Meetup Paris juin 2018

PostgreSQL & ICU

@DanielVerite

A propos de moi

Daniel Vérité -- daniel@manitou-mail.org

Développeur indépendant, spécialisé bases de données et logiciels libres.

Présence en ligne

- Blog Postgres en français: https://blog-postgresql.verite.pro
- Forum https://forums.postgresql.fr
- StackOverflow / dba.stackexchange.com: https://stackoverflow.com/users/238814
- Twitter @DanielVerite

Projets PostgreSQL:

- Appli de mail en base de données: https://www.manitou-mail.com
- Séquences aléatoires uniques:http://github.com/dverite/permuteseq
- SHAnnn-CRYPT (mots de passe): https://github.com/dverite/postgres-shacrypt
- Chiffrement XTEA/Skip32: https://github.com/dverite/cryptint
- Extension ICU: https://github.com/dverite/icu_ext

Qu'est-ce que ICU ?

Ⓘnternational

Ⓒomponents for

Ⓤnicode

Bibliothèque C/C++/Java de référence pour Unicode

http://icu-project.org

Fonctionnalités d'ICU

- Conversion de tous les jeux de caractères de et vers Unicode
- **Comparaison de chaînes de caractères** avec des règles linguistiques.
- Formatage des nombres, dates, montants...
- Calendriers et fuseaux horaires
- Expressions rationnelles multilingues

Qu'est-ce qu'Unicode ?

Un projet

Mission depuis 1988: établir un **jeu universel de caractères** englobant :

- toutes les langues présentes et passées
- tous les symboles textuels

- utilisable par toutes les plateformes et langages informatiques
- disponible sur tous les périphériques

Qu'est-ce qu'Unicode ?

Une organisation (non-profit corporation)

https://www.unicode.org/consortium/consort.html

Full Members (Voting)

Adobe, Apple, Facebook, Google, HUAWEI, IBM, Microsoft, NETFLIX, ORACLE, SAP, shopify

Institutional Members (Voting)

Government of India, Berkeley

Qu'est-ce qu'Unicode ?

Une norme

ISO/IEC 10646:2017, 2700 pages.

Définit l'Universal Coded Character Set (UCS) et les encodages UTF-8, 16, 32…

~ 130000 caractères dans 146 écritures

Qu'est-ce qu'Unicode ?

Un standard

Unicode v11 sorti en juin 2018

- 684 nouveaux caractères
- 6 nouvelles écritures (« scripts »)

- 137439 caractères dont 7110 symboles

- des dizaines de documents techniques décrivant les algorithmes de gestion du texte.

Unicode hors ICU

PostgreSQL toutes versions gère Unicode sur le modèle POSIX / libc

- encodage UTF-8 (taille variable / caractère)
- LC_COLLATE pour collations = règles de tri. Niveau base / colonne / expression
- LC_CTYPE pour classification des caractères. Niveau base.

Collations implicites

Au départ: initdb créé automatiquement les collations disponibles dans l'environnement (`$ locale -a`)

Par la suite, on peut en ajouter via:

```
CREATE COLLATION [ IF NOT EXISTS ] name (
    [ LOCALE = locale, ]
    [ LC_COLLATE = lc_collate, ]
    [ LC_CTYPE = lc_ctype, ]
    [ PROVIDER = provider, ]
    [ VERSION = version ]
);
```

Collations explicites

Les collations sont héritées de template1, ou explicites, par exemple:

```
=# CREATE DATABASE mydb
    LC_COLLATE 'fr_FR.UTF-8'
    LC_CTYPE   'fr_FR.UTF-8';

=# CREATE TABLE codes (
    id SERIAL,
    code text COLLATE "C" -- ou "POSIX"
   );
```

Tris hétérogènes

Problème **1/2** : tris différents d'une plateforme à l'autre avec la même locale Unicode et la même version de PostgreSQL.

Exemple:

```
select * from (values ('"0102"'), ('0102')) as x(x)
 order by x collate "en_US";
```

FreeBSD 11:

```
    x
--------
 "0102"
 0102
```

Debian 9:

```
    x
--------
 0102
 "0102"
```

Mise à jour libc = risque

Problème ***2/2*** **:** manque de stabilité sur la même plateforme

Exemple avec COLLATE "`de_DE.UTF-8`", PG 9.4 sur CentOS:

```
SELECT * from values ('1-1-0'), ('110'),
 ('1-5-9'), ('159') AS x ORDER BY 1;
```

Avec CentOS 6:

```
 column1
---------
 1-1-0
 110
 1-5-9
 159
```

Avec CentOS 7:

```
 column1
---------
 110
 1-1-0
 159
 1-5-9
```

Versionnage des collations

Un index b-tree a absolument besoin de collations 100% stables.

Avec ICU:

1. Toute collation a un numéro de version vérifiable.
2. Aucune différence entre la même collation d'une plateforme à l'autre

⚠ glibc 2.28 (août 2018 ?) intègre une grosse mise à jour d'Unicode.

Créer une collation ICU

```
test=# CREATE COLLATION "frcoll" (
       locale='fr_FR', provider='icu');
CREATE COLLATION

test=# SELECT * FROM pg_catalog.pg_collation
       WHERE collname = 'frcoll' \gx

-[ RECORD 1 ]-+-------
collname      | frcoll
collnamespace | 2200
collowner     | 10
collprovider  | i
collencoding  | -1
collcollate   | fr_FR
collctype     | fr_FR
collversion   | 153.80
```

Noms de collations ICU

Les noms sont des étiquettes structurées suivant la norme **BCP-47** (IETF).

Exemples :

- `fr` : français (région, script, etc. déduits)
- `es-u-co-trad` : espagnol traditionnel.
- `fr-ca-u-ks-level2` : français du Canada avec comparaison insensible à la casse et ponctuation.
- `ru-Latn` : russe exprimé en alphabet latin.
- `en-US-u-va-posix` : comparaisons similaires à `LC_COLLATE=POSIX` avec la libc.

Versions des collations

Chaque collation a un numéro de version donné par une fonction ICU.
Si cette version change, PostgreSQL le détecte dès que la collation est utilisée.

```
test=# SELECT 'a' < 'B' collate "en-US-x-icu";
WARNING:  collation "en-US-x-icu" has version mismatch
DETAIL:  The collation in the database was created using version 58.0.6.50, but
the operating system provides version 153.80.
HINT:  Rebuild all objects affected by this collation and run ALTER COLLATION
pg_catalog."en-US-x-icu" REFRESH VERSION, or build PostgreSQL with the right
library version.
 ?column?
----------
 t
(1 row)
```

Il n'y a pas de test équivalent avec la libc.

Performances d'indexation

PostgreSQL utilise les clefs abrégées (abbreviated keys) d'ICU pour accélerer les comparaisons en masse.

Clés abrégées = représentations spécialement conçues pour se comparer entre elles en accéléré, car :

```
strcmp(abbrev(str1), abbrev(str2)) == strcoll(str1, str2)
```

Résultat typique : indexation 30 % plus rapide.

Exemple sur la base de stackoverflow (38 Go de posts) :

https://blog.anayrat.info/2017/11/19/postgresql-10--icu--abbreviated-keys/

Support incomplet des collations dans Postgres

Les collations insensibles à la casse ou aux accents ne sont pas vraiment utilisables avec PostgreSQL, à cause du tie-breaker dans les comparaisons:

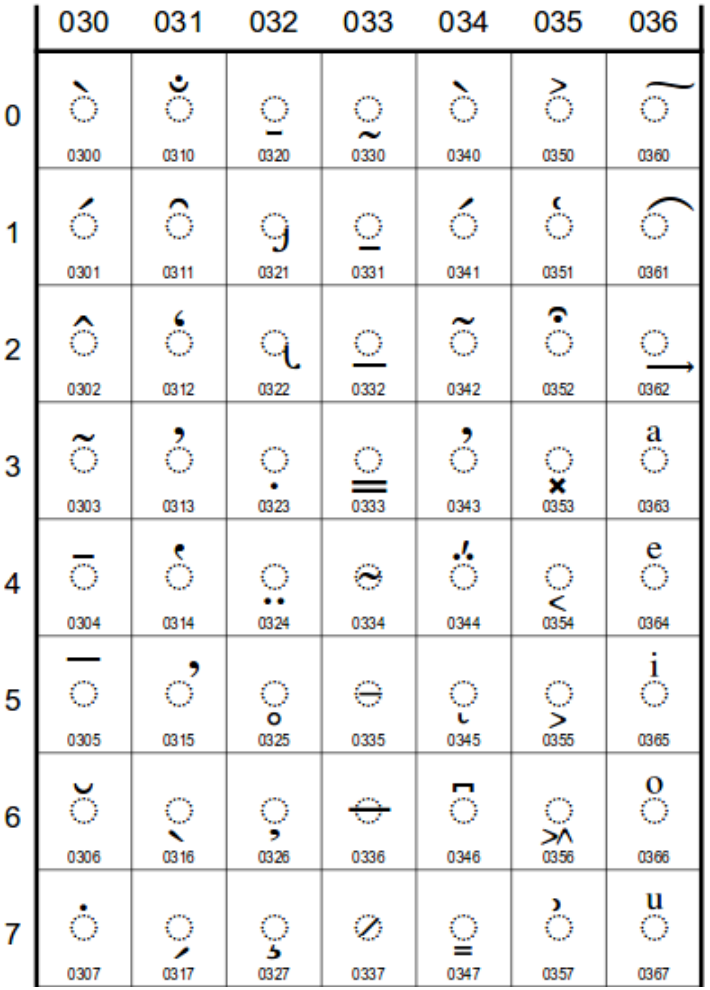
Si égalité linguistique entre s1 et s2
alors résultat = **comparaison binaire**

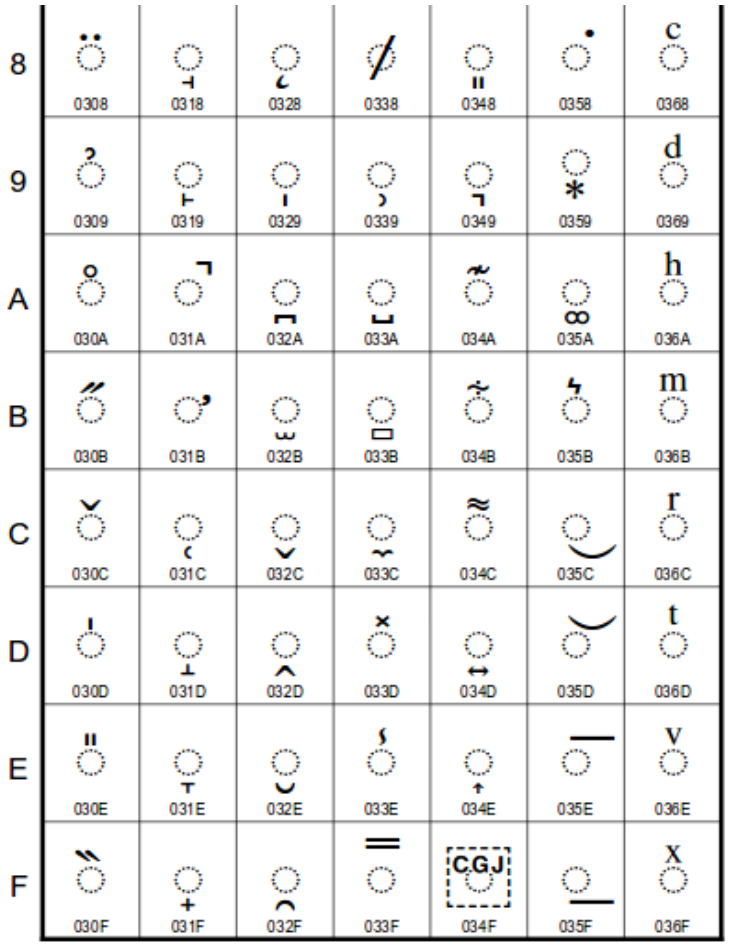
Cf fil de discussion https://bit.ly/2ItnpUM :

```
"strcmp() tie-breaker for identical ICU-collated strings"
```

Diacritiques combinants

U+0300 → U+036F https://www.unicode.org/charts/PDF/U0300.pdf





Diacritiques combinants

```
test=# SELECT E'a\u0300';
 ?column?
----------
 à
```

```
test=# SELECT E'a\u0300', 'à', E'a\u0300' = 'à';

 ?column? | ?column? | ?column?
----------+----------+----------
 à        | à        | f
```

Le résultat sera le même quelle que soit la collation.

⚠ Il n'est pas conforme au standard Unicode.

Fonctions C sous-jacentes aux comparaisons

- Libc POSIX

```
strcoll_l (
  const char *str1,
  const char* str2,
  locale_t loc)
```

- Windows

conversion en UTF-16 (wchar_t)

```
wcscoll_l (
  const wchar_t *str1,
  const wchar_t *str2,
  _locale_t loc)
```

- ICU 53+ et UTF-8

```
ucol_strcollUTF8 (
  const UCollator * coll,
  const char *source,
  int32_t sourceLength,
  const char *target,
  int32_t targetLength, ... )
```

- ICU pre-53 ou non UTF-8

conversion en UTF-16 (UChar)

```
ucol_strcoll (
  const UCollator * coll,
  const UChar *source,
  int32_t sourceLength,
  const UChar *target,
  int32_t targetLength )
```

Dans tous les cas **d'égalité**, il y a une deuxième comparaison avec strcmp()

Tri linguistique, voulu ou subi ?

A propos du coût du tri linguistique versus tri binaire:

R.Haas blog, 2012, "The Perils of Collation-Aware Comparisons":

« *I suspect there are a lot of people out there who are paying it more or less accidentally and don't really care very much about the underlying sorting behavior*»

http://rhaas.blogspot.com/2012/03/perils-of-collation-aware-comparisons.html

Extension icu_ext

icu_ext apporte des fonctions SQL exposant les fonctions en C de ICU :

http://icu-project.org/apiref/icu4c/

```
test=# \dx+ icu_ext
           Objects in extension "icu_ext"
                 Object description
-----------------------------------------------------
 function icu_case_compare(text,text)
 function icu_character_boundaries(text,text)
 function icu_char_name(character)
 function icu_collation_attributes(text,boolean)
 function icu_compare(text,text,text)
 function icu_confusable_strings_check(text,text)
 function icu_default_locale()
 function icu_line_boundaries(text,text)
 function icu_locales_list()
 function icu_number_spellout(double precision,text)
 function icu_sentence_boundaries(text,text)
 function icu_set_default_locale(text)
 function icu_sort_key(text,text)
 function icu_spoof_check(text)
 function icu_unicode_version()
 function icu_version()
 function icu_word_boundaries(text,text)
```

icu_compare

Comparaison de deux chaînes avec une collation ICU donnée

```
=# SELECT icu_compare('abcé','abce','en-u-ks-level1-kc-true');
 icu_compare
-------------
           0

=# SELECT icu_compare('Abcé','abce','en-u-ks-level1-kc-true');
 icu_compare
-------------
           1
```

icu_sort_key

- Renvoie la clé de tri (=clé abrégée) correspondant au texte dans une collation déterminée.
- Intérêt: ORDER BY et indexation

```
test=# select icu_sort_key($$C'est l'été$$, 'fr');
                  icu_sort_key
-----------------------------------------------
 \x2d0969314d4f043f0969314f31013d88448801dc10
(1 row)

test=# select icu_sort_key($$C'est l'été$$, 'fr-u-ks-level1');
          icu_sort_key
-------------------------------
 \x2d0969314d4f043f0969314f31
(1 row)
```

Indexation sur icu_sort_key

Exemple d'index unique avec collation insensible à accents et casse

```
test=# CREATE TABLE uniq(name text);
CREATE TABLE

test=# CREATE UNIQUE INDEX idx ON uniq((icu_sort_key(name,
'fr-u-ks-level1')));
CREATE INDEX

test=# insert into uniq values('Philéas');
INSERT 0 1

test=# insert into uniq values('phileas');
ERROR:  duplicate key value violates unique constraint "idx"
DETAIL:  Key (icu_sort_key(name, 'fr-u-ks-
level1'::text))=(\x4737393f31294d) already exists.
```

icu_collation_attributes

Liste toutes les caractéristiques d'une collation ICU.

Intérêt: indique si la collation fait bien ce qu'on attend.

```
test=# select * from icu_collation_attributes('fr-CA-u-ks-level1-x-icu');
  attribute  |                              value
-------------+-----------------------------------------------------
 displayname | français (Canada, colstrength=primary, Usage privé=icu)
 kn          | false
 kb          | true
 kk          | false
 ka          | noignore
 ks          | level1
 kf          | false
 kc          | false
 version     | 153.80.33
(9 rows)
```

icu_locales_list

```
test=# SELECT name,country,country_code,language
   FROM icu_locales_list() WHERE name like 'es%' limit 5;

  name   |     country      | country_code | language
---------+------------------+--------------+----------
 es      |                  |              | espagnol
 es_419  | Amérique latine  |              | espagnol
 es_AR   | Argentine        | ARG          | espagnol
 es_BO   | Bolivie          | BOL          | espagnol
 es_BR   | Brésil           | BRA          | espagnol
(5 rows)
```

Ces noms correspondent aux collations ICU créées par initdb au moment de l'initialisation de l'instance.

Exemple: **es_AR** => (collname: 'es-AR-x-icu', collcollate : 'es_AR')

icu_case_compare

Compare deux chaînes indépendamment de la casse et sans collation.

```
test=# select icu_case_compare('Ich muß essen', 'Ich MUSS ESSEN');
 icu_case_compare
------------------
                0

test=# select upper('Ich muß essen') = upper('Ich MUSS ESSEN');
 ?column?
----------
 f


test=# select lower('Ich muß essen') = lower('Ich MUSS ESSEN');
 ?column?
----------
 f
```

Anti spoofing/phishing

- icu_spoof_check

```
=# SELECT txt, icu_spoof_check(txt) FROM
            (VALUES ('paypal'), (E'p\u0430ypal')) AS s(txt);

  txt   | icu_spoof_check
--------+-----------------
 paypal | f
 pаypal | t
```

- icu_confusable_strings_check

```
=# SELECT txt, icu_confusable_strings_check('phil', txt) AS confusable
    FROM (VALUES ('phiL'), ('phiI'), ('phi1'), (E'ph\u0131l')) AS s(txt);

 txt  | confusable
------+------------
 phiL | f
 phiI | t
 phi1 | t
 phıl | t
```

Analyse de chaîne 1/2

`icu_{character,word,line,sentence}_boundaries` permettent de découper une chaîne de caractères en prenant en compte une locale et avec les algorithmes préconisés par Unicode.

Exemple:

```
test=# SELECT * FROM icu_sentence_boundaries(
   'M. De Niro est au festival.'
   ' Il réside au Carlton.', 'fr-u-ss-standard');

 tag |           contents
-----+-------------------------------
   0 | M. De Niro est au festival.
   0 | Il réside au Carlton.
(2 rows)
```

Analyse de chaîne 2/2

Combiné avec `icu_char_name,` on peut analyser en détail le contenu des chaînes de caractère.

Exemple réel: pourquoi cette adresse mail est refusée?

monique.[redacted]@laposte.net
lpn-prd-vrin019
Remote Server returned '550 5.1.17 SMTPSEND.Utf8RecipientAddress; UTF-8 recipient address not supported.'

```
test=# SELECT x, icu_char_name(x), to_hex(ascii(x))
   FROM icu_character_boundaries('m‌onique.etc@laposte.net', 'en') as x;

 x |    icu_char_name     | to_hex
---+----------------------+--------
 ? | ZERO WIDTH SPACE     | 200b
 m | LATIN SMALL LETTER M | 6d
 o | LATIN SMALL LETTER O | 6f
 n | LATIN SMALL LETTER N | 6e
 i | LATIN SMALL LETTER I | 69
 q | LATIN SMALL LETTER Q | 71
 u | LATIN SMALL LETTER U | 75
 e | LATIN SMALL LETTER E | 65
 . | FULL STOP            | 2e
```

A venir dans icu_ext ?

- Expressions rationnelles
- Recherches de sous-chaîne
- Transformations / translitérations
- IDNA (noms de domaines)