credativ

Sorting Out Collations in PostgreSQL

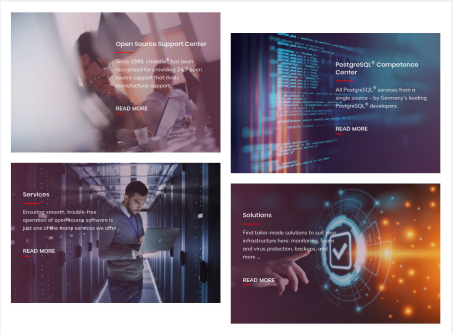
Implementation, Upgrades, Survival Strategies

Josef Machytka <josef.machytka@credativ.de>

2025-11-21 - credativ Tech Talk

credativ GmbH

- Founded 1999 in Jülich, Germany
- Close ties to Open-Source Community
- More than 40 Open-Source experts
- Consulting, development, training, support (3rd-level / 24x7)
- Open-Source infrastructure with Linux, Kubernetes, Proxmox
- Open-Source databases with PostgreSQL
- DevSecOps with Ansible, Puppet, Terraform and others
- Since 2025 independent owner-managed company again



Josef Machytka

credativ

- Professional Service Consultant - PostgreSQL specialist at credativ GmbH
- 33+ years of experience with different databases
- PostgreSQL (13y), BigQuery (7y), Oracle (15y), MySQL (12y), Elasticsearch (5y), MS SQL (5y)
- 10+ years of experience with Data Ingestion pipelines, Data Analysis, Data Lake and Data Warehouse
- 3+ years of practical experience with different LLMs / AI / ML including architecture and principles
- From Czechia, living now 12 years in Berlin

- LinkedIn: linkedin.com/in/josef-machytka
- Medium: medium.com/@josef.machytka
- ResearchGate: researchgate.net/profile/Josef-Machytka
- Academia.edu: academia.edu/JosefMachytka
- sessionize: sessionize.com/josefmachytka

All My Slides:

Existing Talks on Collations

credativ

- There are already plenty of talks on collations and problems with them
  - Collations in PostgreSQL: The good, the bad and the ugly (PGConf.EU 2022)
  - Collation Challenges - Sorting It Out (FOSDEM 2024)
  - Collations from A to Z (PGConf.dev 2024)
  - Language is a Virus (SoCal Linux Expo 2024)
  - Smooth Sailing: How We Tackled PostgreSQL Migration Challenges (PGConf.EU 2024)

- There are also many online articles:
  - Locale Data Changes (PostgreSQL Wiki)
  - Upgrading PostgreSQL on GitLab (GitLab Docs)
  - ICU Collations Against PostgreSQL Data Corruption (Cybertec)
  - Postgres Migration Pitstop: Collations (Crunchy Data)

- In my slides I summarize all important facts from available resources

Terminology

credativ

- **Character set**:
  - Collection of characters that can be represented in a computer system
- **Unicode**:
  - Computing industry standard for consistent encoding, representation, and handling of text
  - Assigns unique code points to characters from most of the world's writing systems
  - Unicode is a character set
- **Locale**:
  - Set of parameters that defines user's language, region, and any special variant preferences
  - Defines conventions for formatting dates, times, numbers, currency symbols, sorting order, etc.

Terminology

- **Collation**:
  - Set of rules for comparing and ordering strings in a specific locale
- **Collation sequence**:
  - The order in which characters are sorted according to collation rules
- **ICU**:
  - International Components for Unicode, a library providing Unicode and globalization support
  - Home page: icu.unicode.org
- **libc**:
  - The standard C library, provides system-level functions including locale and collation support

Encoding vs. Collation

- **Encoding**:
  - Method of converting characters into a specific format for storage or transmission
  - Examples: UTF-8, ASCII, ISO-8859-1, LATIN1

- **Collation**:
  - Set of rules for comparing and ordering characters in a specific locale
  - Examples: en_US.UTF-8, de_DE.UTF-8

Terminology

credativ

- **Common Locale Data Repository (CLDR)**:
  - Repository of locale data maintained by Unicode Consortium
  - Provides standard locale data for software applications
  - Includes information on date/time formats, number formats, collation rules, etc.

- **POSIX locale (ISO/IEC 15897)**:
  - Standard for defining locale settings in Unix-like operating systems
  - Specifies conventions for formatting dates, times, numbers, currency symbols, sorting order, etc.
  - Uses environment variables like LANG, LC_ALL, LC_COLLATE to set locale parameters

- Both CLDR and POSIX define locale data
- They partially overlap but have different scopes and implementations

Terminology

- **Unicode Collation Algorithm (UCA)**:
- Standard algorithm for comparing Unicode strings
- Supplies a default set of collation weights for Unicode characters
- Orders characters by several levels:
  - Primary level: base characters (e.g., ”a” vs. ”b”)
  - Secondary level: accents and diacritics (e.g., ”a” vs. ”á”)
  - Tertiary level: case and variant differences (e.g., ”a” vs. ”A”)

Table 1. Example Differences

Language	Swedish:	z < ö
	German:	ö < z
Usage	German Dictionary:	of < öf
	German Phonebook:	öf < of
Customizations	Upper-First	A < a
	Lower-First	a < A

Table 2. Comparison Levels

Level	Description	Examples
L1	Base characters	role < roles < rule
L2	Accents	role < rôle < roles
L3	Case/Variants	role < Role < rôle
L4	Punctuation	role < "role" < Role
Ln	Identical	role < ro□le < "role"

Examples of UCA from Unicode.org

Let's Summarize It in Simple Terms

credativ

- Unicode defines a default ordering
- But languages and regions have different rules
- A collation is a set of rules for specific language
- Defines comparing and ordering strings
- PostgreSQL up to version 17 does not implement its own collations
- Relies on OS-provided collations (ICU, libc)
- Collations affect text/varchar/char/string data types in:
  - ORDER BY results in queries
  - String comparisons in WHERE clauses (>, <)
  - B-tree indexes order and range searches
  - Unique constraints (can allow duplicates)
  - Range partition boundaries on text columns (records in wrong partition)
  - Materialize views with ORDER BY clauses
  - One case of crash during recovery WAL replay

Locales on Linux Systems

credativ

- Locales are defined on OS level
- Required language specific locale must be installed
- Command " `locale -a` " lists available locales
- File `/etc/locale.gen` defines installed locales
- On Debian/Ubuntu " `dpkg-reconfigure locales` " - add/remove locales
- But it is easier to uncomment needed locales in `/etc/locale.gen` and run " `locale-gen` "

Available Collations in PostgreSQL

credativ

- Catalog `pg_collation` maps SQL collation names to OS collation names
- Column `collprovider` shows which provider is used
- 'd' = database default, 'b' = builtin, 'c' = libc, 'i' = icu
- "Default" collation is placeholder, not a special collation
- Actual value is taken from database settings

```
select oid, collname, collnamespace::regnamespace, collprovider,
       colllocale, collctype, collencoding, collversion
from pg_collation where collprovider = 'd' order by collname;

 oid | collname | collnamespace | collprovider | colllocale | collctype | collencoding | collversion
-----+----------+---------------+--------------+------------+-----------+--------------+-------------
 100 | default  | pg_catalog    | d            |            |           |           -1 |
(1 row)
```

Built-in Collations

- Built-in collations introduced in PostgreSQL 17
- They are platform-independent, not relying on OS libraries
- They guarantee consistent behavior across installations
- Should be stable in scope of major PostgreSQL versions

```
select oid, collname, collnamespace::regnamespace, collprovider, colllocale,
       colltype, collencoding, collversion
from pg_collation where collprovider = 'b' order by collname;

 oid  |    collname     | collnamespace | collprovider |   colllocale    | colltype | collencoding | collversion
------+-----------------+---------------+--------------+-----------------+----------+--------------+-------------
  811 | pg_c_utf8       | pg_catalog    | b            | C.UTF-8         |          |            6 | 1
  962 | ucs_basic       | pg_catalog    | b            | C               |          |            6 | 1
 6411 | pg_unicode_fast | pg_catalog    | b            | PG_UNICODE_FAST |          |            6 | 1
(3 rows)
```

Which Collation Do I Use?

credativ

- Each database has a default collation
- Default collation is inherited by new tables and columns
- We can override the default collation per column

```
-- show current database collation info
SELECT
    pg_encoding_to_char(encoding) AS encoding,
    datlocprovider, datcollate, datctype, datcollversion,
    pg_database_collation_actual_version(oid) AS act_collversion
FROM pg_database
WHERE datname = current_database();

 encoding | datlocprovider | datcollate |  datctype  | datcollversion | act_collversion
----------+----------------+------------+------------+----------------+-----------------
 UTF8     | c              | en_US.utf8 | en_US.utf8 | 2.41           | 2.41
(1 row)
```

Which Collation Do I Use?

- We can list all databases with their collations
- Using `\l` in psql

```
postgres=# \l
                                                  List of databases
   Name    |  Owner   | Encoding | Locale Provider |  Collate   |   Ctype    | Locale | ICU Rules |   Access privileges
-----------+----------+----------+-----------------+------------+------------+--------+-----------+-----------------------
 postgres  | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           |
 template0 | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           | =c/postgres          +
           |          |          |                 |            |            |        |           | postgres=CTc/postgres
 template1 | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           | =c/postgres          +
           |          |          |                 |            |            |        |           | postgres=CTc/postgres
 test      | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           |
(4 rows)
```

Create Database with Specific Collation

credativ

- We can create a database with specific collation
- The specified collation must be available on the system
- And encoding must be compatible with locale settings
- Provider: 'libc', 'builtin', 'icu' (if the server was built with ICU support)

```
-- Create database with German collation with libc provider
CREATE DATABASE mydb
    LOCALE_PROVIDER = 'libc'
    LC_COLLATE = 'de_DE.utf8'
    LC_CTYPE = 'de_DE.utf8'
    TEMPLATE = template0;

-- Create database with built-in C.UTF-8 collation
CREATE DATABASE mydb
    LOCALE_PROVIDER = 'builtin'
    LOCALE          = 'C.UTF-8'
    TEMPLATE        = template0;
```

Set Collation per Column

credativ

- We can override the default collation per column
- Note: locale must be available on the system

```
CREATE TABLE product_catalog (
    product_id SERIAL PRIMARY KEY,
    -- English product name with US English collation
    name_en TEXT COLLATE "en_US.utf8" NOT NULL,
    -- German product name with German collation (handles ä, ö, ü, ß properly)
    name_de TEXT COLLATE "de_DE.utf8",
    -- French product name with French collation (handles accents properly)
    name_fr TEXT COLLATE "fr_FR.utf8",
    -- Product code that needs exact case-sensitive matching
    product_code VARCHAR(20) COLLATE "C" UNIQUE,
    -- Category that should be case-insensitive
    category VARCHAR(50) COLLATE "en_US.utf8",
    -- Binary collation for exact byte comparison
    binary_data TEXT COLLATE "C",
    -- ICU collation for Unicode normalization
    description TEXT COLLATE "und-x-icu",
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Show Collations for Columns

- We can list collations used by table columns
- Following query shows only non-default collations in specific table

```
SELECT table_schema, table_name, column_name, data_type, collation_name
FROM information_schema.columns
WHERE collation_name IS NOT NULL AND collation_name <> 'default'
AND table_name = 'product_catalog'
ORDER BY table_schema, table_name, column_name;

 table_schema |   table_name    | column_name  |     data_type     | collation_name
--------------+-----------------+--------------+-------------------+----------------
 public       | product_catalog | binary_data  | text              | C
 public       | product_catalog | category     | character varying | en_US.utf8
 public       | product_catalog | description  | text              | und-x-icu
 public       | product_catalog | name_de      | text              | de_DE.utf8
 public       | product_catalog | name_en      | text              | en_US.utf8
 public       | product_catalog | name_fr      | text              | fr_FR.utf8
 public       | product_catalog | product_code | character varying | C
(7 rows)
```

Show Collations for Indexes

- We can list collations used by indexes on text columns
- Here is a query to show such indexes in specific table

```
SELECT
    i.indexrelid::regclass::text AS index_name,
    c.collname                   AS collation_name,
    pg_get_indexdef(i.indexrelid) AS index_def
FROM (
SELECT
    indexrelid,
    indrelid,
    unnest(indcollation) AS coll_oid
FROM pg_index
) AS i
JOIN pg_collation c ON c.oid = i.coll_oid
WHERE i.coll_oid <> 0 AND i.indrelid::regclass::text = 'product_catalog'
ORDER BY table_name, index_name, collation_name;
```

```
          index_name            | collation_name |                                                    index_def
--------------------------------+----------------+-----------------------------------------------------------------------------------------------------------------
 idx_product_category           | en_US.utf8     | CREATE INDEX idx_product_category ON product_catalog USING btree (category)
 idx_product_name_de            | de_DE.utf8     | CREATE INDEX idx_product_name_de ON product_catalog USING btree (name_de)
 idx_product_name_en            | en_US.utf8     | CREATE INDEX idx_product_name_en ON product_catalog USING btree (name_en)
 idx_product_name_fr            | fr_FR.utf8     | CREATE INDEX idx_product_name_fr ON product_catalog USING btree (name_fr)
 product_catalog_product_code_key | C              | CREATE UNIQUE INDEX product_catalog_product_code_key ON product_catalog USING btree (product_code)
(5 rows)
```

Auditing Collations in a Database

credativ

- How many columns use specific collations in a database?

```
SELECT
    c.collname,
    c.collprovider,
    c.colllocale,
    COUNT(*) AS column_count
FROM pg_attribute a
JOIN pg_class t      ON t.oid = a.attrelid
JOIN pg_namespace n  ON n.oid = t.relnamespace
JOIN pg_collation c  ON c.oid = a.attcollation
WHERE a.attnum > 0
    AND NOT a.attisdropped
    AND t.relkind IN ('r','p','m','f')
GROUP BY c.collname, c.collprovider, c.colllocale
ORDER BY column_count DESC, c.collname;

  collname   | collprovider | colllocale | column_count
-------------+--------------+------------+--------------
 C           | c            |            |          119
 en_US.utf8  | c            |            |            2
 de_DE.utf8  | c            |            |            1
 fr_FR.utf8  | c            |            |            1
 und-x-icu   | i            | und        |            1
(5 rows)
```

Auditing Collations in a Database

- How many indexes use specific collations in a database?

```
SELECT
    c.collname,
    c.collprovider,
    COUNT(*) AS index_column_count
FROM pg_class idx
JOIN pg_attribute a ON a.attrelid = idx.oid
JOIN pg_collation c ON c.oid = a.attcollation
WHERE idx.relkind = 'i'
    AND a.attnum > 0
GROUP BY c.collname, c.collprovider
ORDER BY index_column_count DESC, c.collname;

  collname  | collprovider | index_column_count
------------+--------------+--------------------
 C          | c            |                 37
 en_US.utf8 | c            |                  2
 de_DE.utf8 | c            |                  1
 fr_FR.utf8 | c            |                  1
(4 rows)
```

Auditing Collations in a Database

- Which objects use specific collation in a database?
- Query checks for relations, types (like enum types) & domains

```
SELECT
d.classid::regclass AS object_class,
CASE
    WHEN d.classid = 'pg_class'::regclass THEN
        (select c.relname::text||' - '||c.relkind::text from pg_class c where oid = d.objid)
    WHEN d.classid = 'pg_type'::regclass  THEN
        d.objid::regtype::text
    ELSE d.objid::text
END AS object
FROM pg_depend d
WHERE d.refclassid = 'pg_collation'::regclass
    AND d.refobjid in
    (SELECT oid FROM pg_collation WHERE collname LIKE 'de_DE%');


object_class |         object
-------------+------------------------
pg_class     | product_catalog - r
pg_class     | idx_product_name_de - i
(2 rows)
```

Client-Side Collation Setting

credativ

- Client can set collation for its session
- Determines the *interpretation* of data sent from/to client
- Data is converted between client encoding and server encoding
- Fully automatic and transparent
- Affects string comparison and ordering in that session
- Does not affect database storage or other clients
- Can be changed arbitrarily at runtime

```
SET lc_collate = 'fr_FR.utf8';
```

Encoding Mismatch Errors

- Encoding mismatch between client and server can cause errors
- Example: client sends LATIN1 data to server expecting UTF-8
- Server cannot interpret byte sequences correctly
- Results in errors during string operations

```
-- Client encoding is LATIN1, server expects UTF8
SELECT 'äöü'::text;

INSERT INTO mytable (mycolumn) VALUES ('äöü'::text);

-- Both command will fail - but only if we test it with LATIN1 client encoding (!)
ERROR:  invalid byte sequence for encoding "UTF8": 0xe4 0xf6 0xfc
```

Locale Categories

- Locale settings are divided into categories
- Each category controls specific aspects of localization

```
SET lc_collate = 'de_DE.utf8';  -- string collation
SET lc_ctype = 'de_DE.utf8';    -- character classification
SET lc_monetary = 'de_DE.utf8'; -- monetary formatting
SET lc_numeric = 'de_DE.utf8';  -- number formatting
SET lc_time = 'de_DE.utf8';     -- date/time formatting
```

Changes in glibc 2.28+

credativ

- The most ”famous” example of breaking changes in collations
- glibc 2.28 introduced significant changes to collation rules
- Library was released in August 2018 and affected many Linux distributions

- Changes in many locales, including en_US.UTF-8, de_DE.UTF-8, fr_FR.UTF-8
- Aimed to improve compliance with Unicode Collation Algorithm (UCA)
- Resulted in different sorting orders for certain characters
- Applications relying on previous collation behavior experienced issues
- PostgreSQL users faced collation version mismatch errors after OS upgrade

- glibc 2.28: New and Updated Locales
- Debian mailing list: Glibc 2.28 breaks collation for PostgreSQL (and others?)

Collation Torture Test

credativ

- Set of 26 million strings designed to test collation behavior
- Created for all valid Unicode code points - 287K code points, 91 strings for each
- Repository glibc-unicode-sorting (Jeremy Schneider) contains tests for collation behavior
- Code generates string list on each tested OS version and results are compared
- Tests:
  - Interactions between character classes (letters, digits, punctuation, symbols, etc.)
  - Strings of different lengths with repeated characters
  - Cases of doubled characters - some languages treat them differently
  - Variations in case (uppercase, lowercase, title case)
  - Accented characters and diacritics

- Collation Torture Test versus Debian
- Collation Surprises: Did Postgres Lose My Data? (PASS 24 Data Community Summit)
- Case-Insensitive Pattern Matching in PostgreSQL (Cybertec)

Collation Torture Test Results

credativ

- Both glibc and ICU change collations regularly
- Each has had at least one release with huge, ecosystem impacting changes
- There are clear “dangerous” OS upgrade points for database workloads:
  - glibc 2.21 caused big issues on Ubuntu 15.04
  - glibc 2.28 on many distros caused massive changes (Debian, RHEL)
  - glibc 2.29+ always small changes on specific locales (Debian, RHEL)
- Even ICU is not immune to collation changes:
  - ICU 55.1 -> 60.2 (Ubuntu 18.04) - mainly in en-US locale
  - ICU 60.2 -> 63.1 (Ubuntu 19.04) - smaller changes in several locales
  - ICU 63.1 -> 66.1 (Ubuntu 20.04) - again changes in en-US
  - ICU 67.1 -> 70.1 (Ubuntu 22.04) - smaller changes

RedHat Example

credativ

- RHEL/CentOS 8 and 9 use glibc 2.28+
- Collation changes affect PostgreSQL installations on these OS versions
- Upgrading from RHEL/CentOS 7 to 8 or 9 can lead to collation version mismatches

```
-- Example sorting difference between RHEL 7 and RHEL 9
SELECT dat FROM (VALUES ('1-a'), ('1a'), ('1-aa')) v(dat) ORDER BY 1;

-- RHEL 7 - Red Hat Enterprise Linux Server release 7.9 (Maipo)
    dat
    ------
    1a
    1-a
    1-aa

-- RHEL 9 - Red Hat Enterprise Linux release 9.0 (Plow)
    dat
    ------
    1-a
    1a
    1-aa
```

Example from Collation Challenges - Sorting It Out (FOSDEM 2024)

RedHat Example

credativ

```
-- Example of unique index not working after OS upgrade
CREATE TABLE testcoll(f1 text primary key);
INSERT INTO testcoll (VALUES ('1-a'), ('1a'), ('1-aa'));

-- After upgrade from RHEL 7 to RHEL 9
SELECT f1 FROM testcoll ORDER BY 1;
    f1
    ------
    1a
    1-a
    1-aa
    (3 rows)

-- Before reindex we can insert duplicate value !!!
INSERT INTO testcoll VALUES ('1-a');
INSERT 0 1

-- Now reindex fails due to duplicate key
REINDEX TABLE testcoll;
2023-05-06 21:00:59.948 UTC [352755] ERROR: could not create unique index "testcoll_pkey"
2023-05-06 21:00:59.948 UTC [352755] DETAIL: Key (f1)=(1-a) is duplicated.
2023-05-06 21:00:59.948 UTC [352755] STATEMENT: REINDEX TABLE testcoll;
ERROR: could not create unique index "testcoll_pkey"
DETAIL: Key (f1)=(1-a) is duplicated.
```

Example from Collation Challenges - Sorting It Out (FOSDEM 2024)

Collation Mismatch Error after OS Upgrade

- OS upgrade can change version of libc or ICU
- Collation rules may change between versions
- Existing indexes may become invalid
- Results in errors during queries using those indexes

```
WARNING: database "test" has a collation version mismatch
DETAIL: The database was created using collation version 2.35, but the operating system provides version 2.36.
HINT: Rebuild all objects in this database that use the default collation and run ALTER DATABASE test REFRESH COLLATION ↩
   VERSION, or build PostgreSQL with the right library version

WARNING:  collation "xx-x-icu" has version mismatch
DETAIL:  The collation in the database was created using version 1.2.3.4, but the operating system provides version 2.3.4.5.
HINT:  Rebuild all objects affected by this collation and run ALTER COLLATION pg_catalog."xx-x-icu" REFRESH VERSION, or build↩
    PostgreSQL with the right library version.
```

Collation Mismatch Error after OS Upgrade

credativ

- To fix collation version mismatch:
  - Rebuild all indexes (and materialized views) using the affected collation
  - Use REINDEX command with CONCURRENTLY option to avoid locking
  - Check RANGE partitions on text columns for misplaced records
  - Run ALTER DATABASE … REFRESH COLLATION VERSION

```
REINDEX SCHEMA CONCURRENTLY xxxxx;
--or
REINDEX DATABASE CONCURRENTLY xxxxx;

ALTER DATABASE xxxxx REFRESH COLLATION VERSION;

NOTICE:  00000: changing version from 2.37 to 2.38
LOCATION:  AlterDatabaseRefreshColl, dbcommands.c:2397
ALTER DATABASE
```

Can Built-in Collations Help?

credativ

- Built-in collations introduced in PostgreSQL 17
- Provide consistent collation behavior across installations on different platforms
- For these databases collation mismatch errors cannot occur after OS upgrade
- But do not solve all problems
- Do not provide language-specific collations - PG still relies on OS for those
- Apps depending on ”de_DE.UTF-8”, ”fr_FR.UTF-8” etc. can still face issues after OS upgrade
- Behaviour of these collations is stable only within major PG versions
- REINDEX might still be needed after PG major upgrade

Summary - Hope is not a Strategy

credativ

- There is currently no simple way to avoid collation issues caused by upgrades
- Built-in PG collations are not language-specific & can change with PG major upgrades
- Even regular small OS libraries upgrades can change collation behavior
- If we depend on language-specific collations we must have some testing strategy
- We can use ”collation torture test” for evaluation of OS/library upgrade
- If we discover issues we must rebuild affected indexes & update database collation version

- Big OS upgrade & PostgreSQL major version upgrades are better done by migration:
  - Migrate data from old database/machine to a new one
  - Either using dump/restore or logical replication

Thank you for your attention!

credativ

All my slides