blob/main/run-icu.sh#L65  
glibc-unicode-sorting / run-icu.sh  
Code Blame 216 lines (194 loc) · 11.5 KB  
65 sudo su - postgres -c "psql -v ON_ERROR_STOP=on" <<EOF  
66  
67 \\timing  
68  
69 drop table if exists unicode_spec;  
70 create table unicode_spec(f1 text,f2 text,f3 text);  
71  
72 copy unicode_spec from program 'curl -ks https://www.unicod  
73  
74 drop table if exists unicode_data;  
75 create table unicode_data(d1 text);  
76  
77 create or replace function insert_codepoint(cp int) return  
78 begin  
79 insert into unicode_data values( chr(cp) ); -- 199  
80  
81 insert into unicode_data values( chr(cp)||'B' ); --  
82 insert into unicode_data values( chr(cp)||'0' ); --  
83 insert into unicode_data values( chr(cp)||'3' ); -- 202  
84 insert into unicode_data values( chr(cp)||',' ); -- 203
```

The Problem
A Solution

Introduction
One Level Deeper
Problem Summary

Collation Torture Test - on RHEL 7

```
CREATE TABLE unsorted_table(strings text);  
\copy unsorted_table from /home/ec2-user/formated-unicode.txt (format csv)  
VACUUM FREEZE ANALYZE unsorted_table;  
\timing  
WITH t AS (SELECT strings FROM unsorted_table ORDER BY strings)  
SELECT md5(string_agg(t.strings,NULL)) FROM t;  
md5
```

```
-----  
7b2be833bc1893742f4b16d76d17e130  
(1 row)
```

Time: 176505.256 ms (02:56.505)

See: <https://github.com/ardentperf/glibc-unicode-sorting>

And: <https://joeconway.com/presentations/formated-unicode.txt>



Joe Conway

PGCon 2023

11/44

INCORRECT

4. Changing sort order is intentional

Unintentional Changes

In 2014, a 300-line commit to refactor an internal cache for perf reasons changed sort order of **22,000 code points** (mostly CJK) in the collation torture test between glibc versions 2.19 and 2.21

sourceware.org/git/?p=glibc.git;a=commit;h=0742a... ☆

[git://sourceware.org](#) / [glibc.git](#) / commit +++ git

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#) ? search: re
(parent: [ee54ce4](#)) | [patch](#)

strcoll: improve performance by removing the cache (#15884)

author Leonhard Holz <leonhard.holz@web.de>
Fri, 17 Oct 2014 10:17:23 +0000 (15:47 +0530)
committer Siddhesh Poyarekar <siddhesh@redhat.com>
Fri, 17 Oct 2014 10:17:23 +0000 (15:47 +0530)
commit 0742aef6e52a935f9ccd69594831b56d807feef3
tree ad38b0391baea7c79db50e1f9bbc31c50e0e5b88 [tree](#)
parent ee54ce44cb734f18fec4f6ccdfbe997d2574321e [commit](#) | [diff](#)

strcoll: improve performance by removing the cache (#15884)

this is a path that should solve bug 15884. It complains about the performance of strcoll(). It was found out that the runtime of strcoll() is actually bound to strlen which is needed for calculating the size of a cache that was installed to improve the comparison performance.

The idea for this patch was that the cache is only useful in rare cases (strings of same length and same first-level-chars) and that it would be better to avoid memory allocation at all. To prove this I wrote a performance test bench-strcoll.c with test data in benchtests-strcoll.tar.gz. Also modifications in benchtests/Makefile and localedata/Makefile are necessary to make it work.

After removing the cache the strcoll method showed the predicted behavior (getting slightly faster) in all but the test case for hindi word sorting. This was due the hindi text having much more equal words than the other ones. For equal strings the performance was worse since all comparison levels were run through and from the second level on the cache improved the comparison

INCORRECT

5. Indexes are the only thing corrupted

Users are safe if they rebuild indexes

Possible Corruption After Sort Order Change

<https://ardentperf.com/2023/03/26/did-postgres-lose-my-data/>

```
create table arabic_dictionary_research (  
  word text,  
  crossreferences text,  
  notes text  
) partition by range (word);  
  
create table arabic_dictionary_research_p1 partition of arabic_dictionary_research  
  for values from ('ا') to ('ح');  
create table arabic_dictionary_research_p2 partition of arabic_dictionary_research  
  for values from ('ح') to ('س');  
create table arabic_dictionary_research_p3 partition of arabic_dictionary_research  
  for values from ('س') to ('ج');  
create table arabic_dictionary_research_p4 partition of arabic_dictionary_research  
  for values from ('ج') to ('ز');  
create table arabic_dictionary_research_p5 partition of arabic_dictionary_research  
  default;
```

Possible Corruption After Sort Order Change

Updating an external collation library can cause corruption that isn't noticed until long afterwards.

Can trigger a sort order change:

- OS Upgrade
- Failover and Hot Standby
 - Patroni, Kubernetes, etc
- Distributed Systems

Can be corrupted by version change:

- Indexes
 - All types, not just btree
- Constraints
 - All types, not just unique/primary-key
- Partitions
- FDWs – eg. mergejoin depends on same local/remote ordering
- *Maybe: un-refreshed materialized views, triggers, generated columns? (I'm not sure)*

INCORRECT

6. Users can rebuild the impacted objects

It's inconvenient but at least there is always a "fix"

Hot Standby to Scale Out Reads

← → ↻ 🏠 postgresql.org/message-id/flat/BA6132ED-1F... ☆ 🗂️ | 😄 ⋮

From: Matthew Kelly <mkelly(at)tripadvisor(dot)com>
To: "pgsql-general(at)postgresql(dot)org" <pgsql-general(at)postgresql(dot)org>
Cc: Matthew Spilich <mspilich(at)tripadvisor(dot)com>
Subject: The dangers of streaming across versions of glibc: A cautionary tale
Date: 2014-08-06 21:24:17
Message-ID: BA6132ED-1F6B-4A0B-AC22-81278F5AB81E@tripadvisor.com
Views: [Raw Message](#) | [Whole Thread](#) | [Download mbox](#) | [Resend email](#)
Lists: [pgsql-general](#)

The following is a real critical problem that we ran into here at TripAdvisor, but have yet figured out a clear way to mitigate.

TL;DR:

Streaming replicas—and by extension, base backups—can become dangerously broken when the source and target machines run slightly different versions of glibc. Particularly, differences in `strcoll` and `strcoll_l` leave "corrupt" indexes on the slave. These indexes are sorted out of order with respect to the `strcoll` running on the slave. Because postgres is unaware of the discrepancy is uses these "corrupt" indexes to perform merge joins; merges rely heavily on the assumption that the indexes are sorted and this causes all the results of the join past the first poison pill entry to not be returned. Additionally, if the slave becomes master, the "corrupt" indexes will in cases be unable to enforce uniqueness, but quietly allow duplicate values.

Context:

We were doing a hardware upgrade on a large internal machine a couple months ago. We followed a common procedure here: stand up a the new HA pair as streaming replica's of the old system; then failover to the new pair. All systems involved were running 9.1.9 (though that is not relevant as we'll see), and built from source.

Immediately, after the failover we saw some weird cases with some small indexes. We thought it was because the streaming replication failover had gone poorly (and because we weren't

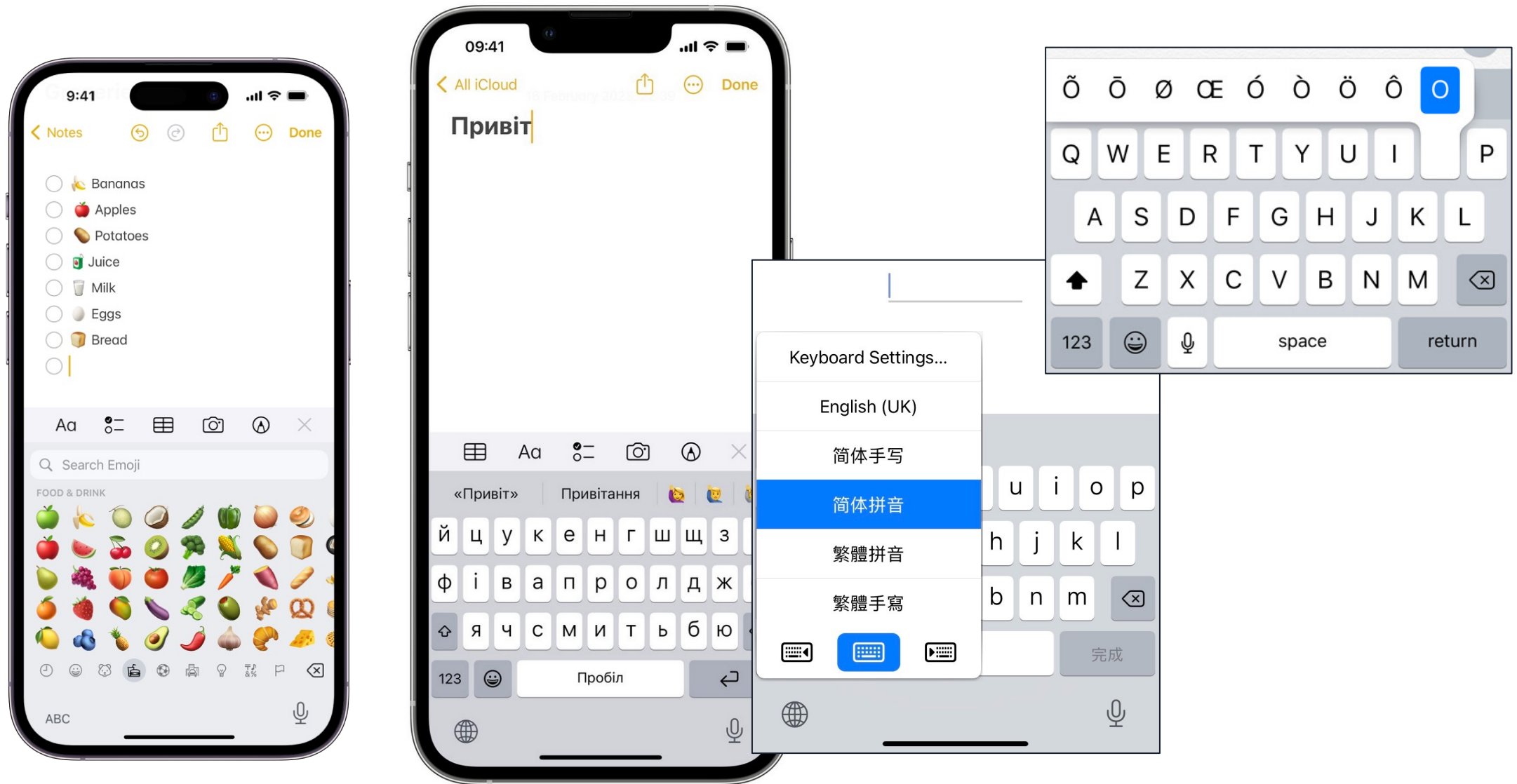
2014

INCORRECT

7. My database doesn't have any characters from that uncommon language with a sort order change

I can safely update the collation library and ignore warnings about corruption

Assume Unexpected Characters



INCORRECT

8. My database understands all of the characters that are in it

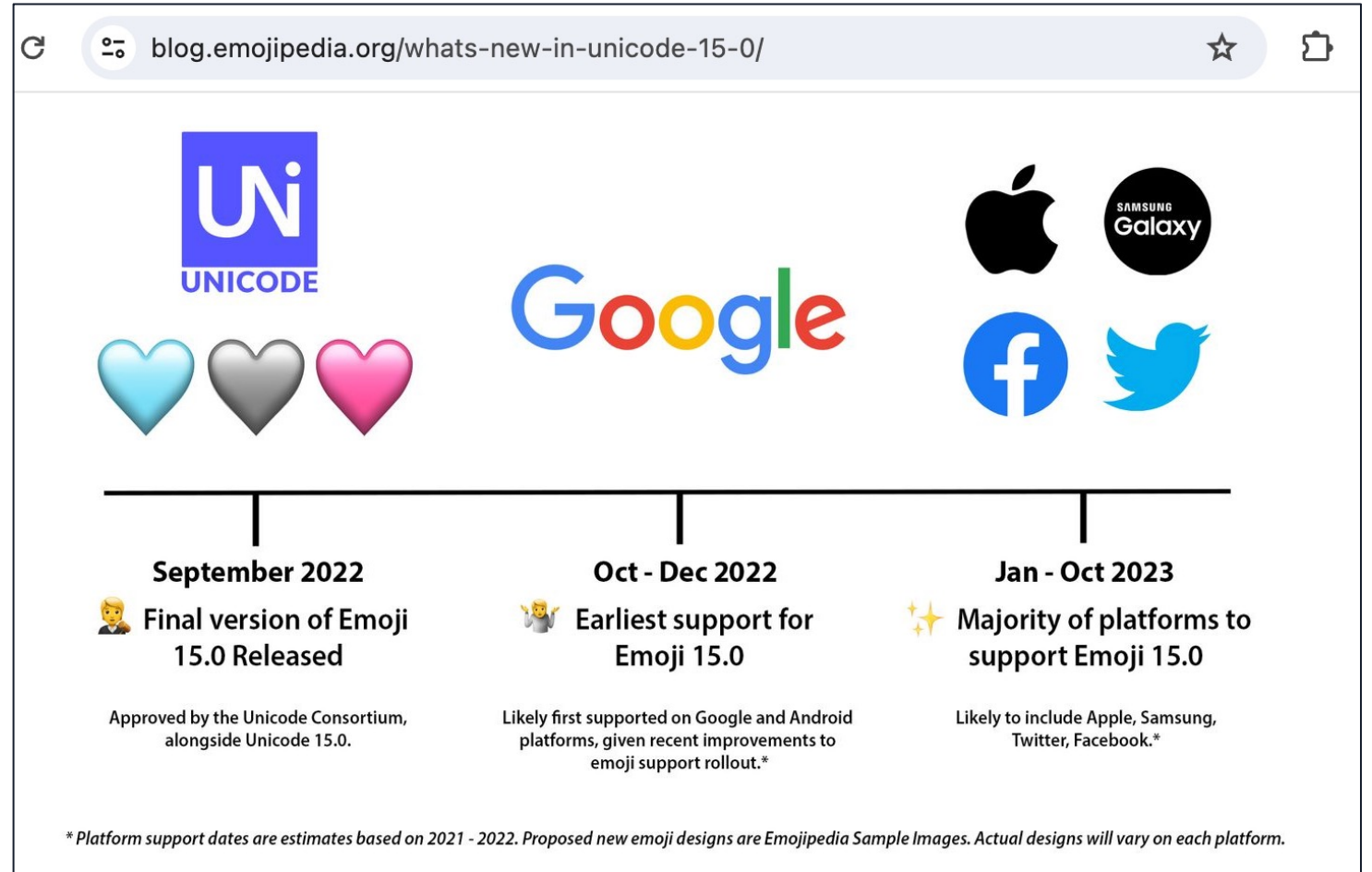
Device and App Updates

New versions of Unicode are deployed quickly to devices and end users

Generally less than a year

A database that rejects unknown code points will not store data entered on current phones & apps, if the data includes new characters

Patches are on the mailing lists, still under discussion



INCORRECT

9. The Postgres warning message about “wrong collation library version” will be displayed to someone

Database Navigator

Enter a part of object name here

- research_texts - 54.235.41.254:5432
- research_texts_hotstandby - 174.129.177.64:5432
 - Databases
 - research_texts
 - Schemas
 - public
 - Tables
 - arabic_dictionary_research
 - Views
 - Materialized Views

Project - General

Name | DataS

- Bookmarks
- Diagrams
- Scripts

```

select count(*) from arabic_dictionary_research where word between '1گ' and '9گ';
select count(*) from arabic_dictionary_research where word between '1و' and '9و';
    
```

Results 1 Results 1 (2) X

select count(*) from ar: | Data filter is not supported

Grid	123 count
1	0

Value X
0

Refresh Save Cancel Export data 200

1 row(s) fetched

“Warning” May Appear in Server Logs Only

<https://ardentperf.com/2023/03/26/did-postgres-lose-my-data/>

And while no messages were ever actively displayed to either the admin who created the hot standby or the researcher who was running SQL in DBeaver, there was a warning message buried in the database log on the hot standby server:

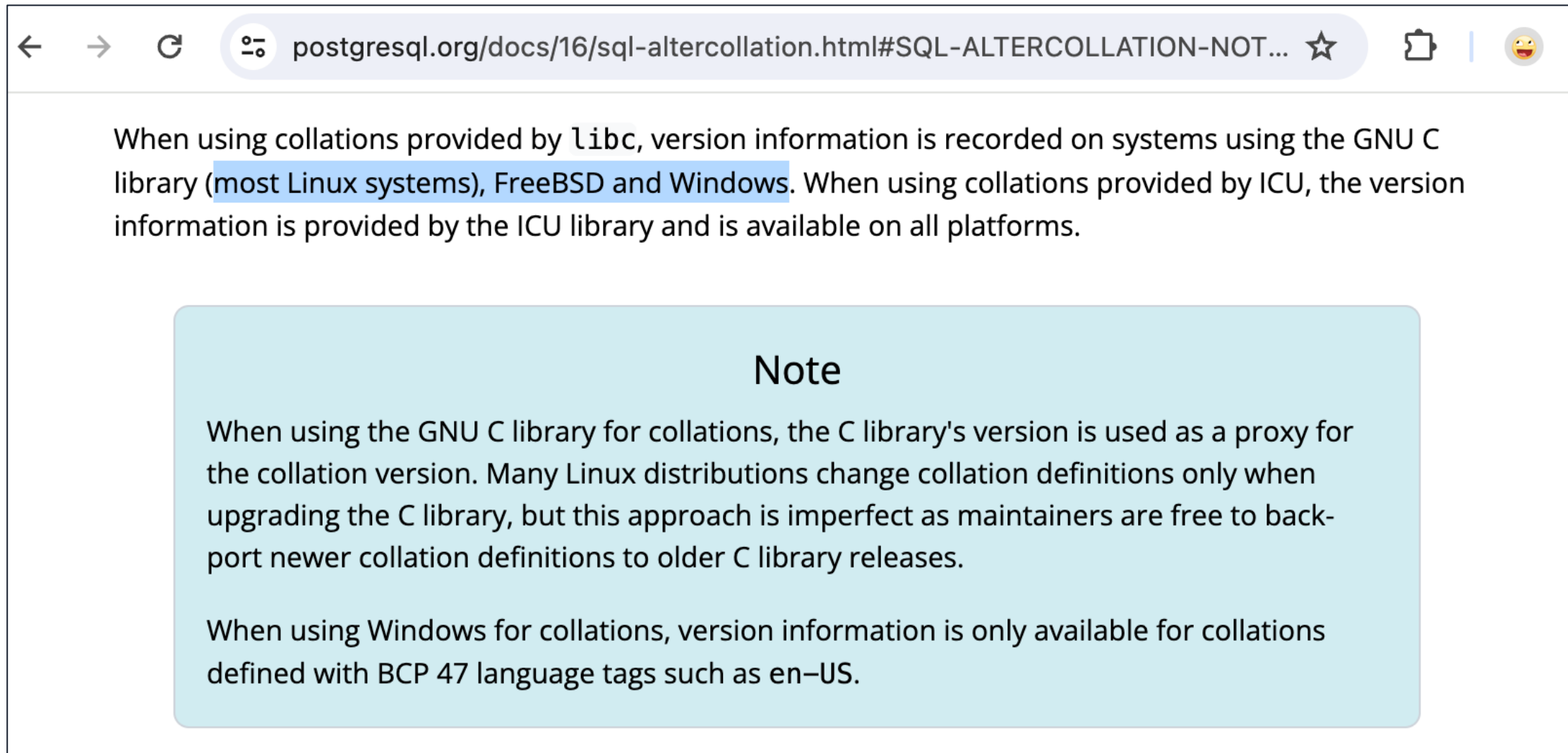
```
ubuntu@ip-10-0-0-117:~$ tail /var/log/postgresql/postgresql-15-main.log
2023-03-26 07:39:47.656 UTC [5053] LOG: restartpoint complete: wrote 71 buffers (0.4%); 0 WAL file(s) added, 0 removed, 0 recycled; write=7.026
s, sync=0.004 s, total=7.039 s; sync files=51, longest=0.003 s, average=0.001 s; distance=266 kB, estimate=14772 kB
2023-03-26 07:39:47.656 UTC [5053] LOG: recovery restart point at 0/3042B20
2023-03-26 07:39:47.656 UTC [5053] DETAIL: Last completed transaction was at log time 2023-03-26 07:36:32.138932+00.
2023-03-26 07:44:55.770 UTC [5053] LOG: restartpoint starting: time
2023-03-26 07:45:09.811 UTC [5053] LOG: restartpoint complete: wrote 141 buffers (0.9%); 0 WAL file(s) added, 0 removed, 0 recycled;
write=14.031 s, sync=0.003 s, total=14.042 s; sync files=22, longest=0.002 s, average=0.001 s; distance=1309 kB, estimate=13425 kB
2023-03-26 07:45:09.811 UTC [5053] LOG: recovery restart point at 0/3189F90
2023-03-26 07:45:09.811 UTC [5053] DETAIL: Last completed transaction was at log time 2023-03-26 07:41:50.782267+00.
2023-03-26 09:20:06.353 UTC [5498] ubuntu@research_texts WARNING: database "research_texts" has a collation version mismatch
2023-03-26 09:20:06.353 UTC [5498] ubuntu@research_texts DETAIL: The database was created using collation version 153.14, but the operating
system provides version 153.112.
2023-03-26 09:20:06.353 UTC [5498] ubuntu@research_texts HINT: Rebuild all objects in this database that use the default collation and run
ALTER DATABASE research_texts REFRESH COLLATION VERSION, or build PostgreSQL with the right library version.
```

Collation.

INCORRECT

10. Postgres can always know what version of C Libraries are installed on the OS

Postgres Detects Version On Common OS's



When using collations provided by `libc`, version information is recorded on systems using the GNU C library (most Linux systems), FreeBSD and Windows. When using collations provided by ICU, the version information is provided by the ICU library and is available on all platforms.

Note

When using the GNU C library for collations, the C library's version is used as a proxy for the collation version. Many Linux distributions change collation definitions only when upgrading the C library, but this approach is imperfect as maintainers are free to back-port newer collation definitions to older C library releases.

When using Windows for collations, version information is only available for collations defined with BCP 47 language tags such as `en-US`.

INCORRECT

11. You can't just

“extract the collation code from an old glibc (GNU C Library) version, build it as an independent library, and install it on a new major OS release”

← → ↻ 🔍 github.com/awslabs/compat-collation-for-glibc/ 📄 ☆ 🗑️




README Code of conduct License Security ☰

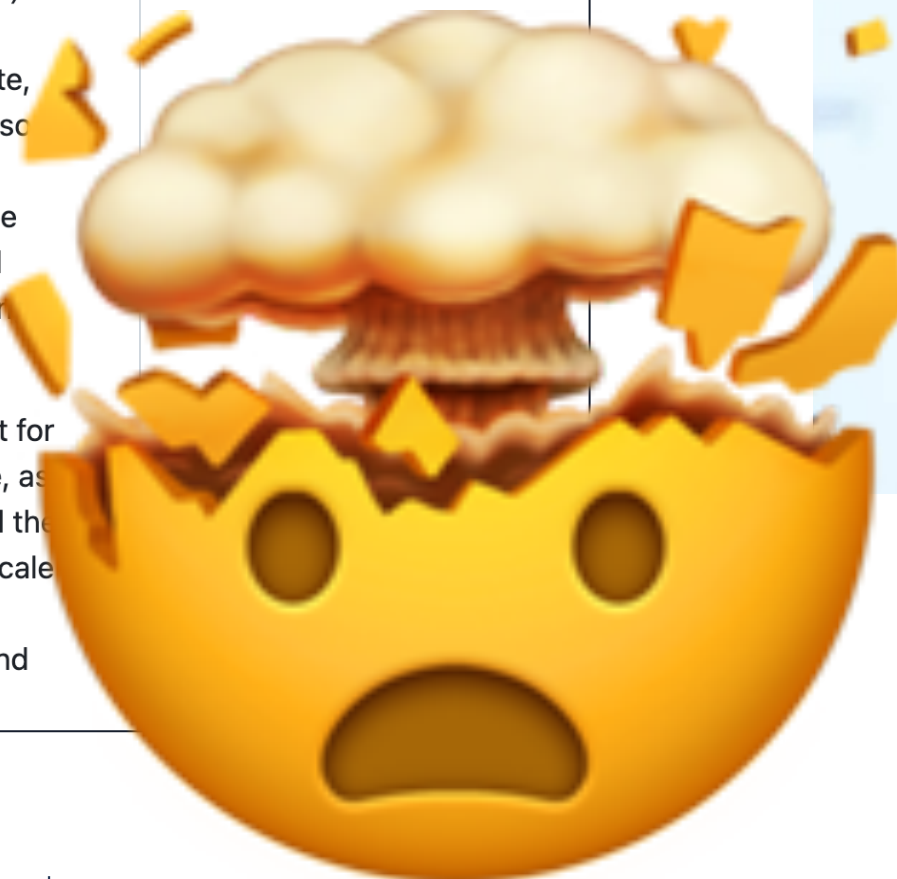
Overview

glibc is the GNU C Library implementation, which is used on all major Linux distributions (e.g. CentOS/AlmaLinux/Rocky, Debian/Ubuntu, SuSE). The glibc library, libc.so, provides most of the foundational C routines such as open, read, write, malloc, printf, and literally thousands more. It also provides the interface to the Linux kernel via syscalls. For the purposes of this discussion, the facility of interest is the locale functionality, and more specifically the functions that provide string sorting according to localized collation rules.

Locale specific sorting is important and relevant for programs such as PostgreSQL. That is because, as a database, PostgreSQL must frequently sort and then persist string data according to the specified locale collation. In order for this to work durably and correctly, the sort order must be determinant and immutable.

Contributors 3

-  **sharmay** Yogesh Sharma
-  **jconway** Joe Conway
-  **amazon-auto** Amazon Git



CONFERENCE SCHEDULE - PGCON 2023

[Back](#)

SORTING OUT GLIBC COLLATION CHALLENGES

Date: 2023-05-31

Time: 10:00-10:45

Room: DMS 1140

Level: Intermediate

Background: "libc" is commonly used as a shorthand for the "standard C library", a library of standard functions that can be used by all C programs. glibc is the GNU C Library implementation, which is used on Linux. The glibc library, libc.so, provides most of the foundational C routines such as malloc, printf, and fopen. glibc provides the interface to the Linux kernel via syscalls.

For the purposes of this talk, the facility of interest is the locale function, which returns the locale according to localized collation rules. In order for PostgreSQL to work with a locale, it must use the glibc library. Since glibc implements the sort order, if/when glibc changes the sort order, PostgreSQL, and thereby causes data corruption. Indexes that have been created according to the currently installed version of glibc.

Proposed Solution: A solution, outlined in this talk, demonstrates a method to create a specific glibc base-version. That may then be used on another Linux system and/or OS upgrades.

Summary: If a PostgreSQL database resides on, for example, a RHEL 7 system and is upgraded to RHEL 8 with glibc version 2.28, the majority of indexes built on the RHEL 7 system will have examples of the types of breakage that can occur, the proposed solution

SPEAKER

[Joe Conway](#)



Collation Challenges

Sorting It Out

Joe Conway
conway@amazon.com
mail@joeconway.com

AWS
May 31, 2023



INCORRECT

12. ICU solves everything

Ubuntu - ICU

ICU Version	Operating System	Total en-US	Unicode Blocks en-US	Total ja-JP	Unicode Blocks ja-JP	Total zh-Hans-CN	Unicode Blocks zh-Hans-CN	Total ru-RU	U I
52.1-3ubuntu0.8	Ubuntu 14.04.6 LTS								
55.1-7ubuntu0.5	Ubuntu 16.04.7 LTS	(324 blocks)	286654 (Full Diff)	(324 blocks)	286654 (Full Diff)	(324 blocks)	286654 (Full Diff)	(324 blocks)	2 (Full Diff)
60.2-3ubuntu3.1	Ubuntu 18.04.6 LTS	(66 blocks)	23741 (Full Diff)	(66 blocks)	23741 (Full Diff)	(68 blocks)	24415 (Full Diff)	(66 blocks)	2 (Full Diff)
63.1-6	Ubuntu 19.04	(41 blocks)	688 (Full Diff)	(41 blocks)	688 (Full Diff)	(41 blocks)	688 (Full Diff)	(41 blocks)	6 (Full Diff)
66.1-2ubuntu2	Ubuntu 20.04.3 LTS	(57 blocks)	6497 (Full Diff)	(58 blocks)	6501 (Full Diff)	(56 blocks)	6513 (Full Diff)	(57 blocks)	6 (Full Diff)
67.1-4	Ubuntu 20.10	0	0	0	0	0	0	0	0
67.1-6ubuntu2	Ubuntu 21.04	0	0	0	0	0	0	0	0
67.1-7ubuntu1	Ubuntu 21.10	0	0	0	0	0	0	0	0
70.1-2	Ubuntu 22.04 LTS	(47 blocks)	879 (Full Diff)	(47 blocks)	875 (Full Diff)	(48 blocks)	887 (Full Diff)	(47 blocks)	8 (Full Diff)
71.1-3ubuntu1	Ubuntu 22.10	0	0	0	0	0	0	0	0

ICU is a far better choice than the operating system C library

But it doesn't solve everything

Every single Ubuntu LTS in the last 8 years has ICU sort order changes



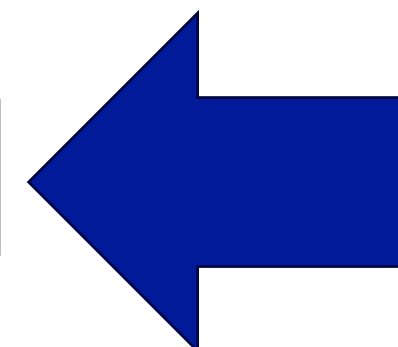
INCORRECT

13. ICU never had a huge sort order change like the glibc 2.28 fiasco

Ubuntu - ICU

ICU Version	Operating System	Total en-US	Unicode Blocks en-US	Total ja-JP	Unicode Blocks ja-JP	Total zh-Hans-CN	Unicode Blocks zh-Hans-CN	Total ru-RU	
52.1-3ubuntu0.8	Ubuntu 14.04.6 LTS								
55.1-7ubuntu0.5	Ubuntu 16.04.7 LTS	(324 blocks)	286654 (Full Diff)	(324 blocks)	286654 (Full Diff)	(324 blocks)	286654 (Full Diff)	(324 blocks)	
60.2-3ubuntu3.1	Ubuntu 18.04.6 LTS	(66 blocks)	23741 (Full Diff)	(66 blocks)	23741 (Full Diff)	(68 blocks)	24415 (Full Diff)	(66 blocks)	
63.1-6	Ubuntu	(41 blocks)	688 (Full Diff)	(41 blocks)	688 (Full Diff)	(41 blocks)	688 (Full Diff)	(41 blocks)	
		(57 blocks)	6497 (Full Diff)	(58 blocks)	6501 (Full Diff)	(56 blocks)	6513 (Full Diff)	(57 blocks)	
		0	0	0	0	0	0	0	
		0	0	0	0	0	0	0	
67.1-7ubuntu1	Ubuntu 21.10	0	0	0	0	0	0	0	
70.1-2	Ubuntu 22.04 LTS	(47 blocks)	879 (Full Diff)	(47 blocks)	875 (Full Diff)	(48 blocks)	887 (Full Diff)	(47 blocks)	
71.1-3ubuntu1	Ubuntu 22.10	0	0	0	0	0	0	0	

Every single code point (the total count in Unicode 15 is 286,654) had at least one string changing sort order between ICU 52 and ICU 55



A “diff” between 26 million sorted strings from ICU 67.1 (Ubuntu 21.10) and ICU 70.1 (Ubuntu 22.04) using the locale “en-US” reported 879 distinct characters in patterns that moved to a different location. Those characters were spread over 47 Unicode Blocks.

Click “879” for a complete list of all strings that “diff” says changed position. There are more than 879, since many code points had multiple strings change position. Click “Full Diff” to see the raw output of the diff command.

Click here for a summary of which string patterns and how many distinct code points appear in each of the 47 impacted unicode blocks



Collation Torture Test Summary

- Both glibc and ICU have regular collation changes.
- Both had at least one release with very large numbers of changes.
- **PL/pgSQL code is published on github to generate a table with the 26 million strings in the “collation torture test”**
- **Can checksum the sorted list to create a test and detect changes**

<https://github.com/ardentperf/glibc-unicode-sorting/blob/main/run-icu.sh#L65>

INCORRECT

14. Assume Devrim and Christoph are happy to build old ICU versions for you



*Unclear if we want this?
Join the mailing lists and let's discuss!*

New contributors always welcome!

INCORRECT

15. Sort order doesn't change in library updates with just patch version changes

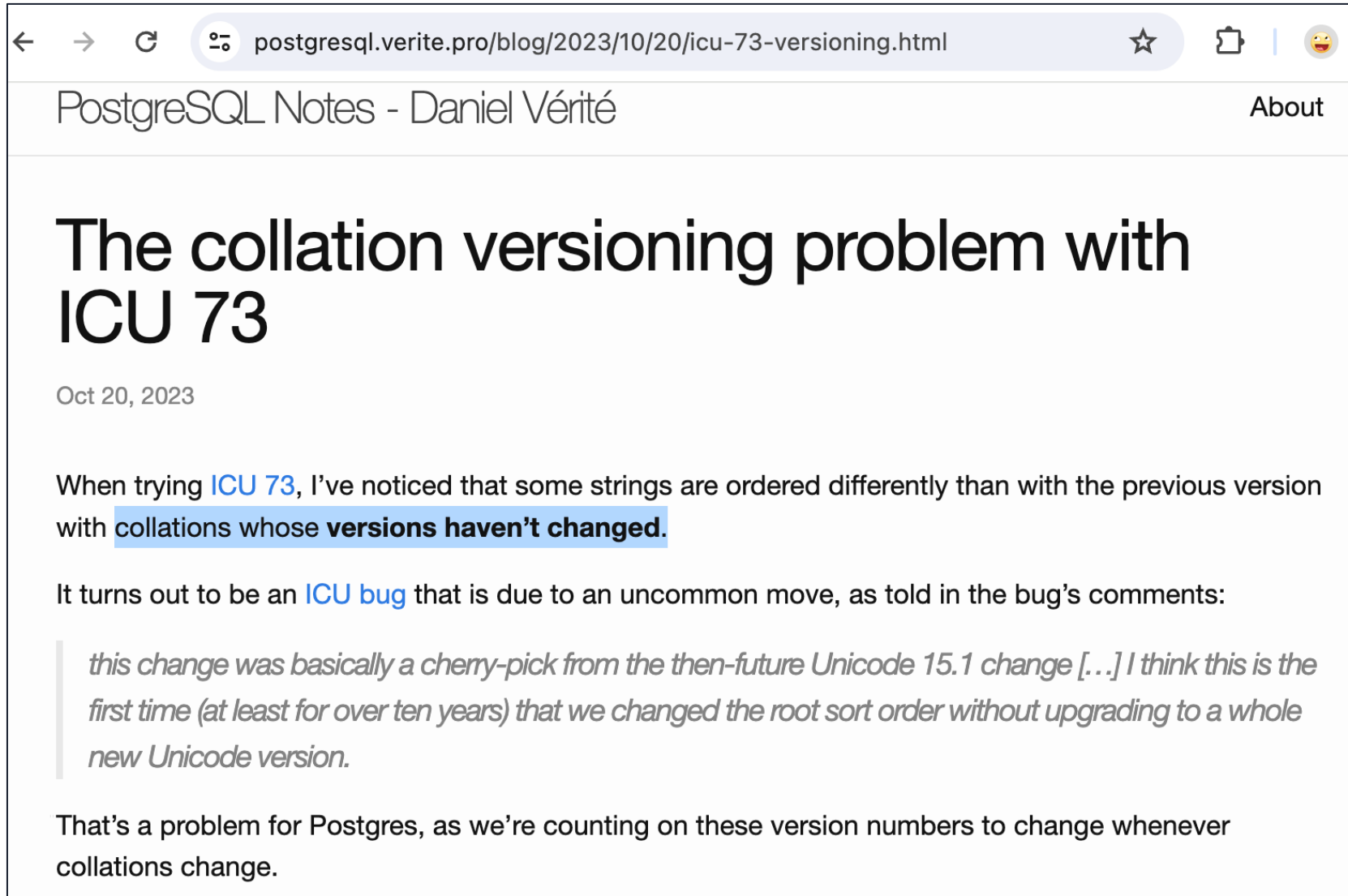


`glibc 2.26-59.amzn2`

INCORRECT

16. Sort order doesn't change in library updates with NO version changes

When It Changed With No Version Bump



The screenshot shows a web browser window with the address bar containing the URL `postgresql.verite.pro/blog/2023/10/20/icu-73-versioning.html`. The page title is "PostgreSQL Notes - Daniel Vérité" and there is an "About" link in the top right. The main heading of the article is "The collation versioning problem with ICU 73", dated "Oct 20, 2023". The text of the article states: "When trying ICU 73, I've noticed that some strings are ordered differently than with the previous version with collations whose versions haven't changed." It then explains that this is an ICU bug due to an uncommon move, and includes a quote: "this change was basically a cherry-pick from the then-future Unicode 15.1 change [...] I think this is the first time (at least for over ten years) that we changed the root sort order without upgrading to a whole new Unicode version." The article concludes that this is a problem for Postgres because it relies on version numbers to indicate when collations change.

INCORRECT

17. Postgres doesn't yet have builtin collation that avoids all corruption risks

POSIX locale – also known as C locale

The screenshot shows a web browser window displaying the Open Group Base Specifications Issue 7, 2018 edition website. The URL is pubs.opengroup.org/onlinepubs/9699919799/. The page is titled "7. Locale" and is part of the "Base Definitions" section. The main content describes the POSIX locale, also known as the C locale, and lists several environment variables: `LC_CTYPE`, `LC_COLLATE`, `LC_MONETARY`, and `LC_NUMERIC`. The page also includes a navigation menu with links for "Previous", "Home", and "Next", and a search bar. The footer of the page contains the copyright information: "© 2001-2018 IEEE and The Open Group".

INDEX

Search..

[[Alphabetic](#) | [Topic](#) | [Word Search](#)]

Select a Volume:
[[Base Definitions](#) | [System Interfaces](#) | [Shell & Utilities](#) | [Rationale](#)]
[[Frontmatter](#)]
[[Main Index](#)]

Base Definitions

- [1. Introduction](#)
- [2. Conformance](#)
- [3. Definitions](#)
- [4. General Concepts](#)
- [5. File Format Notation](#)
- [6. Character Set](#)
- [7. Locale](#)
- [8. Environment Variables](#)
- [9. Regular Expressions](#)
- [10. Directory Structure and Devices](#)
- [11. General Terminal Interface](#)
- [12. Utility Conventions](#)
- [13. Headers](#)

<<< [Previous](#) [Home](#) [Next](#) >>>

The Open Group Base Specifications Issue 7, 2018 edition
IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)
Copyright © 2001-2018 IEEE and The Open Group

7. Locale

7.1 General

A locale is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

`LC_CTYPE`
Character classification and case conversion.

`LC_COLLATE`
Collation order.

`LC_MONETARY`
Monetary formatting.

`LC_NUMERIC`
Numeric, non-monetary formatting.

7.2 POSIX Locale

Conforming systems shall provide a POSIX locale, also known as the C locale. In POSIX.1 the requirements for the POSIX locale are more extensive than the requirements for the C locale as specified in the ISO C standard. However, in a conforming POSIX implementation, the POSIX locale and the C locale are identical. The behavior of standard utilities and functions in the POSIX locale shall be as if the locale was defined via the `localedef` utility with input data from the POSIX locale tables in [Locale Definition](#).

For C-language programs, the POSIX locale shall be the default locale when the `setlocale()` function is not called.

The POSIX locale can be specified by assigning to the appropriate environment variables the values "C" or "POSIX".

All implementations shall define a locale as the default locale, to be invoked when no environment variables are set, or set to the empty string. This default locale can be the POSIX locale or any other implementation-defined locale. Some implementations may provide facilities for local installation administrators to set the default locale, customizing it for each location. POSIX.1-2017 does not require such a facility.

INCORRECT

18. Postgres **C** and **C.UTF-8** are the same

<u>libc provider</u> C collation	<u>libc provider</u> C.UTF-8 collation
implemented internally; does not call libc (the PG provider name of “libc” is misleading)	calls libc

INCORRECT

19. Sort order doesn't change in `C.UTF-8`

Sort Order Changed in glibc C.UTF-8

From: "Daniel Verite" <daniel(at)manitou-mail(dot)org>
To: pgsql-hackers(at)postgresql(dot)org
Subject: pg_collation.collversion for C.UTF-8
Date: 2023-04-18 12:35:50
Message-ID: 8a3dc06f-9b9d-4ed7-9a12-2070d8b0165f@manitou-mail.org
Views: [Raw Message](#) | [Whole Thread](#) | [Download mbox](#) | [Resend email](#)
Lists: [pgsql-hackers](#)

Hi,

get_collation_actual_version() in pg_locale.c currently excludes C.UTF-8 (and more generally C.*) from versioning, which makes pg_collation.collversion being empty for these collations.

```
char *
get_collation_actual_version(char collprovider, const char *collcollate)
{
    ...
    if (collprovider == COLLPROVIDER_LIBC &&
        pg_strcasecmp("C", collcollate) != 0 &&
        pg_strncasecmp("C.", collcollate, 2) != 0 &&
        pg_strcasecmp("POSIX", collcollate) != 0)

```

This seems to be based on the idea that C.* collations provide an immutable sort like "C", but it appears that it's not the case.

For instance, consider how these C.UTF-8 comparisons differ between recent linux systems:

U+1D400 = Mathematical Bold Capital A

```
Debian 9.13 (glibc 2.24)
=> select 'A' < E'\U0001D400' collate "C.UTF-8";
?column?
-----
t
```

```
Debian 10.13 (glibc 2.28)
=> select 'A' < E'\U0001D400' collate "C.UTF-8";
?column?
-----
f
```

```
Debian 11.6 (glibc 2.31)
=> select 'A' < E'\U0001D400' collate "C.UTF-8";
?column?
-----
f
```

```
Ubuntu 22.04 (glibc 2.35)
=> select 'A' < E'\U0001D400' collate "C.UTF-8";
?column?
-----
t
```

sourceware.org/glibc/wiki/Proposals/C.UTF-8

glibc wiki Login

Self: [Proposals/ C.UTF-8](#)

Home Page Recent Changes Find Page Help Contents **Proposals/C.UTF-8**

Immutable Page Info Attachments More Actions: ▾

C.UTF-8 locale

2015

Contents

- Status
- Problem Statement
- Proposal
 - Builtin
 - Defaults
- Other Art
 - POSIX
 - Debian
 - Fedora/RedHat
 - OS X
 - References

1. Status

◆ Merged for glibc 2.35

2. Problem Statement

Modern systems need a modern encoding system to deal with global data. The old customs data as [ASCII](#) (or [ISO 8859-1](#)) is long past and has no business in the 21st century. Per hitting [mojibake](#) today is deplorable.

However, there is no way today to select UTF-8 encoding without also picking a country/lang locale. Many projects hardcode en_US.UTF-8, or maybe try one or two more (like en_GB.UTF-8 de_DE.UTF-8), before giving up and failing. This is also why distros often do not select a UTF-8 by default since the related locale attributes are undesirable.

Python blazed an admirable trail here by putting encoding front and center with its 3.x series runs into a problem where it has to guess as to the encoding of stdin/stdout/stderr. By makin available, this can be handled gracefully.

3. Proposal

The world has largely settled on the [Unicode standard](#) with [UTF-8](#) as the leading encoding. Hence we will provide an amalgamation of POSIX's C locale with UTF-8 encoding.

The new locale name shall be C.UTF-8. It shall be the C locale but with UTF-8 encodings.

Setting LC_ALL=C.UTF-8 will ignore LANGUAGE just like it does with LC_ALL=C. See guess_category_value()

git://sourceware.org / glibc.git / commit

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#) | [commit](#) ▾ | ? search:

(parent: [f5117c6](#)) | [patch](#)

Add generic C.UTF-8 locale (Bug 17318)

2021

author	Carlos O'Donnell <carlos@redhat.com>	Wed, 1 Sep 2021 19:19:19 +0000 (15:19 -0400)
committer	Carlos O'Donnell <carlos@redhat.com>	Mon, 6 Sep 2021 15:30:28 +0000 (11:30 -0400)
commit	466f2be6c08070e9113ae2fdc7acd5d8828cba50	
tree	c4fb7c10d98994298dcd451df71f1be790b575e9	tree
parent	f5117c6504888fab5423282a4607c552b90fd3f9	commit diff

Add generic C.UTF-8 locale (Bug 17318)

We add a new C.UTF-8 locale. This locale is not builtin to glibc, but is provided as a distinct locale. The locale provides full support for UTF-8 and this includes full code point sorting via STRCMP-based collation (strcmp or wcsmp).

The collation uses a new keyword 'codepoint_collation' which drops all collation rules and generates an empty zero rules collation to enable STRCMP usage in collation. This ensures that we get full code point sorting for C.UTF-8 with a minimal 1406 bytes of overhead (LC_COLLATE structure information and ASCII collating tables).

The new locale is added to SUPPORTED. Minimal test data for specific code points (minus those not supported by collate-test) is provided in C.UTF-8.in, and this verifies code point sorting is working reasonably across the range. The locale was tested manually with the full set of code points without failure.

The locale is harmonized with locales already shipping in various downstream distributions. A new tst-iconv9 test is added which verifies the C.UTF-8 locale is generally usable.

Testing for fnmatch, regex, and recompile is provided by extending bug-regex1, bug-regex19, bug-regex4, bug-regex6, transbug, tst-fnmatch, tst-regex-truncated, and tst-regex to use C.UTF-8.

Tested on x86_64 or i686 without regression.

Reviewed-by: Florian Weimer <fweimer@redhat.com>

Sort Order Changed in glibc C.UTF-8

<u>libc_provider</u> C collation	<u>libc_provider</u> C.UTF-8 collation
implemented internally; does not call libc (the PG provider name of “libc” is misleading)	calls libc
stable & safe; does not change	changes should be uncommon (less than icu and libc linguistic locales), but history shows that both character semantics and sort order have not remained unchanged for example in Debian/Ubuntu (cf. mailing list thread)

INCORRECT

20. Collation provider is only for sort order

Postgres "C" Locale Only Understands ASCII

- ✓ CTYPE = upper, lower, initcap, regex character classes, etc

```
-- show the inability of "C" to uppercase accented characters
test=> select initcap('élysée' collate "C");
initcap
-----
éLysée
```

Accented characters not uppercased correctly
Thinks accented character is not a letter

```
-- show the ability of "C.utf8" to uppercase accented characters
test=> select initcap('élysée' collate "C.utf8");
initcap
-----
Élysée
```

<https://postgresql.verite.pro/blog/2024/03/13/binary-sorted-indexes.html>

INCORRECT

21. CTYPE doesn't change in C.UTF-8

Upper,etc can change too

From: Thomas Munro <thomas(dot)munro(at)gmail(dot)com>
To: Jeff Davis <pgsql(at)j-davis(dot)com>
Cc: Daniel Verite <daniel(at)manitou-mail(dot)org>, pgsql-hackers(at)postgresql(dot)org
Subject: Re: pg_collation.collversion for C.UTF-8
Date: 2023-06-17 05:54:35
Message-ID: CA+hUKGKr-b33uw_3nUEa80aft0RKy0D+oo41ztRLyuby4oQX8g@mail.gmail.com
Views: [Raw Message](#) | [Whole Thread](#) | [Download mbox](#) | [Resend email](#)
Lists: [pgsql-hackers](#)

On Sat, Jun 17, 2023 at 10:03 AM Jeff Davis <pgsql(at)j-davis(dot)com> wrote:
> I assume you mean that the collation order can't (shouldn't, anyway)
> change. But what about the ctype (upper/lower/initcap) behavior? Is
> that also locked down for all time, or could it change if some new
> unicode characters are added?

Fair point. Considering that our collversion effectively functions as a proxy for ctype version too, Daniel's patch makes a certain amount of sense.

Our versioning is nominally based only on the collation category, not locales more generally or any other category they contain (nominally, as in: we named it collversion, and our code and comments and discussions so far only contemplated collations in this context). But, clearly, changes to underlying ctype data could also cause a constraint CHECK (x ~ '[:digit:]') or a partial index with WHERE (upper(x) <> 'ß') to be corrupted, which I'd considered to be a separate topic, but Daniel's patch would cover with the same

INCORRECT

22. Users want DB-wide linguistic sort

No widely used major database today would default to code-point or binary sort order

Code Point Order as Database Default

<https://ardentperf.com/2024/05/22/default-sort-order-in-db2-sql-server-oracle-postgres-17/>

	Default Collation	Server/Client	System Catalogs	UCA Support
Oracle	Code Point Order ‡ (called BINARY)	Property of connection/client, can change	Always BINARY	Unicode Versions 6.1 / 6.2 / 7.0 / 12.1 <i>builtin</i>
Db2	Code Point Order (called IDENTITY)	Property of database/server, cannot change	Always IDENTITY for Unicode DBs	Unicode Versions 4.0 / 5.0 / 5.2 / 7.0 <i>builtin</i>
SQL Server	OS default locale with 8-bit encoding	Property of database/server, can change DB default for new objects, cannot server/catalogs	Server collation	Not supported (afaik?)
Postgres	OS default locale with Unicode	Property of database/server, cannot change	Database collation	Unicode Version 4.2+ <i>installed separately</i>

‡ If Oracle client locale is Europe, Middle East, Quebec, or a few other unlucky countries – then the default behavior is that ORDER BY and a few functions like regex sort with client locale, while operators like greater-than, less-than, group-by and indexes still use code-point/BINARY order.

Anecdotally, it seems common to run Oracle with default settings for database-wide collation.

Oracle third-party apps like eBusiness Suite require binary (code-point) collation. Some SQL Server third-party apps also mandate a specific collation, for portability.

Code Point Order as Database Default

postgresql.verite.pro/blog/2024/03/13/binary-sorted-indexes.html

PostgreSQL Notes - Daniel Vérité

Using binary-sorted indexes

Mar 13, 2024

In a [previous post](#), I mentioned that Postgres databases often have text indexes sorted linguistically rather than bitwise, which is why they need to be **reindexed** on libc or ICU upgrades. In this post, let's discuss how to use bitwise sorts, and what are the upsides and downsides of doing so.

Sorting strings in binary means comparing the bytes inside the strings without caring at all about what characters they represent. For instance in an UTF-8 database, when considering the strings `Beta` and `aLpha`:

- a bitwise comparison says that `'Beta' < 'alpha'`, since the code point of the upper-case letter `B` is `0x42` and the code point of the lower-case letter `a` is `0x61`.
- a linguistic comparison says that `'alpha' < 'Beta'` because it understands that the letter `a` comes before `B` even when cases are mixed. More generally linguistic collations have sorting rules concerning accents, punctuation, symbols, plus potentially regional tailorings.

A brief pros and cons comparison of these sorts could look like this:

	Linguistic order	Binary order
Ease of use	✔ better	✘ worse
Human readability	✔ better	✘ worse
Range search (*)	✔ better	✘ worse
Performance	✘ worse	✔ better
Portability	✘ worse	✔ 100%
Real immutability	✘ No	✔ Yes
LIKE prefix search	✘ No	✔ Yes

(*) Locating strings between two bounds, for instance to output paginated results

Ongoing discussion: making a case for binary at DB level?

Home My Network Jobs Messaging Notifications

Jobin Augustine · 1st
Passionate about PostgreSQL
11h · 🌐

After dealing with a large set of troubles users are getting into due to character collations rules (Index corruptions/upgrade troubles, Wrong query results, etc.) I am sure that the majority of PostgreSQL users are not aware of the character collation-related troubles that await them if the data directory is initialized (initdb) with all system defaults, which takes the host machine's localizations. My suggestion? Stick with binary collation on the server side unless you have a compelling reason to do otherwise.


ardentperf.com/2024/05/22/default-sort-order-in-db2-sql-server-oracle-postgres-17/

Default Sort Order in Db2, SQL Server, Oracle & Postgres 17

POSTED BY JEREMY · MAY 22, 2024 · LEAVE A COMMENT

FILED UNDER COLLATION, COMPARISON, DATABASE, DB2, ORACLE, POSTGRES, SORT, SQL, SQLSERVER

TLDR: I was starting to think that the best choice of default DB collation (for sort order, comparison, etc) in Postgres might be ICU. But after spending some time reviewing the landscape, **I now think that code-point order is the best default DB collation – mirroring Db2 and Oracle – and linguistic sorting can be used via SQL when it's actually needed for the application logic.** In existing versions of Postgres, this would be something like C or C.UTF-8 and Postgres 17 will add the builtin collation provider (more details at the bottom of this article). This ensures that the system catalogs always use code-point collation, and it is a similar conclusion to what Daniel Vérité seems to propose in his [March 13 blog](#), "Using binary-sorted indexes". I like the suggestion he closed his blog with: `SELECT ... FROM ... ORDER BY colname COLLATE "unicode"` – when you need natural language sort order.



INCORRECT

23. Postgres isn't likely to get a new builtin collation solving these problems

Usable character semantics and no corruption risks

23. PostgreSQL builtin c Usable cha

→ ↻ 🏠 postgresql.org/about/news/postgresql-17-beta-1-released-2865/



PostgreSQL 17 Beta 1 Released!

Posted on **2024-05-23** by PostgreSQL Global Development Group

📁 PostgreSQL Project

The PostgreSQL Global Development Group announces that the first beta release of PostgreSQL 17 is now **available for download**. This release contains previews of all features that will be available when PostgreSQL 17 is made generally available, though some details of the release can change during the beta period.

You can find information about all of the PostgreSQL 17 features and changes in the **release notes**:

<https://www.postgresql.org/docs/17/release-17.html>

In the spirit of the open source PostgreSQL community, we strongly encourage you to test the new features of PostgreSQL 17 on your systems to help us eliminate bugs or other issues that may exist. While we do not advise you to run PostgreSQL 17 Beta 1 in production environments, we encourage you to find ways to run your typical application workloads against this beta release.

Your testing and feedback will help the community ensure that the PostgreSQL 17 release upholds our standards of delivering a stable, reliable release of the world's most advanced open source relational database. Please read more about our **beta testing process** and how you can contribute:

<https://www.postgresql.org/developer/beta/>

Taking a Step Back

IS COLLATION TOO COMPLICATED?

- Should we ignore the complexity?
- Handle it in the application?
- Let's start with what would be missing, and fix one problem at a time

Locale “C”

THE NON-LOCALE

- Binary string comparison
- Character semantics are only defined for ASCII characters
- Problems:
 - Sort order is encoding-dependent
 - LOWER() and UPPER() don't handle accented characters
 - 'Z' sorts before 'a' (and other unnatural sort orders)
 - No case-insensitive sorting

Locale “C.UTF-8”

IMPROVED IN VERSION 17 WITH BUILTIN PROVIDER

- Code point sort order
- Character semantics are based on Unicode
- Problems:
 - ‘Z’ sorts before ‘a’ (and other unnatural sort orders)
 - No case-insensitive sorting
 - Caveats when using libc provider
 - solved with builtin provider in 17

ICU Root (“und”) Collation

NATURAL LANGUAGE SEMANTICS NOT TIED TO A SPECIFIC LOCALE

- Natural language sort order based on Unicode and CLDR
- Provides reasonable semantics in a variety of locales
 - Solves 'Z' < 'a' problem
- Problems:
 - Collation changes over time can cause inconsistent indexes
 - Must carefully manage library versions
 - Slower performance than code point order or binary order
 - Not specific to any locale, so will produce surprising results for some languages or regions

Initdb

SELECT DATABASE DEFAULT COLLATION

```
# database collation builtin C.UTF-8
```

```
# (version 17+)
```

```
initdb --locale-provider=builtin \  
      --builtin-locale=C.UTF-8 data
```

```
# database collation ICU root collation
```

```
# (version 15+)
```

```
initdb --locale-provider=icu \  
      --icu-locale=und data
```


COLLATE clause

APPLY COLLATIONS TO INDIVIDUAL QUERIES

-- Builtin C.UTF-8

-- (version 17+)

```
SELECT * FROM mytable ORDER BY t COLLATE PG_C_UTF8;
```

-- ICU root collation

-- (requires ICU)

```
SELECT * FROM mytable ORDER BY t COLLATE UNICODE;
```

ICU Tips & Tricks

CREATING SPECIALIZED COLLATIONS

- Case-insensitive
- Specific locale
- Numeric values in strings

ICU Provider – Case Insensitive

ICU CUSTOMIZABILITY

```
CREATE COLLATION case_insensitive(  
    PROVIDER=icu,  
  
    DETERMINISTIC=false,  
    LOCALE= 'und-u-ks-level2'  
  
);  
  
SELECT 'z' = 'Z' COLLATE case_insensitive; -- true
```

ICU Provider – Specific Locale

ICU CUSTOMIZABILITY

```
CREATE COLLATION french_ca(  
    PROVIDER=icu,  
    LOCALE= 'fr-CA'  
);
```

```
SELECT * FROM mytable ORDER BY t COLLATE french_ca;
```

ICU Provider – Numbers

ICU CUSTOMIZABILITY

```
CREATE COLLATION collate_numbers(  
    PROVIDER=icu,  
  
    DETERMINISTIC=false,  
    LOCALE= 'und-u-kn'  
  
);
```

```
SELECT 'id-45' < 'id-123' COLLATE collate_numbers; -- true
```

Future Work

WHAT'S NEXT?

- More standards-compliant UCS_BASIC
- Unicode case folding
- What are the guidelines for using code point vs. natural language sort?
- Should we reject unassigned code points?
- Should we force Unicode normalization?
- Should Postgres take responsibility for managing different versions of collation libraries?

Conclusion

CONSIDER THE TRADE-OFFS

- Code point order collation is fast and stable
- Natural language collations produce superior results for humans
 - But can change and may produce inconsistent indexes
- "C.UTF-8" locale is a balance that offers code point order collation and Unicode character semantics
 - Improved with builtin provider in version 17
- Choose what makes sense
- Use COLLATE clause to control where a collation applies

Thank You!

Jeff Davis
Jeremy Schneider

