

Open in app ↗

Medium

 Search

# PostgreSQL and UTF8 Related Issues



Josef Machytka

4 min read · Just now



Listen



Share



More

*PostgreSQL is strongly UTF-8 oriented, but some foreign data wrappers allow us to import invalid UTF-8 data into tables and use them in SELECTs.*

PostgreSQL supports multiple character sets, allowing us to store text in a variety of encodings. Documentation states that all supported charsets can be used by the client (parameter `client_encoding`), but not all are supported for server-side encoding. There are also some other limitations regarding compatibility of the database charset and sorting order (collation).

In practice, most PostgreSQL databases use the default UTF-8 encoding, with collation and `ctype` determined by the operating system. For example:

- **Linux:** Typically `C.UTF-8` or `en_US.utf8`
- **macOS:** `en_US.UTF-8`
- **Windows:** Usually a national collation, such as `English_Germany.1252`.  
(PostgreSQL documentation states that UTF-8 encoding can be used with any locale on Windows.)

To specify a different collation or `ctype` for the database, we can include it in the `CREATE DATABASE` command or apply it later in queries using `COLLATION 'xx_XX'`.

UTF charsets are standard across programming languages, so we usually do not see many problems with character sets in PostgreSQL. But problems can occur when we need to migrate data from some other database into PostgreSQL — especially older

data, created in times when a much bigger variety of charsets was used across different applications and systems.

A typical error people often encounter when migrating into PostgreSQL from databases like Oracle or Sybase ASE is a problem with the “zero byte character”:

```
[22021]: ERROR: invalid byte sequence for encoding "UTF8": 0x00
```

This limitation was implemented in PostgreSQL because it internally used old C-style strings, which in the past were terminated with a C-null character (i.e., the 0x00 byte). Looking into PostgreSQL code, comments for some already existing conversion functions say that for conversion these functions replace the 0x00 byte with an empty string. This is a reasonable workaround, but some clients are concerned that some information could be lost this way. The answer is not simple and depends on the application that consumes the data.

The PostgreSQL community discussed several solutions for this problem. Usage of `bytea` fields is not as simple as `TEXT` data, therefore existing best practices suggest either replacing 0x00 bytes with an empty string, or using some special escape sequence like `\0`, or using the special UTF-8 U+FFFD character (marked as the 'replacement character'). In the case of the last two options, information is at least not fully lost, but modified, and places where the 0x00 character was replaced can be identified.

Here our story might end, PostgreSQL does not allow insertion of invalid characters. But working on different migration tasks, I have encountered some anomalies in PostgreSQL's otherwise strictly UTF-8-oriented behavior.

I found that by using some foreign data wrappers, it is possible to actually import data in different charsets into PostgreSQL and use them in queries, and the database will not raise any error. An error would occur only if I tried to modify such data. Maybe this is nothing new, but so far, I have not found any info about this.

How it works can be seen in the following example with the `file_fdw` extension, which I will use to access an external CSV file. I used a simple Python code and

generated a very small CSV file with short strings in the Latin-1 character set. The code used these words with Latin-1 characters (here written in UTF-8 of course):

- “über”
- “façade”
- “Café”
- “naïve”
- “élève”

Opening that file in Visual Studio Code, I see this:

```
id,text
1,über
2,façade
3,Café
4,naïve
5,élève
6,normal_valid_text
```

Then I attached it to PostgreSQL using `file_fdw`:

```
CREATE EXTENSION file_fdw;
CREATE SERVER csv_files FOREIGN DATA WRAPPER file_fdw;

CREATE FOREIGN TABLE t_latin_1 (id INT, textvalue TEXT)
  SERVER csv_files
  OPTIONS (filename 'test_data_latin-1.csv', format 'csv', header 'on');
```

Now, when I try to select data from that table, I see:

```
SELECT * FROM t_latin_1;

id | textvalue
---+-----
 1 | über
```

```

2 | faade
3 | Caf
4 | naive
5 | lve
6 | normal_valid_text
(6 rows)

```

No error message on select — PostgreSQL does not check validity of strings. I tried to copy data inside PostgreSQL:

```
CREATE TABLE local_t_latin_1 AS SELECT * FROM t_latin_1;
```

It works, no error message, obviously no check of validity of strings either. I select data from the local copy and see exactly that distorted output as above. No error message about invalid strings. It looks like PostgreSQL checks validity of strings only during some operations.

So I dump data into SQL format:

```
pg_dump -d postgres -U postgres --table=local_t_latin_1 > local_t_latin_1.sql
```

I get an SQL file that contains these data for the table:

```

CREATE TABLE public.local_t_latin_1 (
    id integer,
    textvalue text
);

ALTER TABLE public.local_t_latin_1 OWNER TO postgres;
--
-- Data for Name: local_t_latin_1; Type: TABLE DATA; Schema: public; Owner: pos
--
COPY public.local_t_latin_1 (id, textvalue) FROM stdin;
1 <EF><BF><BD>ber
2 fa<EF><BF><BD>ade
3 Caf<EF><BF><BD>

```

```
4 na<EF><BF><BD>ve
5 <EF><BF><BD>l<EF><BF><BD>ve
6 normal_valid_text
\.
```

But I cannot restore this text SQL file; the data is wrong:

```
ERROR: invalid byte sequence for encoding "UTF8": 0xfc
CONTEXT: COPY local_t_latin_1, line 1
```

On the other hand, a custom format dump seems to work. It is in a binary format and therefore is restorable because it is most likely inserted in a different way.

## Summary

I tested this behavior in different versions of PostgreSQL, and it is the same in all the latest versions, including 17. I also find the same behavior (surprisingly) in `JDBC_FDW`, which actually claims to be using UTF-8 by default. But I was able to copy data in different charsets from other databases into local PostgreSQL tables using this FDW when I directly copied them from foreign tables. And if I use them only in `SELECTs`, I get no error message. Only if I tried to update that particular value while trying to preserve all those characters invalid from the point of UTF-8. I do not know if this is behavior by design or just a flaw in implementation, but currently it works.





Image created by the author using DeepDreamGenerator

Postgresql

Utf 8

Data Migration



Edit profile

Written by Josef Machytka

58 Followers · 15 Following

I work as PostgreSQL specialist & database reliability engineer at NetApp Deutschland, Open Source Services division.

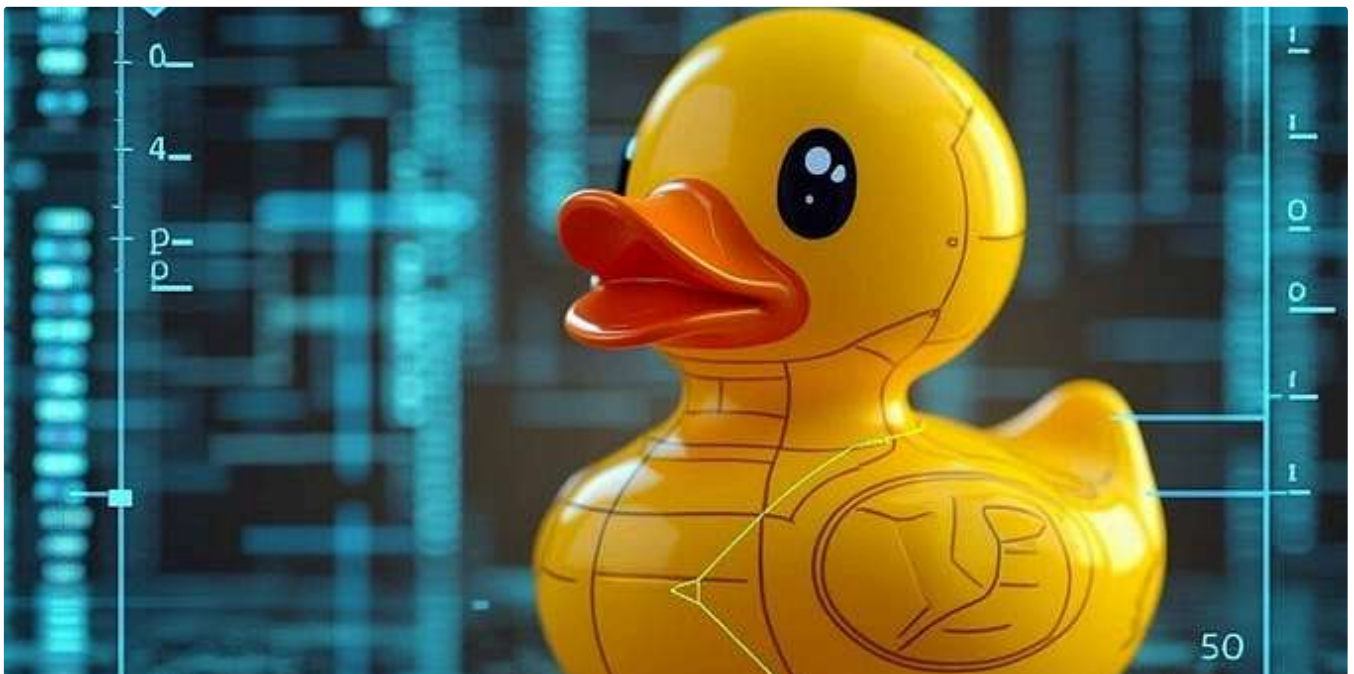
No responses yet



What are your thoughts?

Respond

More from Josef Machytka



Josef Machytka

## DuckDB Database File as a New Standard for Sharing Data?

This is not my original idea; I came across it in an excellent article titled “DuckDB Beyond the Hype” by Alireza Sadeghi. However, it...



Dec 30, 2024 17 2



probe_id int64	2024-12-18_avg_tem. double	2024-12-19_avg_tem. double	2024-12-20_avg_tem. double	2024-12-21_avg_tem. double	2024-12-27_avg_tem. double	2024-12-28_avg_tem. double	2024-12-29_avg_tem. double	2024-12-30_avg_tem. double	
1	-0.09333333333333331	0.008709499040937112	-0.0951097961531423	0.018285515229008652	0.02408290016364426	-0.06485408315666406	0.012230606453306028	2.89	
2	-1.6681481481481474	-0.011747858418378	-0.19133368175211993	-0.06308527766649575	-0.04171747826158102	-0.02843359476698743	-0.025157660909087	-31.82	
3	0.5514285714285712	0.0376647002665746	0.2730037656149165	0.1404807877274826	0.011376852694737617	0.05923246646506877	0.08681195292101234	-3.990000000000001	
4	0.15520325203252047	-0.00869218717319168	0.08583476862546609	-0.06119660214476116	0.10196071702661273	-0.08186630396378788	0.08996674464646202	19.36	
5	0.5163888888888885	-0.04570049073644817	0.01953642221439854	-0.015513540007672	-0.0135900022754125	-0.07238289422942779	-0.1835378423480908		
6	1.016548672566371	0.02907114059955196	0.08107486316968297	0.017795231646764282	0.11904802583941525	0.04725992074913865	0.08915156107911504	-27.380000000000003	
7	0.07568807339449547	0.14391917325689826	0.0551261347262228	-0.09983962752198643	0.16227462051478873	-0.15890124113132445	-0.05464776857339331		
8	2.0346000000000001	0.1148238688388646	0.09521098292944779	0.06277199528672414	0.1003629933070622	0.137548212300761314	0.05208853538214023	17.95	
9	1.545056179775286	0.0905560777879753	-0.03765412237862671	0.046972677461617703	-0.06521116185527977	-0.003200910688735	0.067779515103001	-19.71	
10	-0.04380000000000001	-0.20492896786129136	-0.08407264902227768	-0.055358886158886	0.061434627070469434	0.08977896362267106	0.07613119220133417	-13.08	
11	-3.7764347626869957	-0.10416217933853829	0.07367131734759187	0.10342240982853617	0.04691991417915546	0.0842402263416545	0.07204998928179399	-19.3	
12	5.5206666666666666	0.2100927160344700	0.10093850710864261	-0.05575695574662115	0.06861218940926169	-0.0077777932469663	0.03304196580641206	45.96	
13	-0.4471967616822432	-0.04059024910542247	0.05239493131063208	-0.0214055895940591	-0.03592269770578314	0.07835406119365161	-0.03261512699306271	4.923333333333334	
14	-2.0013392857142858	0.07316668336395598	-0.050057994486042	-0.02237049309405826	0.039495450804054426	0.07728930017610073	0.03379195482260548	-1.7650000000000006	
15	-2.2092391304347827	-0.00514496824155	-0.056626768137670	0.002132036763498929	-0.0502924418318254	0.05335375404928863	0.06826872209785106	5.0200000000000005	
16	-5.104608095652175	-0.1803382623814361	0.012928384872972334	-0.059576542858545	-0.09010858405407832	0.03176683704763877	-0.0086681223794503	20.44	
17	4.781382978723486	0.0659882476780733	0.16317028213207262	-0.07717688773880778	0.0336502038722565	0.0891822381972314	0.0912850507993609	-13.65	
18	0.09807256637168122	-0.002597352359666	0.038032066751196233	0.05646905735349645	-0.056157513598531	0.04520215540037235	-0.0746272461962213		
19	2.093851063829783	-0.011483227106448	-0.02934499864170985	-0.12917833341526482	-0.09808158720020972	-0.06237591444411659	-0.1436466606662121		
20	0.6865853658536589	-0.1047552749540883	-0.02643016057244126	-0.05337479195569904	-0.0348886009314119	0.004723957698249741	-0.09954026202597237		
981	6.274818101818182	-0.05899642646771017	-0.15701241468168753	-0.014281049403521	0.005729976703744055	0.07811998462155151	0.00303317574310037	35.695	
982	4.146236559139785	-0.006120066473466	-0.12709462668479013	0.07320853830092017	0.160077813251191	0.0847571866019098	0.060769155358762025	45.92	
983	-2.24244889795910374	0.1292289729655479	-0.03578956372968345	-0.11036742167256543	0.0985075288287772	0.08381679949749558	0.130114031511466	26.12	
984	-4.8618918918918919	-0.014959834176213	-0.06395120195650242	0.10667740909053037	0.046130732866371114	0.10371431376077715	-0.1831907299622613	0.035	
985	-1.8353846515846156	0.003296305450702291	0.09753330664909428	0.1435671160092685	-0.146125083800932	0.04264527854992811	0.04527577740735554	45.31	
986	-4.5237272727272724	0.021925843797540274	0.018245244860026953	-0.026906124387028	0.00700146657595972	0.05016651930445020	0.09480504012551043	8.267499999999999	
987	0.940693069306931	0.17302650062481123	-0.12358152747873527	-0.04427029407290484	0.0031278984522367	0.057873365677165	0.05902411002970675	7.306666666666666	
988	-0.909724770642023	-0.02859900578515256	-0.0325738158877223	0.024569307822198974	-0.04084684167358322	0.084980424344399809	-0.0081810140448637	-17.515	
989	3.5681806792452837	0.19396788557089517	0.09409573405199327	-0.1733985185185182	0.0817664332403276	0.0632990507794010	-0.0953987805679721	6.27	
990	-4.490602409638554	-0.012684438773310816	0.1326593665410001	-0.1005713371668568	0.0382005065532511	-0.010583178705726	-0.024069368211442	2.5549999999999997	
991	-2.127849462365592	0.06413518089512372	0.03634113253189271	0.0526666952410176	0.1177017821684689	0.0043562066936853	-0.002490603653953		
992	-0.580091743119266	-0.02636662790196389	0.06159776085763601	0.0568653072739868	-0.11079811859732074	-0.07561719025622922	-0.002432837532352	-46.48	
993	3.4080990825688072	-0.0359274650840987	-0.0327235366839524	-0.055926547933885	-0.005098547548169	-0.030956338743224	-0.10192112685241057		
994	1.014807218045113	0.06102599214551422	-0.02018151491365789	0.0930426049610529	0.09056344001167159	0.10671040530775777	-0.09901044093960255		
995	-2.29923076923077	0.20774707867194425	0.04415908120009207	-0.0553594598366952	-0.0678010119571237	0.05968745894988238	0.011847770143864587		
996	1.4535245901639349	0.024149191364233966	0.04559683800928681	-0.13891605347874647	-0.08022091745526695	-0.032139952113877	0.04877205139074573	40.25	
997	-2.591138211382114	-0.00115191341321969	-0.06748401691747602	-0.12257084855994097	-0.07479163742613288	0.09856108471696426	0.020141248435970503		

Josef Machytka

## DuckDB Performance Problems with Inappropriate Pivoting Queries on Very Large Datasets

The DuckDB documentation clearly states that this tool is designed for handling datasets fitting into memory. I fully understand that I'm...

Jan 3 6



Josef Machytka

## Quick and Easy Data Exports to Parquet Format Using DuckDB



The Parquet format has become almost an industry standard for Data Lakes and Data Lakehouses, thanks to its efficiency and compact storage...

Dec 5, 2024 🖱️ 17



```
ate the table
TABLE special_data_types (
  INT AUTO_INCREMENT PRIMARY KEY,
  me VARCHAR(50) NOT NULL,
  atus ENUM('active', 'inactive', 'pending') NOT NULL,
  missions SET('read', 'write', 'execute') NOT NULL,
  all_number TINYINT NOT NULL,
  dium_number MEDIUMINT NOT NULL,
  scription TEXT,
  ta BLOB,
  eated_at DATE NOT NULL

ert 10 rows of data
INTO special_data_types (name, status, permissions, small_number, medium_number, description, data, created_at)
e', 'active', 'read,write', 5, 1000, 'Alice description', 'Alice data', '2023-01-01'),
, 'inactive', 'read', 10, 2000, 'Bob description', 'Bob data', '2023-02-01'),
lie', 'pending', 'write,execute', 15, 3000, 'Charlie description', 'Charlie data', '2023-03-01'),
d', 'active', 'read,write,execute', 20, 4000, 'David description', 'David data', '2023-04-01'),
, 'inactive', 'execute', 25, 5000, 'Eve description', 'Eve data', '2023-05-01'),
k', 'pending', 'read,write', 30, 6000, 'Frank description', 'Frank data', '2023-06-01'),
e', 'active', 'read', 35, 7000, 'Grace description', 'Grace data', '2023-07-01'),
, 'inactive', 'write,execute', 40, 8000, 'Hank description', 'Hank data', '2023-08-01'),
, 'pending', 'read,write,execute', 45, 9000, 'Ivy description', 'Ivy data', '2023-09-01'),
, 'active', 'execute', 50, 10000, 'Jack description', 'Jack data', '2023-10-01');
```



Josef Machytka

## DuckDB as a Rudimentary Data Migration Tool

After exploring how to use DuckDB as an intelligent ETL tool for PostgreSQL, and how to extend its ETL capabilities with simple Python...

Nov 30, 2024 🖱️ 26 💬 2



See all from Josef Machytka

## Recommended from Medium



ACTIVE

**CANCELED**

## TN PROGRESS



## PostgreSQL Domain Types and Enums: Ensuring Data Integrity

Jan 6  14

## Postgres Security 101: User Access and Authorization (4/8)

<https://medium.com/@josef.machytka/postgresql-and-utf8-related-issues-85d36818eebd>

Oct 4, 2024 🖱 93



## Lists



### Staff picks

800 stories · 1569 saves



### Stories to Help You Level-Up at Work

19 stories · 920 saves



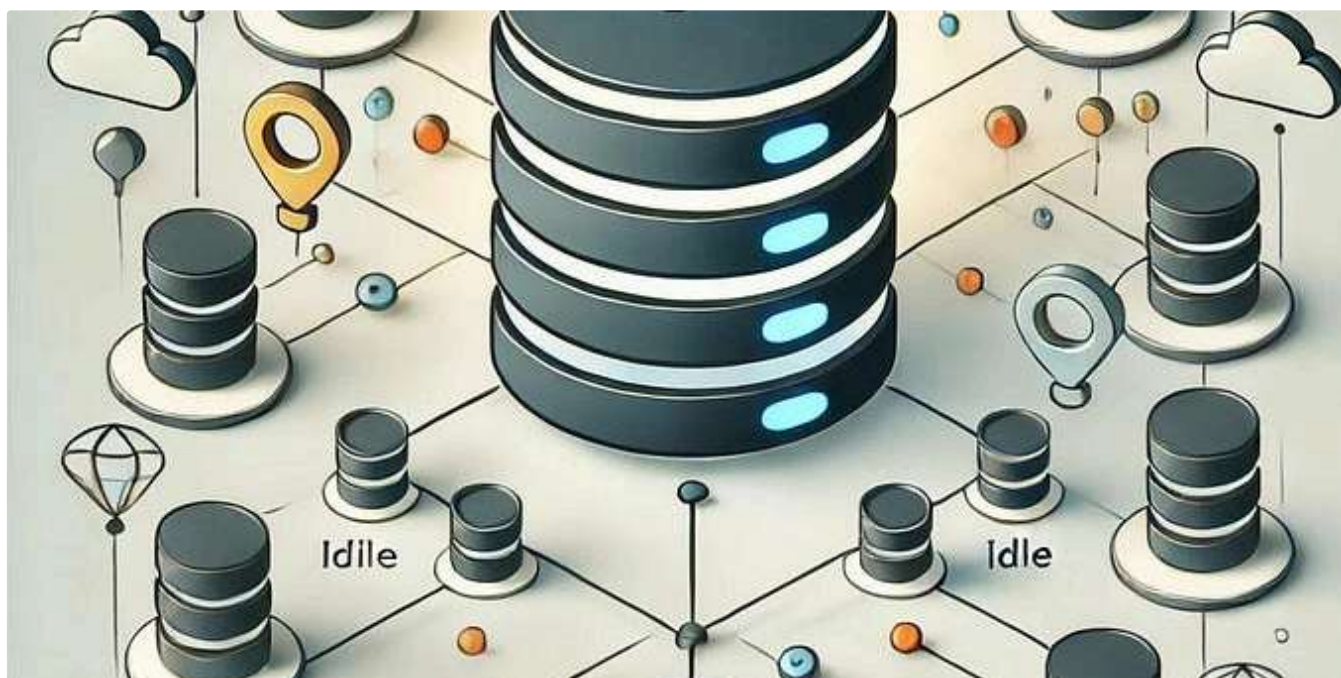
### Self-Improvement 101

20 stories · 3223 saves



### Productivity 101

20 stories · 2723 saves



Sruthi Ganesh

## Managing Open Idle Connections in PostgreSQL


Managing database connections is a critical aspect of administering a PostgreSQL database. Leaving unused or stale connections open can...

Dec 30, 2024 🖱 3







 In Stackademic by Mayur Koshti

## Native Support for Advanced Data Types in PostgreSQL

Practical Examples of Using PostgreSQL's Advanced Data Types

★ 4d ago 🖱 73 💬 1



 In Oracle Developers by Sharad Chandran

## Boost your Application Performance with Oracle Connection Pooling in Django

This is a guest post by Suraj Shaw, a member of Oracle Database's Technical Staff, who works on language drivers and frameworks. This post...

Oct 29, 2024 🖱 3



```
file_1.py > main
71 def main():
72     # if batch size reaches 25 items, write to dynamodb and reset
73     if len(records_batch) == 25:
74         batch_write(table, records_batch)
75         records_batch = []
76         print(f"Written {i} records")
77
78     # Write any remaining records
79     if records_batch:
80         batch_write(table, records_batch)
81         print(f"Written remaining {len(records_batch)} records")
82
83 if __name__ == '__main__':
84     main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console

Written 750 records  
Written 775 records  
Written 800 records  
Written 825 records  
Written 850 records  
Written 875 records  
Written 900 records  
Written 925 records  
Written 950 records  
Written 975 records  
Written 1000 records

Dmitry Romanoff

## Efficient Batch Writing to DynamoDB with Python: A Step-by-Step Guide

When working with AWS DynamoDB, especially for applications that need to handle large volumes of data, efficient record insertion is...

Jan 8 🖱 3

[See more recommendations](#)