When 7-bit ASCII ain't enough

- about NLS, Collation, Charsets, Unicode and such

Kim

@kibeha

https://kibeha.dk

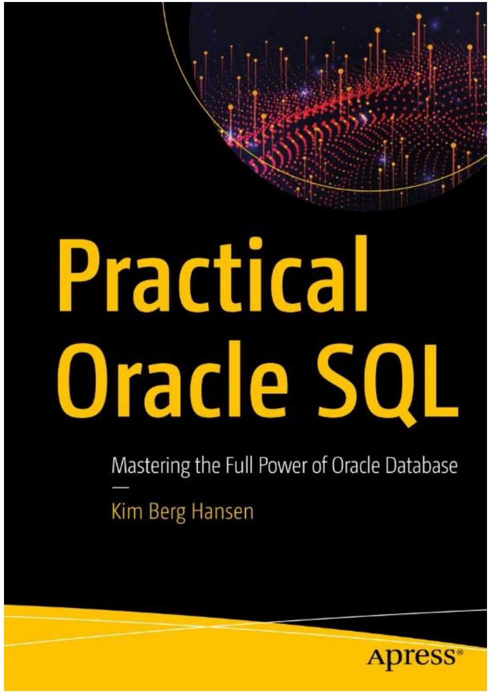
About me

- Danish geek
- SQL & PL/SQL developer since 2000
- Developer at Trivadis since 2016 https://www.trivadis.com
- Oracle Certified Expert in SQL
- Oracle ACE Director
- SQL quizmaster https://devgym.oracle.com
- Blogger https://kibeha.dk
- Likes to cook and read sci-fi
- Member of Danish Beer Enthusiasts

@kibeha

Author of "Practical Oracle SQL"

- Not a SQL-101 book
- Not a reference manual replacement
- For developers knowing basic SQL-92 syntax but wanting to advance further
- More elaborate examples relating to daily life as very simple examples are difficult to relate to work
- Useful SQL features that aren't widely used - but should be
- More background in an interview in NoCOUG Journal: http://nocoug.org/Journal/NoCOUG_Journal_202002.pdf#page=4
- The book:
  https://www.apress.com/gp/book/9781484256169
  https://www.amazon.com/Practical-Oracle-SQL-Mastering-Database/dp/1484256166



MASH PROGRAM

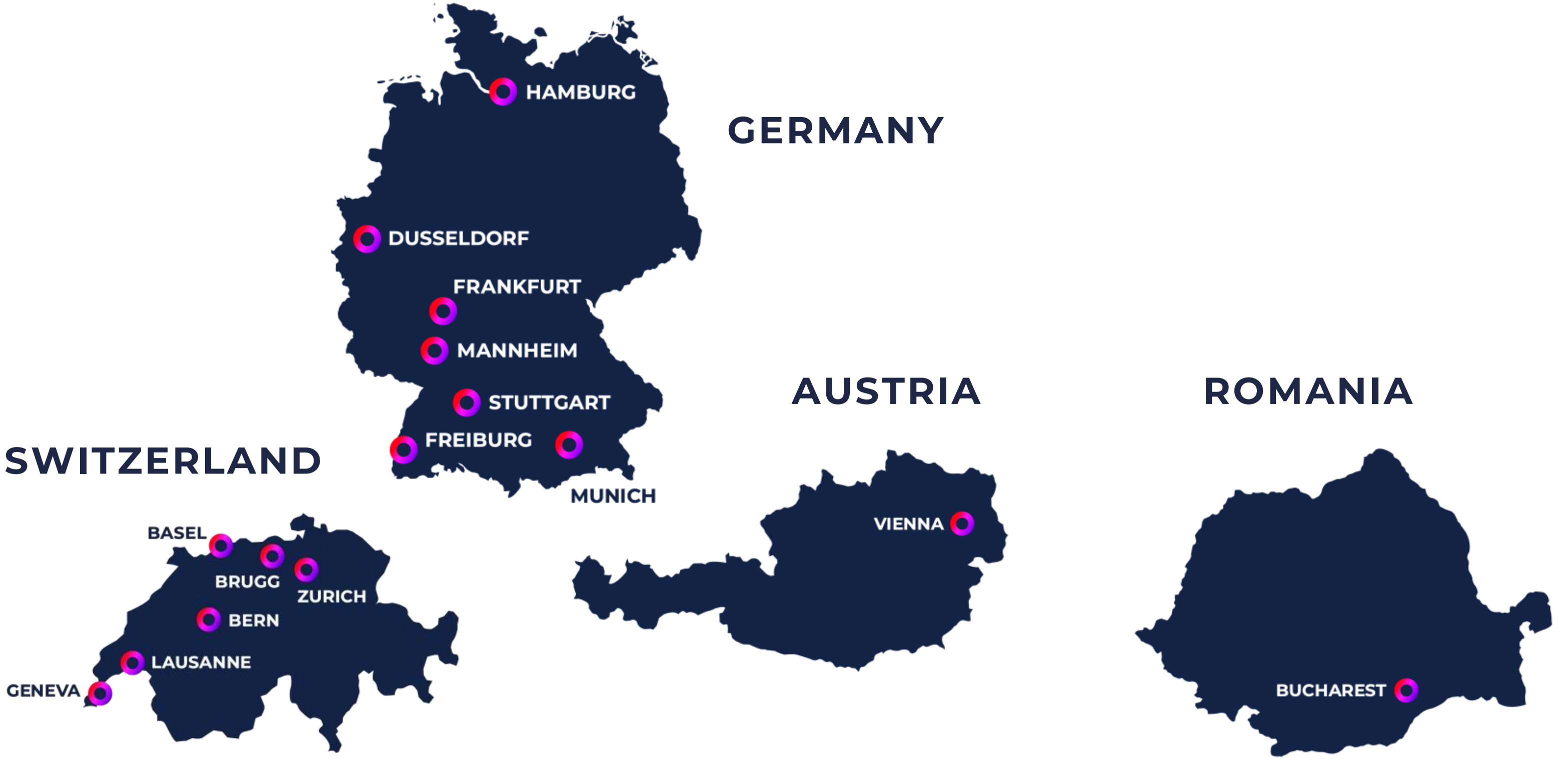
Mentor and Speaker Hub

- Our goal is to *connect* speakers with mentors to assist in *preparing* technical sessions and *improving* presentation skills

Interested? Read more and get in touch

https://mashprogram.wordpress.com

5 TRIVADIS - PART OF ACCENTURE



Agenda

- Circumstances leading to invalid characters in the database
- Character sets and encodings
- Database character set / National character set
- BYTE versus CHAR length semantics
- NLS settings
- Linguistic sorting and matching
- Data-bound collation
- Database Migration Assistant for Unicode (DMU)

trivadis

Circumstances leading to invalid characters in the database

"Stop! You're showelling bits the wrong way!"

Testing inserts

A simple test table

```
create table movies (
   seq         integer
 , title       varchar2(30 char)
 , inserted_by varchar2(30 char)
);
```

Database characterset the 12.2 default AL32UTF8

```
select parameter, value from nls_database_parameters where parameter = 'NLS_CHARACTERSET';

PARAMETER            VALUE
-------------------- --------------------
NLS_CHARACTERSET     AL32UTF8
```

Good Linux UTF-8 insert

NLS_LANG in Linux session has been set to match OS locale
It happens to match DB charset - UTF characters passed unchanged back and forth

```
[oracle@vbgeneric ~]$ locale | grep LANG
LANG=en_US.UTF-8
[oracle@vbgeneric ~]$ export NLS_LANG=american_america.al32utf8


insert into movies values (1, 'Jesús, nuestro Señor', 'Lin UTF-8 AL32UTF8');

select inserted_by, title, lengthb(title) as bytes, length(title) as chars
     , dump(title) as title_dump from movies;

INSERTED_BY          TITLE                   BYTES CHARS TITLE_DUMP
-------------------- ----------------------- ----- ----- --------------------------------
Lin UTF-8 AL32UTF8   Jesús, nuestro Señor       22    20 Typ=1 Len=22: 74,101,115,195,186
                                                         ,115,44,32,110,117,101,115,116,1
                                                         14,111,32,83,101,195,177,111,114
```

Good Windows CP437 insert

NLS_LANG in Windows CMD has been set to match codepage
Conversion happens here both on insert and query

```
C:\>chcp
Active code page: 437

C:\>set NLS_LANG=american_america.us8pc437

insert into movies values (2, 'Jesús, nuestro Señor', 'Win 437 US8PC437');

INSERTED_BY          TITLE                    BYTES CHARS TITLE_DUMP
-------------------- ------------------------ ----- ----- --------------------------------
Lin UTF-8 AL32UTF8   Jesús, nuestro Señor        22    20 Typ=1 Len=22: 74,101,115,195,186
                                                          ,115,44,32,110,117,101,115,116,1
                                                          14,111,32,83,101,195,177,111,114

Win 437 US8PC437     Jesús, nuestro Señor        22    20 Typ=1 Len=22: 74,101,115,195,186
                                                          ,115,44,32,110,117,101,115,116,1
                                                          14,111,32,83,101,195,177,111,114
```

Linux without NLS_LANG

Not using NLS_LANG gives problems both on insert and query
Even the one inserted in this session displays wrongly

```
[oracle@vbgeneric ~]$ unset NLS_LANG

insert into movies values (3, 'Jesús, nuestro Señor', 'Lin UTF-8 {unset}');

INSERTED_BY          TITLE                   BYTES CHARS TITLE_DUMP
-------------------- ----------------------- ----- ----- --------------------------------
Lin UTF-8 AL32UTF8   Jesus, nuestro Se?or       22    20 Typ=1 Len=22: 74,101,115,195,186
                                                         ,115,44,32,110,117,101,115,116,1
                                                         14,111,32,83,101,195,177,111,114
Win 437 US8PC437     Jesus, nuestro Se?or       22    20 Typ=1 Len=22: 74,101,115,195,186
                                                         ,115,44,32,110,117,101,115,116,1
                                                         14,111,32,83,101,195,177,111,114
Lin UTF-8 {unset}    Jes??s, nuestro Se??or     30    22 Typ=1 Len=30: 74,101,115,239,191
                                                         ,189,239,191,189,115,44,32,110,1
                                                         17,101,115,116,114,111,32,83,101
                                                         ,239,191,189,239,191,189,111,114
```

Linux without NLS_LANG

Cause: Without the NLS_LANG, it defaults to AMERICAN_AMERICA.US7ASCII

```
select sys_context('USERENV', 'LANGUAGE') as
language from dual;

LANGUAGE
----------------------------------------
AMERICAN_AMERICA.AL32UTF8

select sci.client_charset
from v$session_connect_info sci
where sci.sid =
         sys_context('USERENV', 'SID')
and network_service_banner like 'TCP%';

CLIENT_CHARSET
----------------------------------------
US7ASCII
```

Database session uses database character set = AL32UTF8

Client connection without NLS_LANG guesses character set = US7ASCII

INSERT: OS sends bytes in UTF8 but DB interprets as 7-bit ASCII and converts

SELECT: DB converts UTF8 to 7-bit ASCII - unconvertible chars become ?

Linux with wrong NLS_LANG

Setting OS locale to Danish with ISO-8859-1 characterset (remember terminal setting)
But setting NLS_LANG to AL32UTF8 - connection CLIENT_CHARSET believes it

```
[oracle@vbgeneric bin]$ export LANG=da_DK.iso88591
[oracle@vbgeneric bin]$ locale | grep LANG
LANG=da_DK.iso88591

[oracle@vbgeneric bin]$ export NLS_LANG=american_america.al32utf8


select sci.client_charset
from v$session_connect_info sci
where sci.sid = sys_context('USERENV', 'SID')
and network_service_banner like 'TCP%';

CLIENT_CHARSET
----------------------------------------
AL32UTF8
```

Linux with wrong NLS_LANG

DB believes client charset = DB charset => no conversion is taking place
ISO-8859-1 byte values are interpreted as UTF-8, which can lead to unexpected errors

```
insert into movies values (4, 'Jesús, nuestro Señor', 'Lin ISO8859 AL32UTF8');

ERROR:
ORA-01756: quoted string not properly terminated
```

ñ is decimal 241 or binary 11110001. UTF-8 defines chars begin 11110xxx is 4 byte char.
DB believes **ñor'** is single 4-byte UTF-8 char, so it thinks terminating ' is missing.
Adding a couple extra characters makes the string terminated so a row is created.

```
insert into movies values (5, 'Jesús, nuestro Señores', 'Lin ISO8859 AL32UTF8');

1 row created.
```

Linux with wrong NLS_LANG

Querying also shows no conversion takes place - UTF-8 bytes are interpreted as ISO
The last line was inserted in this session and displays "correctly" (but is wrong!)

```
INSERTED_BY          TITLE                   BYTES CHARS TITLE_DUMP
-------------------- ----------------------- ----- ----- --------------------------------
Lin UTF-8 AL32UTF8   JesÃºs, nuestro SeÃ±or      22    20 Typ=1 Len=22: 74,101,115,195,186
                                                         ,115,44,32,110,117,101,115,116,1
                                                         14,111,32,83,101,195,177,111,114
Win 437 US8PC437     JesÃºs, nuestro SeÃ±or      22    20 Typ=1 Len=22: 74,101,115,195,186
                                                         ,115,44,32,110,117,101,115,116,1
                                                         14,111,32,83,101,195,177,111,114
Lin UTF-8 {unset}    Jesï¿½ï¿½s, nuestro Seï¿½ï¿½or      30    22 Typ=1 Len=30: 74,101,115,239,191
                                                         ,189,239,191,189,115,44,32,110,1
                                                         17,101,115,116,114,111,32,83,101
                                                         ,239,191,189,239,191,189,111,114
Lin ISO8859 AL32UTF8 Jesús, nuestro Señores     22    19 Typ=1 Len=22: 74,101,115,250,115
                                                         ,44,32,110,117,101,115,116,114,1
                                                         11,32,83,101,241,111,114,101,115
```

Linux with wrong NLS_LANG

Set the locale and NLS_LANG back to correct values

```
[oracle@vbgeneric bin]$ export LANG=en_US.UTF-8
[oracle@vbgeneric bin]$ locale | grep LANG
LANG=en_US.UTF-8

[oracle@vbgeneric bin]$ export NLS_LANG=american_america.al32utf8


select sci.client_charset
from v$session_connect_info sci
where sci.sid = sys_context('USERENV', 'SID')
and network_service_banner like 'TCP%';

CLIENT_CHARSET
----------------------------------------
AL32UTF8
```

Linux with wrong NLS_LANG

The � characters are correct UTF created by database at conversion
The ▒ characters are invalid - there's now corrupt text in the database

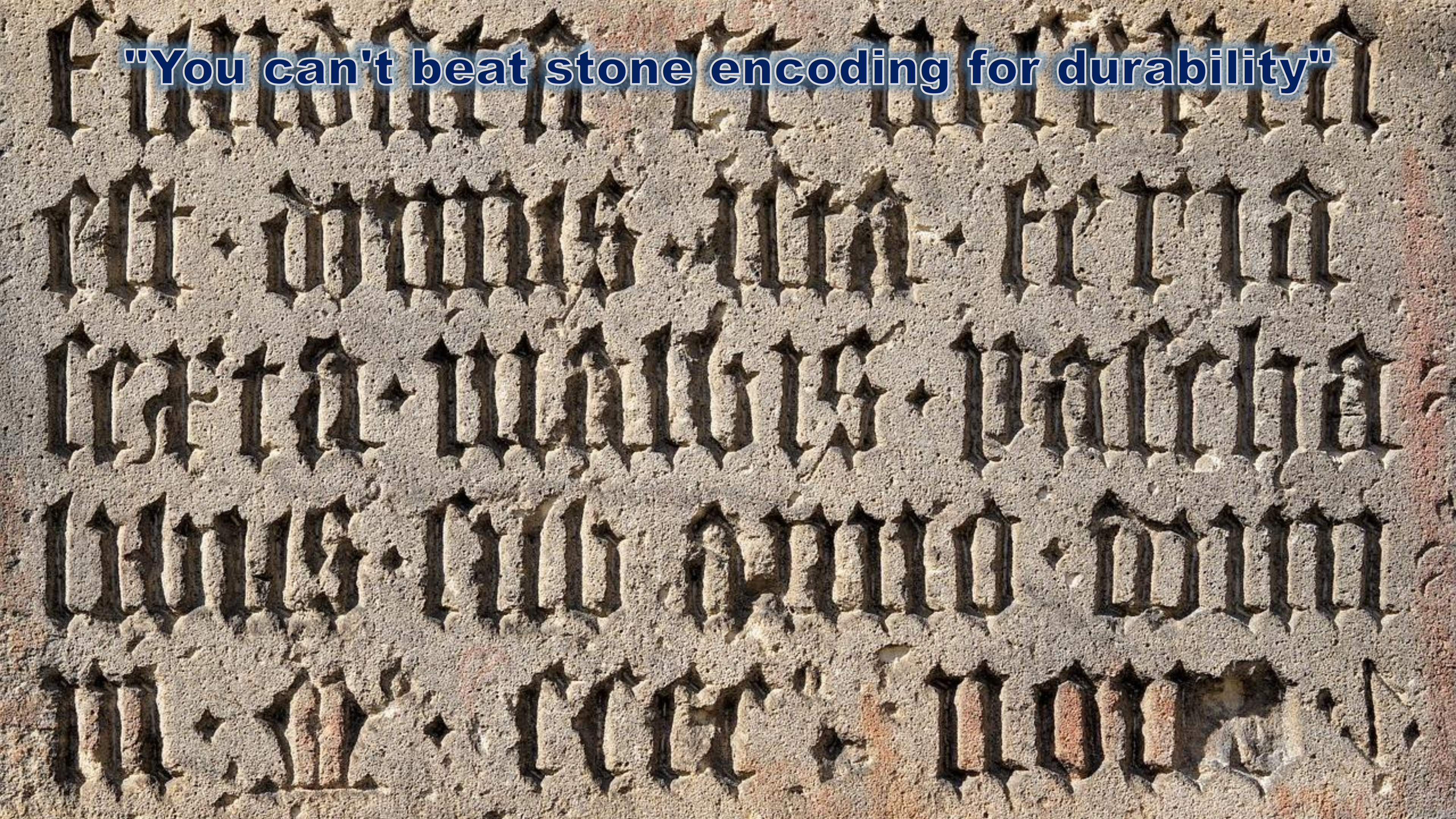
```
INSERTED_BY          TITLE                    BYTES CHARS TITLE_DUMP
-------------------- ------------------------ ----- ----- --------------------------------
Lin UTF-8 AL32UTF8   Jesús, nuestro Señor        22    20 Typ=1 Len=22: 74,101,115,195,186
                                                          ,115,44,32,110,117,101,115,116,1
                                                          14,111,32,83,101,195,177,111,114
Win 437 US8PC437     Jesús, nuestro Señor        22    20 Typ=1 Len=22: 74,101,115,195,186
                                                          ,115,44,32,110,117,101,115,116,1
                                                          14,111,32,83,101,195,177,111,114
Lin UTF-8 {unset}    Jes��s, nuestro Se��or        30    22 Typ=1 Len=30: 74,101,115,239,191
                                                          ,189,239,191,189,115,44,32,110,1
                                                          17,101,115,116,114,111,32,83,101
                                                          ,239,191,189,239,191,189,111,114
Lin ISO8859 AL32UTF8 Jes▒s, nuestro Se▒ores        22    19 Typ=1 Len=22: 74,101,115,250,115
                                                          ,44,32,110,117,101,115,116,114,1
                                                          11,32,83,101,241,111,114,101,115
```

Lesson

- Client NLS_LANG characterset is important
  - If it matches the database charset, no conversion takes place!
  - If it does not match DB charset, conversion is attempted in "best effort" manner
- Client NLS_LANG characterset **should match** the **OS locale / codepage** (or client setting if the client program allows different codepage than OS)
  - If charset OS <> NLS_LANG = DB, wrong bytes are **not** converted!
  - If charset OS <> NLS_LANG <> DB, conversion happens to/from wrong charset

The ONE thing you MUST learn!

Character sets and encodings



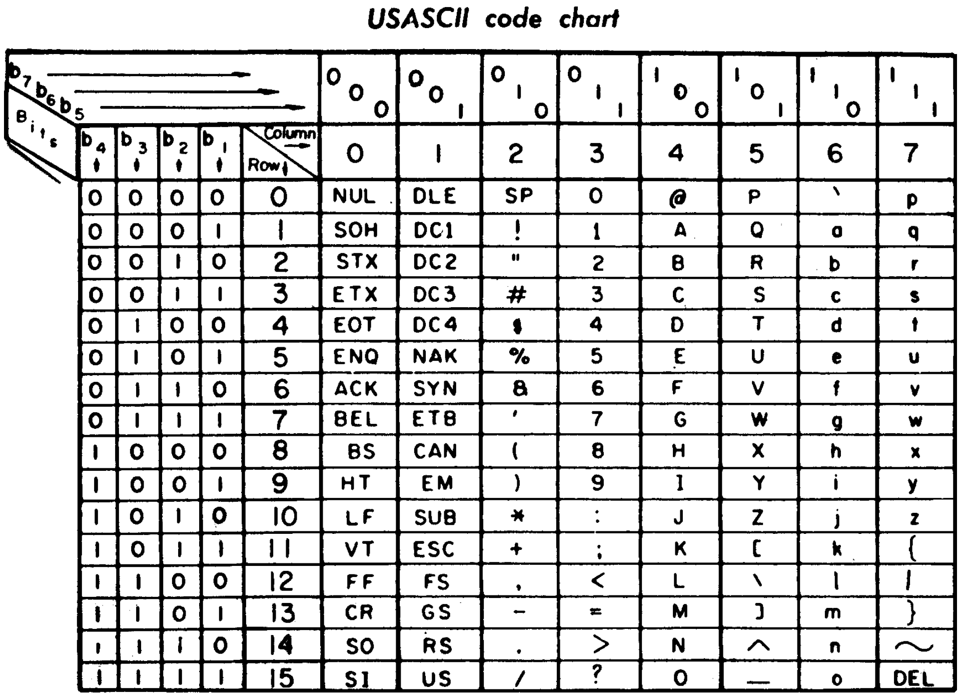
"You can't beat stone encoding for durability"

est · domus · ista · feria
sexta · in albis · pascha
libus · sub · anno · domini

Encoding

- How to store / transmit letters?
  - Visual: write an "SOS" on a piece of paper / send the letter with the postman
  - Audio: record saying "SOS" on tape / say "SOS" on the phone
  - Signal: send "··· ––– ···" with radio or flashlight
  - Digital: store / send 21 bits "`101001110011111010011`"
  - Digital: store / send 24 bits "`010100110100111101010011`"
- Which to choose?
  - You have to agree with recipient

7-bit ASCII (00 to 7F)

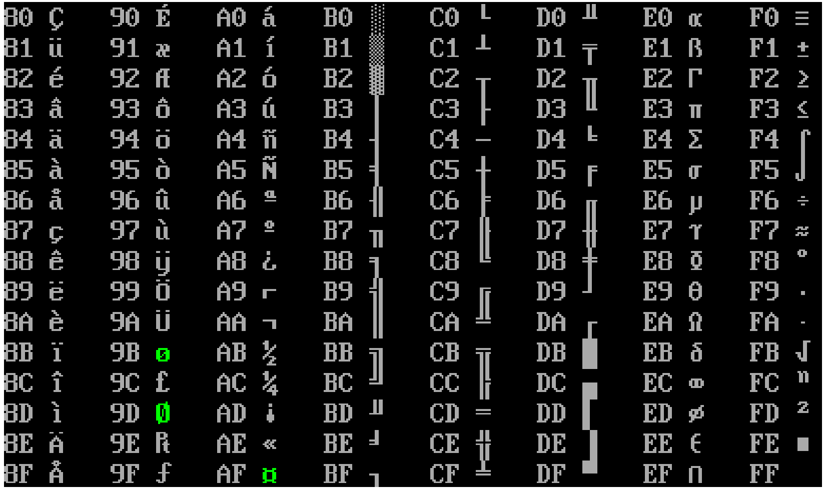


USASCII code chart

Bits b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

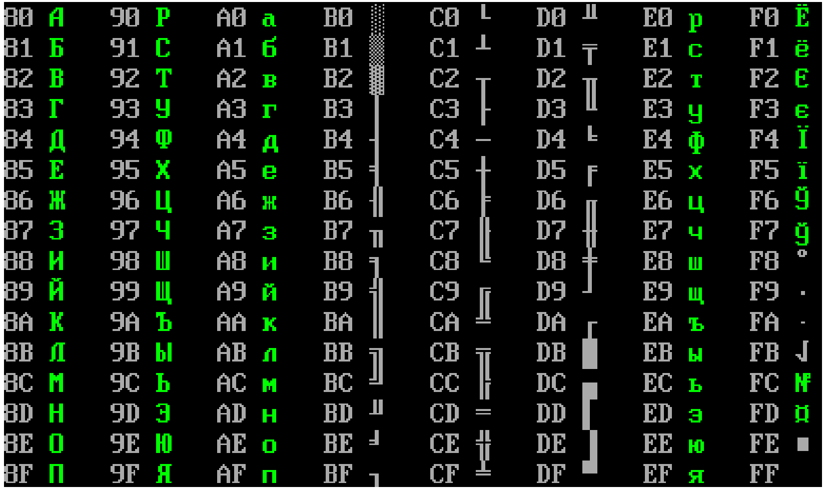
8th bit (80 to FF) - ASCII codepages

Codepage 865 Nordic



```
80 Ç   90 É   A0 á   B0 ░   C0 └   D0 ╨   E0 α   F0 ≡
81 ü   91 æ   A1 í   B1 ▒   C1 ┴   D1 ╤   E1 ß   F1 ±
82 é   92 Æ   A2 ó   B2 ▓   C2 ┬   D2 ╥   E2 Γ   F2 ≥
83 â   93 ô   A3 ú   B3 │   C3 ├   D3 ╙   E3 π   F3 ≤
84 ä   94 ö   A4 ñ   B4 ┤   C4 ─   D4 ╘   E4 Σ   F4 ⌠
85 à   95 ò   A5 Ñ   B5 ╡   C5 ┼   D5 ╒   E5 σ   F5 ⌡
86 å   96 û   A6 ª   B6 ╢   C6 ╞   D6 ╓   E6 µ   F6 ÷
87 ç   97 ù   A7 º   B7 ╖   C7 ╟   D7 ╫   E7 τ   F7 ≈
88 ê   98 ÿ   A8 ¿   B8 ╕   C8 ╚   D8 ╪   E8 Φ   F8 °
89 ë   99 Ö   A9 ⌐   B9 ╣   C9 ╔   D9 ┘   E9 Θ   F9 ∙
8A è   9A Ü   AA ¬   BA ║   CA ╩   DA ┌   EA Ω   FA ·
8B ï   9B ø   AB ½   BB ╗   CB ╦   DB █   EB δ   FB √
8C î   9C £   AC ¼   BC ╝   CC ╠   DC ▄   EC ∞   FC ⁿ
8D ì   9D Ø   AD ¡   BD ╜   CD ═   DD ▌   ED φ   FD ²
8E Ä   9E ₧   AE «   BE ╛   CE ╬   DE ▐   EE ε   FE ■
8F Å   9F ƒ   AF ¤   BF ┐   CF ╧   DF ▀   EF ∩   FF
```

Codepage 866 Russian (Cyrillic II)

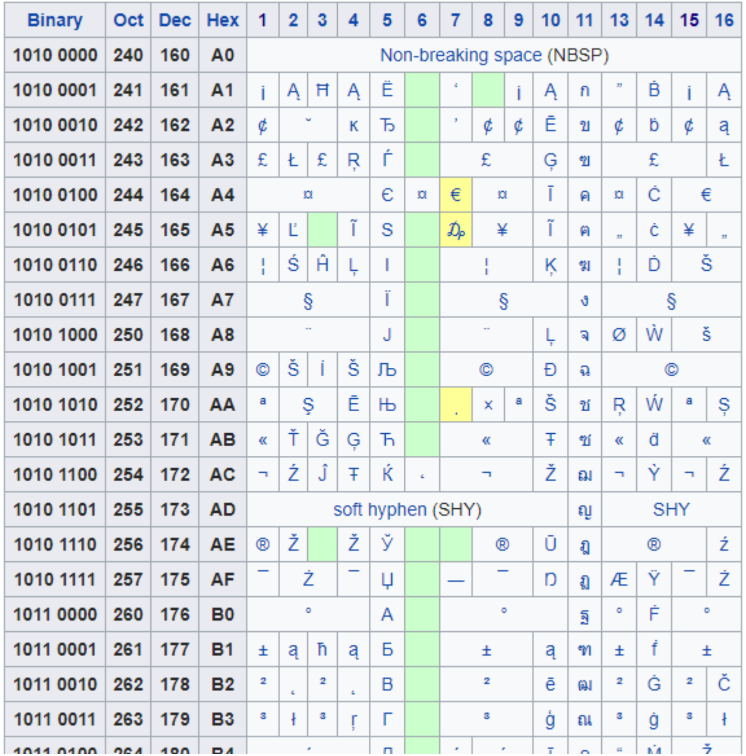


```
80 А   90 Р   A0 а   B0 ░   C0 └   D0 ╨   E0 р   F0 Ё
81 Б   91 С   A1 б   B1 ▒   C1 ┴   D1 ╤   E1 с   F1 ё
82 В   92 Т   A2 в   B2 ▓   C2 ┬   D2 ╥   E2 т   F2 Є
83 Г   93 У   A3 г   B3 │   C3 ├   D3 ╙   E3 у   F3 є
84 Д   94 Ф   A4 д   B4 ┤   C4 ─   D4 ╘   E4 ф   F4 Ї
85 Е   95 Х   A5 е   B5 ╡   C5 ┼   D5 ╒   E5 х   F5 ї
86 Ж   96 Ц   A6 ж   B6 ╢   C6 ╞   D6 ╓   E6 ц   F6 Ў
87 З   97 Ч   A7 з   B7 ╖   C7 ╟   D7 ╫   E7 ч   F7 ў
88 И   98 Ш   A8 и   B8 ╕   C8 ╚   D8 ╪   E8 ш   F8 °
89 Й   99 Щ   A9 й   B9 ╣   C9 ╔   D9 ┘   E9 щ   F9 ∙
8A К   9A Ъ   AA к   BA ║   CA ╩   DA ┌   EA ъ   FA ·
8B Л   9B Ы   AB л   BB ╗   CB ╦   DB █   EB ы   FB √
8C М   9C Ь   AC м   BC ╝   CC ╠   DC ▄   EC ь   FC №
8D Н   9D Э   AD н   BD ╜   CD ═   DD ▌   ED э   FD ¤
8E О   9E Ю   AE о   BE ╛   CE ╬   DE ▐   EE ю   FE ■
8F П   9F Я   AF п   BF ┐   CF ╧   DF ▀   EF я   FF
```

Green signifies differences from Codepage 437 United States (original IBM "PC-ASCII" codepage)

ISO-8859

- International standard codepages
- 16 different (both latin and other alphabets)
- Hex 80 to 9F unassigned
(intended for control chars like 00 to 1F)
- ISO-8859-1 (Latin-1) very popular for webpages (before UTF-8)
- Versions ("parts") made sometimes for small changes, like ISO-8859-15 for €



Comparison of the various parts (1–16) of ISO/IEC 8859

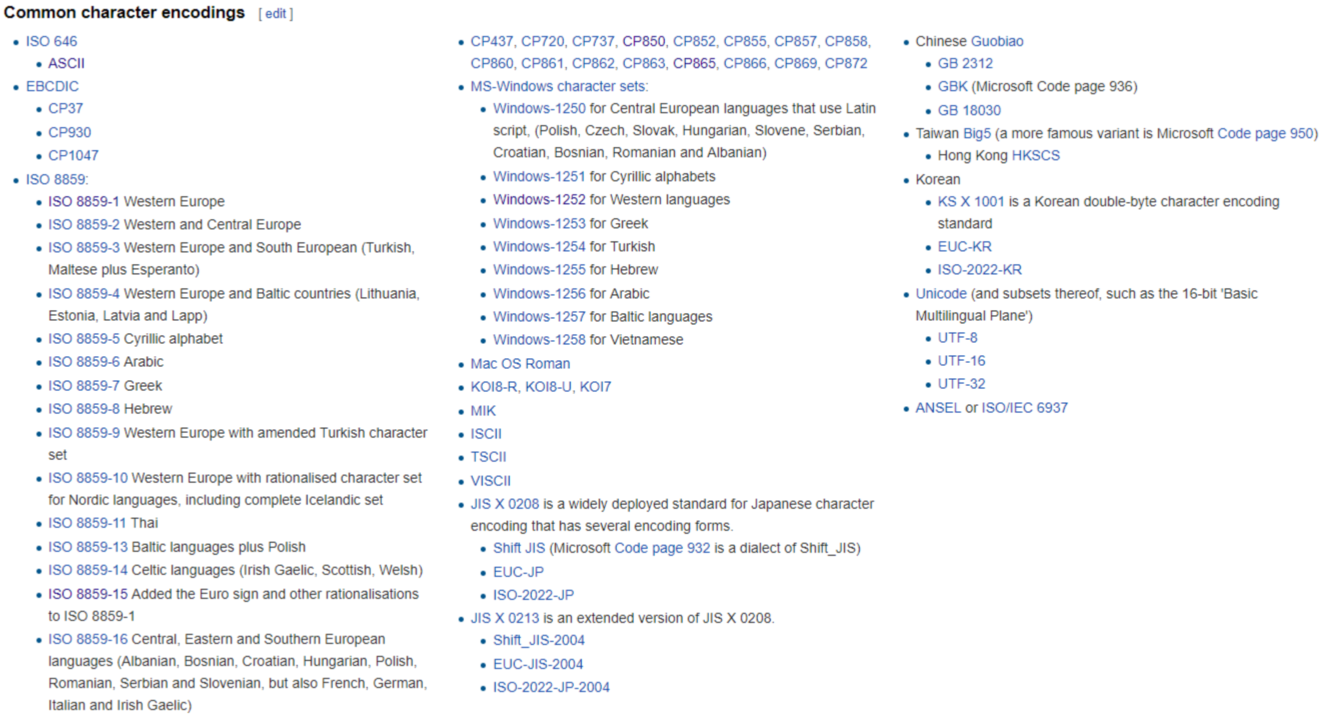
Binary	Oct	Dec	Hex	1	2	3	4	5	6	7	8	9	10	11	13	14	15	16
1010 0000	240	160	A0	Non-breaking space (NBSP)														
1010 0001	241	161	A1	¡	Ą	Ħ	Ą	Ё		‘		¡	Ą	ก	”	Ḃ	¡	Ą
1010 0010	242	162	A2	¢	˘		ĸ	Ђ		’	¢	¢	Ē	ข	¢	ḃ	¢	ą
1010 0011	243	163	A3	£	Ł	£	Ŗ	Ѓ		£			Ģ	ฃ	£			Ł
1010 0100	244	164	A4	¤				Є	¤	€	¤		Ī	ค	¤	Ċ	€	
1010 0101	245	165	A5	¥	Ľ		Ĩ	Ѕ		₯	¥		Ĩ	ฅ	„	ċ	¥	„
1010 0110	246	166	A6	¦	Ś	Ĥ	Ļ	І		¦			Ķ	ฆ	¦	Ḋ	Š	
1010 0111	247	167	A7	§				Ї		§				ง	§			
1010 1000	250	168	A8	¨				Ј		¨			Ļ	จ	Ø	Ẁ	š	
1010 1001	251	169	A9	©	Š	İ	Š	Љ		©			Đ	ฉ	©			
1010 1010	252	170	AA	ª	Ş		Ē	Њ		ͺ	×	ª	Š	ช	Ŗ	Ẃ	ª	Ș
1010 1011	253	171	AB	«	Ť	Ğ	Ģ	Ћ		«			Ŧ	ซ	«	ḋ	«	
1010 1100	254	172	AC	¬	Ź	Ĵ	Ŧ	Ќ	،	¬			Ž	ฌ	¬	Ỳ	¬	Ź
1010 1101	255	173	AD	soft hyphen (SHY)										ญ	SHY			
1010 1110	256	174	AE	®	Ž		Ž	Ў			®		Ū	ฎ	®			ź
1010 1111	257	175	AF	¯	Ż		¯	Џ		―	¯		Ŋ	ฏ	Æ	Ÿ	¯	Ż
1011 0000	260	176	B0	°				А		°				ฐ	°	Ḟ	°	
1011 0001	261	177	B1	±	ą	ħ	ą	Б		±			ą	ฑ	±	ḟ	±	
1011 0010	262	178	B2	²	˛	²	˛	В		²			ē	ฒ	²	Ġ	²	Č
1011 0011	263	179	B3	³	ł	³	ŗ	Г		³			ģ	ณ	³	ġ	³	ł
1011 0100	264	180	B4	´				Д		΄	´		ī	ด	“	Ṁ	Ž	

MS-Windows (ANSI) codepages

- Win-1252 standard English Windows
- Win-1252 originally based on ISO-8859 draft
- But is a "superset" with printable instead of control characters in the 80 to 9F range
  - Smart quotes (“”), ellipsis (…), other typographical characters
- Win-1252 webpages often mislabelled as ISO-8859-1
  - Non-windows clients would display wrongly

- 874 – Windows Thai
- 1250 – Windows Central Europe
- 1251 – Windows Cyrillic
- 1252 – Windows Western
- 1253 – Windows Greek
- 1254 – Windows Turkish
- 1255 – Windows Hebrew
- 1256 – Windows Arabic
- 1257 – Windows Baltic
- 1258 – Windows Vietnamese

Too many encodings



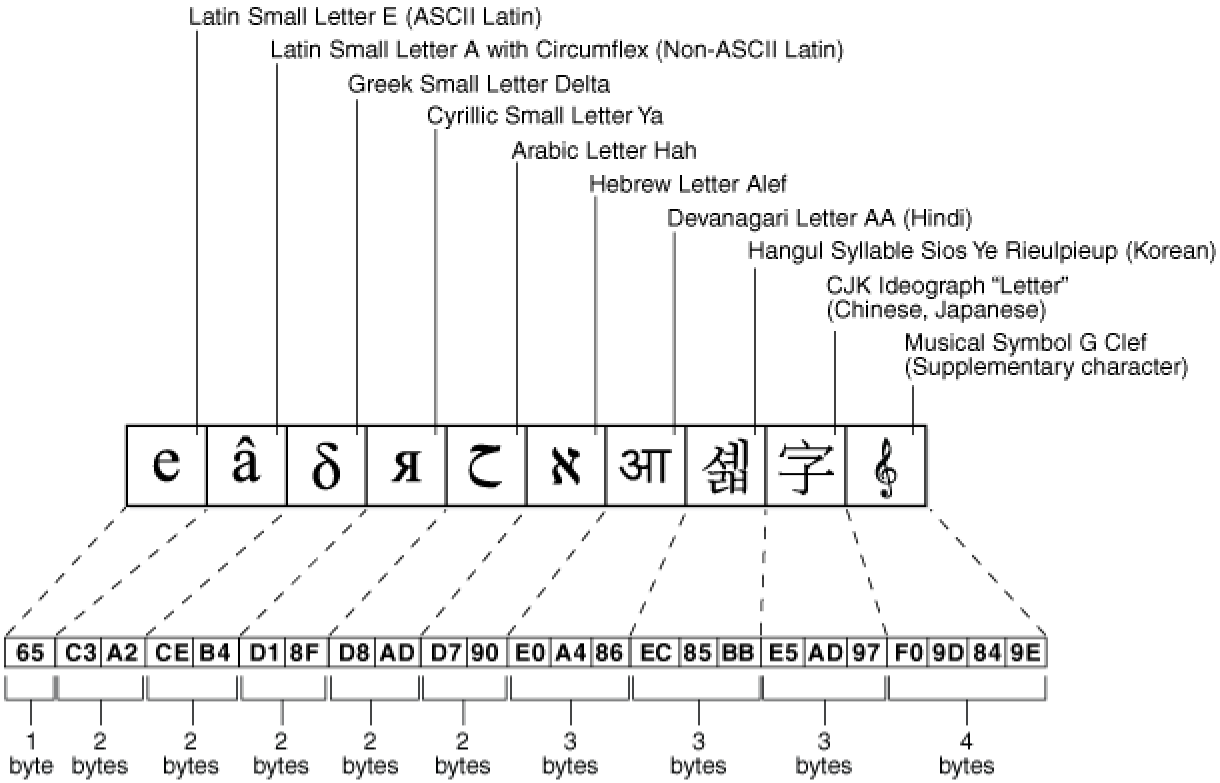
Common character encodings [edit]

- ISO 646
  - ASCII
- EBCDIC
  - CP37
  - CP930
  - CP1047
- ISO 8859:
  - ISO 8859-1 Western Europe
  - ISO 8859-2 Western and Central Europe
  - ISO 8859-3 Western Europe and South European (Turkish, Maltese plus Esperanto)
  - ISO 8859-4 Western Europe and Baltic countries (Lithuania, Estonia, Latvia and Lapp)
  - ISO 8859-5 Cyrillic alphabet
  - ISO 8859-6 Arabic
  - ISO 8859-7 Greek
  - ISO 8859-8 Hebrew
  - ISO 8859-9 Western Europe with amended Turkish character set
  - ISO 8859-10 Western Europe with rationalised character set for Nordic languages, including complete Icelandic set
  - ISO 8859-11 Thai
  - ISO 8859-13 Baltic languages plus Polish
  - ISO 8859-14 Celtic languages (Irish Gaelic, Scottish, Welsh)
  - ISO 8859-15 Added the Euro sign and other rationalisations to ISO 8859-1
  - ISO 8859-16 Central, Eastern and Southern European languages (Albanian, Bosnian, Croatian, Hungarian, Polish, Romanian, Serbian and Slovenian, but also French, German, Italian and Irish Gaelic)
- CP437, CP720, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872
- MS-Windows character sets:
  - Windows-1250 for Central European languages that use Latin script, (Polish, Czech, Slovak, Hungarian, Slovene, Serbian, Croatian, Bosnian, Romanian and Albanian)
  - Windows-1251 for Cyrillic alphabets
  - Windows-1252 for Western languages
  - Windows-1253 for Greek
  - Windows-1254 for Turkish
  - Windows-1255 for Hebrew
  - Windows-1256 for Arabic
  - Windows-1257 for Baltic languages
  - Windows-1258 for Vietnamese
- Mac OS Roman
- KOI8-R, KOI8-U, KOI7
- MIK
- ISCII
- TSCII
- VISCII
- JIS X 0208 is a widely deployed standard for Japanese character encoding that has several encoding forms.
  - Shift JIS (Microsoft Code page 932 is a dialect of Shift_JIS)
  - EUC-JP
  - ISO-2022-JP
- JIS X 0213 is an extended version of JIS X 0208.
  - Shift_JIS-2004
  - EUC-JIS-2004
  - ISO-2022-JP-2004
- Chinese Guobiao
  - GB 2312
  - GBK (Microsoft Code page 936)
  - GB 18030
- Taiwan Big5 (a more famous variant is Microsoft Code page 950)
  - Hong Kong HKSCS
- Korean
  - KS X 1001 is a Korean double-byte character encoding standard
  - EUC-KR
  - ISO-2022-KR
- Unicode (and subsets thereof, such as the 16-bit 'Basic Multilingual Plane')
  - UTF-8
  - UTF-16
  - UTF-32
- ANSEL or ISO/IEC 6937

Unicode to the rescue?

- Encode practically any characters - alphabets (latin, chinese, arabic, etc.), formulas, symbols, emoji, abstract characters, ...
- Characters assigned a code point - code points encoded in:
  - UTF-8 - variable length 1-4 bytes, 00 to 7F (7-bit ASCII) in single byte, others in 2, 3 or 4 bytes with most common chars having lowest byte count
  - UCS-2 - fixed length 2 bytes, cannot encode all Unicode
  - UTF-16 - variable length 2 or 4 bytes, replacement for UCS-2
  - UTF-32 - fixed length 4 bytes
- UTF-8 now most popular encoding for webpages
- Still not "single encoding everywhere" - developers still need to care about it!

UTF-8 / AL32UTF8 bytes per character


Latin Small Letter E (ASCII Latin)
Latin Small Letter A with Circumflex (Non-ASCII Latin)
Greek Small Letter Delta
Cyrillic Small Letter Ya
Arabic Letter Hah
Hebrew Letter Alef
Devanagari Letter AA (Hindi)
Hangul Syllable Sios Ye Rieulpieup (Korean)
CJK Ideograph "Letter" (Chinese, Japanese)
Musical Symbol G Clef (Supplementary character)
e
â
δ
я
ح
א
आ
섋
字
𝄞
65
C3 A2
CE B4
D1 8F
D8 AD
D7 90
E0 A4 86
EC 85 BB
E5 AD 97
F0 9D 84 9E
1 byte
2 bytes
2 bytes
2 bytes
2 bytes
2 bytes
3 bytes
3 bytes
3 bytes
4 bytes

Database character set / National character set

"I'm a tibetan - I'm being polite!"

Database character set

- Chosen at database creation time
- Normally not possible to change (except from singlebyte to AL32UTF8 with DMU)
- Used for
  - Data in CHAR, VARCHAR2, CLOB, LONG
  - Identifiers (object names) *
  - Source code (SQL and PL/SQL) *
  - *) Not all objects/identifiers may use multibyte characters
    (I would not recommend using multibyte other than in data)
- From 12.2 the default at installation is AL32UTF8

National character set

- Used for multibyte data if the database character set is singlebyte
- Can be UTF8 or AL16UTF16 - default is AL16UTF16
- Used for data in NCHAR, NVARCHAR2, NCLOB
- If database character set is multibyte, national character set is not really needed (except possibly if support for UTF-8 as well as UTF-16 is needed)

Plug-compatibility

- From version 12.2, PDBs of different DB charsets can be in some circumstances be plugged into the same multitenant CDB
- If CDB is AL32UTF8, any PDB can be plugged in
- Otherwise PDB charset must be a subset of CDB charset (plug-compatible) then the plugged-in PDB is changed to the CDB charset
  - For example WE8ISO8859P1 is a subset of WE8MSWIN1252

BYTE versus CHAR length semantics

"The game of Spot-The-Multi-Byte..."

```
s= tyJCtd mGzaTb baZpAe"><h1 id="h.P_OdGDwrTPnOT6"
extures/home/floors/paving-stones">Paving stones</a
div><footer class="yxgWrb"><section id="h.s_eBis0C
iv class="LS81yb VICjCf" tabindex="-1"><div class="
div id="h.s_9TDSuGgilpFy" class="hJDwNd-AhqUyc-wN
Ayb-c4YZDc"><div class="tyJCtd mGzaTb baZpAe"><sma
 </strong>❤<strong> by StruffelProductions (2017
lock;"><a class="dhtgD" href="https://www.youtube.
.google.com/url?
ww.artstation.com%2Fstruffelproductions
```

Length semantics

- Length is not just length
  - `'Señor'` is 5 chars as well as 5 bytes in WE8ISO8859P1
  - `'Señor'` is 5 chars but 6 bytes in AL32UTF8
- Max length of a VARCHAR2 column can be specified in bytes or in chars
- If no indication is given whether the length is in bytes or in chars,
  the parameter NLS_LENGTH_SEMANTICS is used (values BYTE or CHAR)
- Recommended only to set NLS_LENGTH_SEMANTICS=CHAR on session basis
- When max storage limit of VARCHAR2 (4000B or 32K) is used,
  even if length is specified in chars, only 4000B / 32K **bytes** can be stored

Define semantics directly

Define whether column length is specified in bytes or characters
Parameter NLS_LENGTH_SEMANTICS is used if you do not specify in DDL

```
create table movies (
   title   varchar2(100 BYTE)
);


create table movies (
   title   varchar2(100 CHAR)
);


create table movies (
   title   varchar2(100)
);
```

```
-- Up to 100 bytes regardless of DB charset
-- If multibyte characters, only "whole" chars
-- are stored - i.e. not "half" a char


-- Up to 100 characters regardless of charset
-- Could use f.ex. 400 bytes if storing 100
-- 4-byte characters in UTF-8


-- Will be created using the session value of
-- NLS_LENGTH_SEMANTICS (BYTE or CHAR) at time
-- of table creation
```

Byte limit of columns (or PL/SQL variables)

Even when specifying 4000 CHAR, the column will only store up to 4000 bytes
(if using MAX_STRING_SIZE = EXTENDED, then limit is 32KB)

```
create table movies (
   title   varchar2(4000 BYTE)
);


create table movies (
   title   varchar2(4000 CHAR)
);
```

```

-- Up to 4000 bytes




-- Also up to 4000 bytes, which might be 4000
-- single-byte chars or 1000 4-byte chars or
-- anything in between
```

Finding different lengths

LENGTH(string)	returns length measured in characters of input charset
LENGTHB(string)	returns length measured in bytes
LENGTHC(string)	returns length measured in Unicode complete characters
LENGTH2(string)	returns length measured in UCS2
LENGTH4(string)	returns length measured in USC4

NLS settings

"Can't y'awl just use good old US 7-bit ASCII!"

Viewing NLS parameter values

NLS_*_PARAMETERS shows values at database, instance and session level
V$NLS_PARAMETERS shows "current values" - almost equal to session level

```
select coalesce(dp.parameter, ip.parameter, sp.parameter, np.parameter) as parameter
     , nvl2(dp.parameter, dp.value, '{N/A}') as database_value
     , nvl2(ip.parameter, ip.value, '{N/A}') as instance_value
     , nvl2(sp.parameter, sp.value, '{N/A}') as session_value
     , nvl2(np.parameter, np.value, '{N/A}') as v$nls_value
  from nls_database_parameters dp
  full outer join nls_instance_parameters ip
      on ip.parameter = dp.parameter
  full outer join nls_session_parameters sp
      on sp.parameter = coalesce(dp.parameter, ip.parameter)
  full outer join v$nls_parameters np
      on np.parameter = coalesce(dp.parameter, ip.parameter, sp.parameter)
 order by parameter;
```

Results on my 12.2

trivadis

V$NLS like SESSION values + Charset parameters from DATABASE values

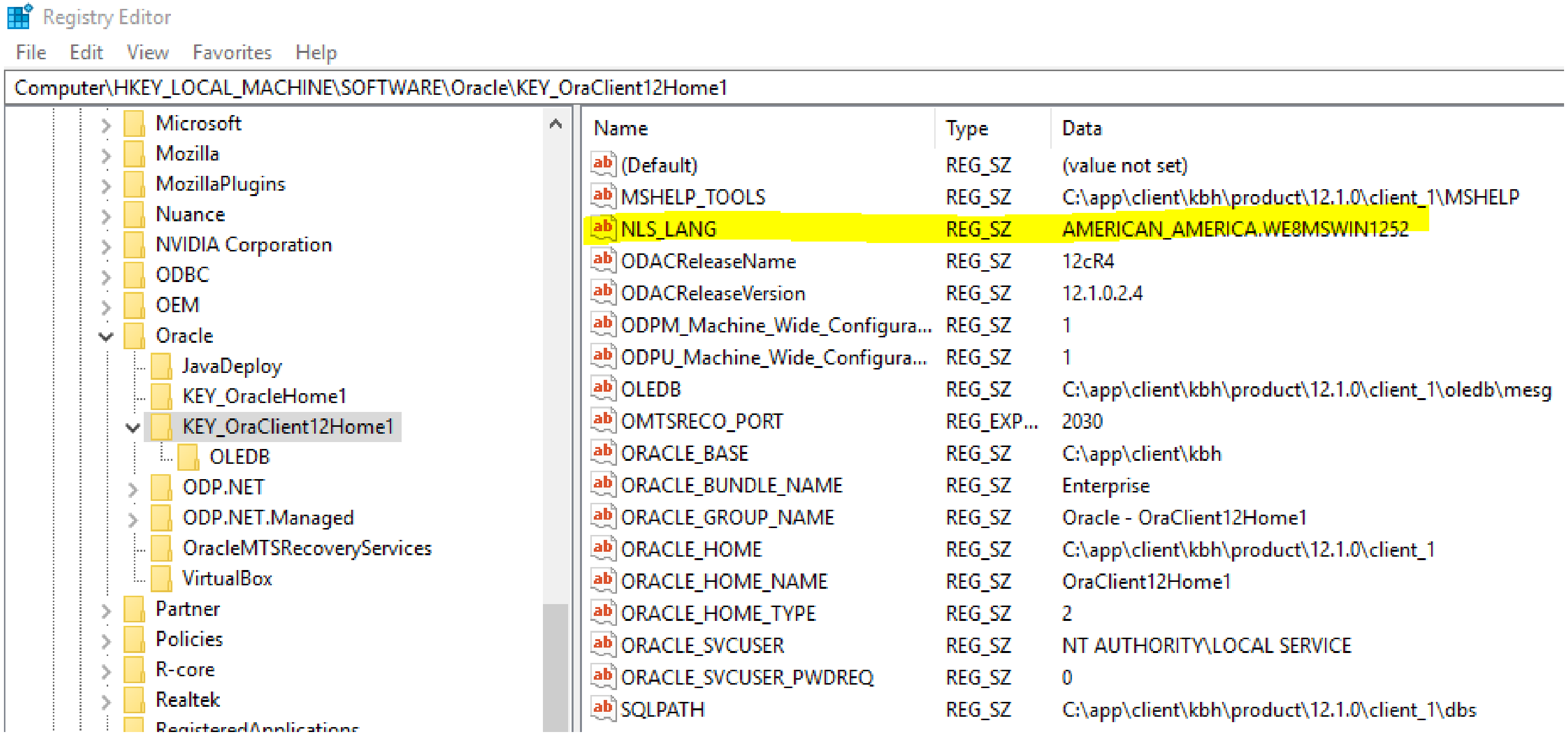
(2 NLS_TIME_* parameters are "currently used for internal purposes only")

```
PARAMETER                DATABASE_VALUE               INSTANCE_VALUE SESSION_VALUE                V$NLS_VALUE
------------------------ ---------------------------- -------------- ---------------------------- ----------------------------
NLS_CALENDAR             GREGORIAN                                   GREGORIAN                    GREGORIAN
NLS_CHARACTERSET         AL32UTF8                     {N/A}          {N/A}                        AL32UTF8
NLS_COMP                 BINARY                       BINARY         BINARY                       BINARY
NLS_CURRENCY             $                                           $                            $
NLS_DATE_FORMAT          DD-MON-RR                                   DD-MON-RR                    DD-MON-RR
NLS_DATE_LANGUAGE        AMERICAN                                    AMERICAN                     AMERICAN
NLS_DUAL_CURRENCY        $                                           $                            $
NLS_ISO_CURRENCY         AMERICA                                     AMERICA                      AMERICA
NLS_LANGUAGE             AMERICAN                     AMERICAN       AMERICAN                     AMERICAN
NLS_LENGTH_SEMANTICS     BYTE                         BYTE           BYTE                         BYTE
NLS_NCHAR_CHARACTERSET   AL16UTF16                    {N/A}          {N/A}                        AL16UTF16
NLS_NCHAR_CONV_EXCP      FALSE                        FALSE          FALSE                        FALSE
NLS_NUMERIC_CHARACTERS   .,                                          .,                           .,
NLS_RDBMS_VERSION        12.2.0.1.0                   {N/A}          {N/A}                        {N/A}
NLS_SORT                 BINARY                                      BINARY                       BINARY
NLS_TERRITORY            AMERICA                      AMERICA        AMERICA                      AMERICA
NLS_TIMESTAMP_FORMAT     DD-MON-RR HH.MI.SSXFF AM                    DD-MON-RR HH.MI.SSXFF AM     DD-MON-RR HH.MI.SSXFF AM
NLS_TIMESTAMP_TZ_FORMAT  DD-MON-RR HH.MI.SSXFF AM TZR                DD-MON-RR HH.MI.SSXFF AM TZR DD-MON-RR HH.MI.SSXFF AM TZR
NLS_TIME_FORMAT          HH.MI.SSXFF AM                              HH.MI.SSXFF AM               HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT       HH.MI.SSXFF AM TZR                          HH.MI.SSXFF AM TZR           HH.MI.SSXFF AM TZR
```

Session NLS settings

- On connection defaults to values derived from NLS_LANG registry entry or NLS_LANG environment variable if such exists
- NLS_LANG in format <language>_<territory>.<charset>
  - `american_america.utf8`
  - `danish_denmark.we8iso8859p1`
- Many settings like currency, calendar, datetime formats, numeric characters, etc. get derived values from the territory
- Most settings can then be overruled with ALTER SESSION commands
  - `alter session set nls_date_format = 'YYYY-MM-DD';`
- Different client programs might choose to use NLS_LANG or do ALTER SESSION

Windows registry



Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Oracle\KEY_OraClient12Home1

- Microsoft
- Mozilla
- MozillaPlugins
- Nuance
- NVIDIA Corporation
- ODBC
- OEM
- Oracle
  - JavaDeploy
  - KEY_OracleHome1
  - KEY_OraClient12Home1
    - OLEDB
  - ODP.NET
  - ODP.NET.Managed
  - OracleMTSRecoveryServices
  - VirtualBox
- Partner
- Policies
- R-core
- Realtek
- RegisteredApplications

Name	Type	Data
(Default)	REG_SZ	(value not set)
MSHELP_TOOLS	REG_SZ	C:\app\client\kbh\product\12.1.0\client_1\MSHELP
NLS_LANG	REG_SZ	AMERICAN_AMERICA.WE8MSWIN1252
ODACReleaseName	REG_SZ	12cR4
ODACReleaseVersion	REG_SZ	12.1.0.2.4
ODPM_Machine_Wide_Configura...	REG_SZ	1
ODPU_Machine_Wide_Configura...	REG_SZ	1
OLEDB	REG_SZ	C:\app\client\kbh\product\12.1.0\client_1\oledb\mesg
OMTSRECO_PORT	REG_EXP...	2030
ORACLE_BASE	REG_SZ	C:\app\client\kbh
ORACLE_BUNDLE_NAME	REG_SZ	Enterprise
ORACLE_GROUP_NAME	REG_SZ	Oracle - OraClient12Home1
ORACLE_HOME	REG_SZ	C:\app\client\kbh\product\12.1.0\client_1
ORACLE_HOME_NAME	REG_SZ	OraClient12Home1
ORACLE_HOME_TYPE	REG_SZ	2
ORACLE_SVCUSER	REG_SZ	NT AUTHORITY\LOCAL SERVICE
ORACLE_SVCUSER_PWDREQ	REG_SZ	0
SQLPATH	REG_SZ	C:\app\client\kbh\product\12.1.0\client_1\dbs

Environment variable

Windows environment variable overrules registry
(Depends on client - JDBC / OCI based drivers should do so)

```
set NLS_LANG=american_america.us8pc437
```

Linux / unix environment variable

```
export NLS_LANG=danish_denmark.al32utf8
```

Valid NLS values

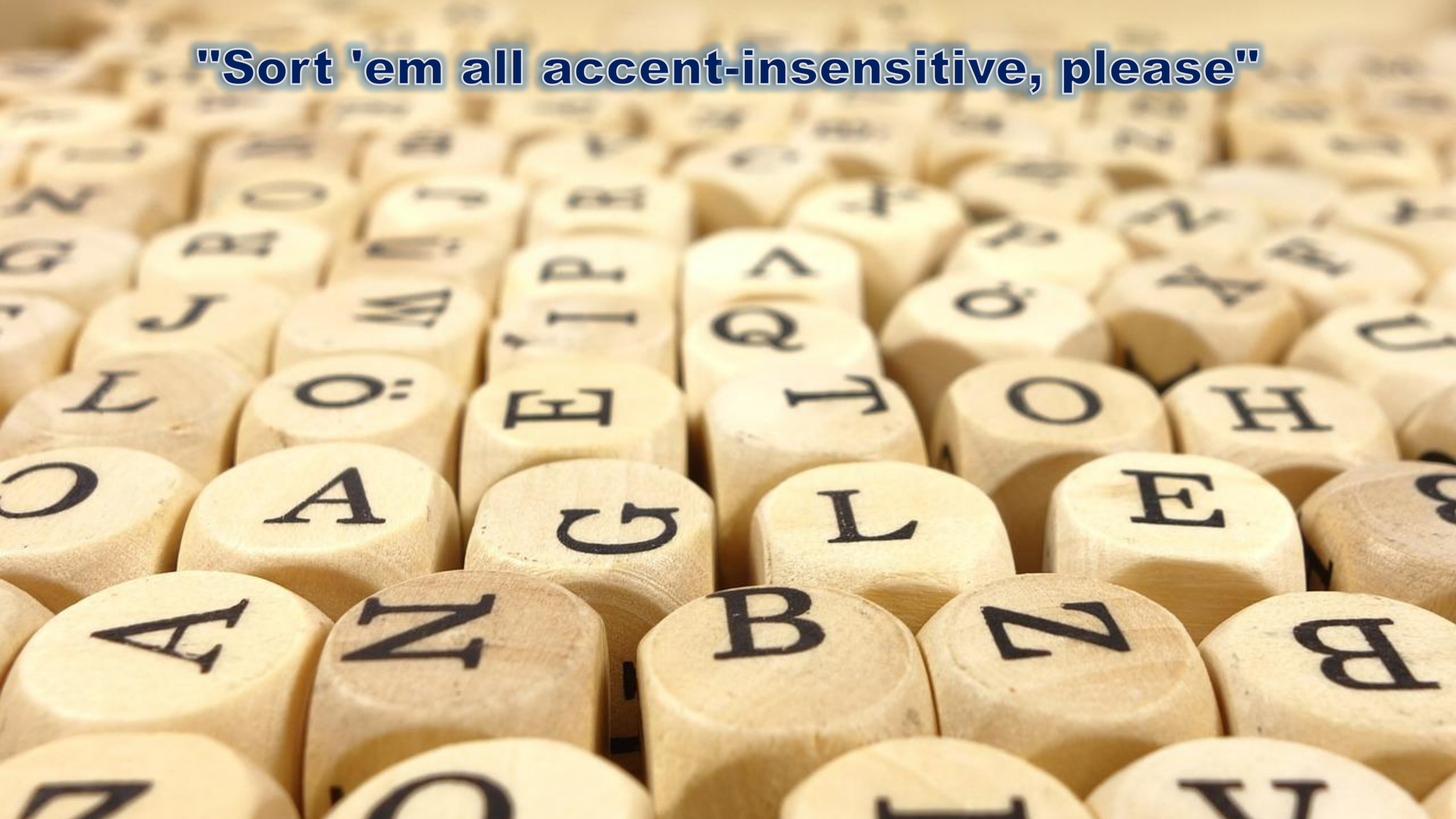
View V$NLS_VALID_VALUES show what values may be used
(If ISDEPRECATED='TRUE' then value should probably not be used)

```
select parameter, count(*) as value_cnt
    , count(nullif(isdeprecated,'TRUE'))
         as non_depr
  from v$nls_valid_values
 group by parameter
 order by parameter;

PARAMETER     VALUE_CNT   NON_DEPR
------------ ---------- ----------
CHARACTERSET        247        222
LANGUAGE             79         78
SORT                131        127
TERRITORY           130        125
```

```
select value, isdeprecated, con_id
  from v$nls_valid_values
 where parameter = 'CHARACTERSET'
 order by value;

VALUE            ISDEP CON_ID
---------------- ----- ------
AL16UTF16        FALSE      0
AL24UTFFSS       TRUE       0
AL32UTF8         FALSE      0
AR8ADOS710       FALSE      0
AR8ADOS710T      TRUE       0
AR8ADOS720       FALSE      0
AR8ADOS720T      TRUE       0
...
```

Linguistic sorting and matching

"Sort 'em all accent-insensitive, please"



NLS_SORT parameter

- Can be BINARY or a linguistic collation
- Defaults to derived value from NLS_LANGUAGE
- Makes ORDER BY sort by the specified collation

- NLS_SORT = {collation}_**CI** means Case Insensitive
  - Ignores case - a and A are considered identical
- NLS_SORT = {collation}_**AI** means Accent Insensitive
  - Ignores case + diacritics/accents - á, ä, a, Ä and A are considered identical

NLS_SORT parameter

- Linguistic collation ignores punctuation marks

- Problem at my previous work
  - Application with many VARCHAR2 keys - should be sorted binary
  - Application sets NLS_LANGUAGE -> unwanted sort of key columns
  - ALTER SESSION SET NLS_SORT=BINARY in AFTER LOGON trigger
  - Queries sorting on descriptive columns (non-keys) explicitly use NLSSORT()
  - (12.2 alternative: create tables with schema default collation BINARY and descriptive columns having linguistic collation)

NLS_COMP parameter

- Can be BINARY, LINGUISTIC or ANSI
  (ANSI supported for backwards compatibility - not completely like LINGUISTIC)
- Makes comparisons use collation specified in NLS_SORT

- DISTINCT operation using NLS_SORT=XGerman_CI and NLS_COMP=LINGUISTIC
  – It considers `große` and `GROSSE` identical - which is returned is indeterminate

NLSSORT() function

- `NLSSORT(expression, 'NLS_SORT=collation')`
- Returns a collation key (string of bytes = RAW)
- Typically used in ORDER BY
- Function based index using NLSSORT with a given collation can be picked up by optimizer for ordering in sessions where NLS_SORT parameter is that collation

- Can be used for linguistic comparisons too like for example:
  `NLSSORT(exp1, 'NLS_SORT=coll') > NLSSORT(exp2, 'NLS_SORT=coll')`
- If application needs linguistic comparisons often, consider using NLS_COMP parm

NLSSORT in ORDER BY

Ordering by case insensitive Danish collation that sorts AA as Å

```
create table stores (
   store_id  varchar2(5  char) primary key
 , city      varchar2(20 char)
);


select store_id, city
  from stores
 order by nlssort(city, 'NLS_SORT=DANISH_M_CI');
```

```
STORE CITY
----- --------------------
BB002 andst
ÅÅ001 AUNING
BA001 karup
AA002 Korsør
BA002 KYBEHUSE
AA001 København
ÅÅ002 Kaastrup
BB001 AALBORG
AB002 Ålestrup
AB001 Aarhus
```

NLS_UPPER / NLS_LOWER / NLS_INITCAP

- `NLS_UPPER(expression, 'NLS_SORT=collation')`
- `NLS_LOWER(expression, 'NLS_SORT=collation')`
- `NLS_INITCAP(expression, 'NLS_SORT=collation')`

- Can be used in comparisons or ORDER BY as alternative to using "_CI" collation
- Depending on collation chosen, will handle special situations in some languages
  - German lowercase ß in uppercase is spelled SS
  - In Dutch `'ij'` is considered as a single character so at beginning of words NLS_INITCAP will turn `'ijsland'` into `'IJsland'`
  - `etc...`

Collations with special linguistic knowledge

Example of XGERMAN collation handling special rule for ß and SS
Note difference in going from lower to uppercase and vice versa

```
select nls_upper(
          'Grüß Gott'
        , 'NLS_SORT=XGERMAN'
       ) as greeting
  from dual;

select nls_lower(
          'GRÜSS GOTT'
        , 'NLS_SORT=XGERMAN'
       ) as greeting
  from dual;
```

```
GREETING
-----------
GRÜSS GOTT


GREETING
-----------
grüss gott
```

20c German capital ẞ support

- Capital ẞ part of Rechtschreibung since 2017:

  https://de.wikipedia.org/wiki/Gro%C3%9Fes_%C3%9F

- Supported from 20c by collation XGERMAN_S and XGERMAN_DIN_S:

  https://docs.oracle.com/en/database/oracle/oracle-database/20/newft/new-german-linguistic-sorts-capital-sharp-s-support.html

Data-bound collation

"Yeehaw, let's go bind some data!"

Data-bound collation

- 12.2 feature - needs MAX_STRING_SIZE set to EXTENDED
- Rather than putting NLS_SORT in all queries, define collation on a column
- Define collation at multiple levels
  - Statement level with COLLATE operator
  - Column level (table, view, materialized view)
    - Specified on column directly
    - Or inherited from defaults on table or schema
  - Function call collation
- Default collation when nothing is specified is pseudo-collation USING_NLS_SORT
  This means "behave like used to do" using NLS_SORT / NLS_COMP

Valid collations

View valid collations with V$NLS_VALID_VALUES

```
select value
  from v$nls_valid_values
 where parameter = 'SORT'
   and isdeprecated = 'FALSE'
 order by value;
```

```
VALUE
--------------------
ARABIC
ARABIC_ABJ_MATCH
ARABIC_ABJ_SORT
ARABIC_MATCH
ASCII7
AZERBAIJANI
BENGALI
BIG5
BINARY
BULGARIAN
CANADIAN_M
CATALAN
CROATIAN
CZECH
CZECH_PUNCTUATION
DANISH
DANISH_M
DUTCH
```

Specify on table

Collation can be set on a column directly or as default collation for the table
Changing default table collation does not change columns - only new columns

```
create table stores (
   store_id  varchar2(5  char)
             primary key
 , city      varchar2(20 char)
             collate danish_m_ci
)
default collation binary;
```

```
insert into stores values ('AA001', 'København');
insert into stores values ('AA002', 'Korsør');
insert into stores values ('AB001', 'Aarhus');
insert into stores values ('AB002', 'Ålestrup');
insert into stores values ('BA001', 'karup');
insert into stores values ('BA002', 'KYBEHUSE');
insert into stores values ('BB001', 'AALBORG');
insert into stores values ('BB002', 'andst');
insert into stores values ('ÅÅ001', 'AUNING');
insert into stores values ('ÅÅ002', 'Kaastrup');
commit;
```

Data-bound collation overrules session

Even though session NLS_SORT is binary, ordering by CITY uses the column collation
In this case DANISH_M_CI, so case insensitive and AA sorts like Å

```
alter session set nls_sort = binary;

select store_id, city
  from stores
 order by city;
```

```
STORE CITY
----- --------------------
BB002 andst
ÅÅ001 AUNING
BA001 karup
AA002 Korsør
BA002 KYBEHUSE
AA001 København
ÅÅ002 Kaastrup
BB001 AALBORG
AB002 Ålestrup
AB001 Aarhus
```

Overrule on statement level

Use COLLATE operator in ORDER BY clause - here use collation without AA = Å sorting
Alternatively COLLATE on inline view column (also works on real view column)

```
alter session set nls_sort = binary;

select store_id, city
  from stores
 order by city collate danish_ci;


select store_id, city2
  from (
   select store_id
        , city collate danish_ci
             as city2
     from stores
  )
 order by city2;
```

```
STORE CITY
----- --------------------
BB001 AALBORG
AB001 Aarhus
BB002 andst
ÅÅ001 AUNING
ÅÅ002 Kaastrup
BA001 karup
AA002 Korsør
BA002 KYBEHUSE
AA001 København
AB002 Ålestrup
```

Comparison with collation

Comparisons on the collated column obeys "_CI" case insensitivity

```
select store_id, city
  from stores
 where city like '%U%'
 order by city;
```

```
STORE CITY
----- --------------------
ÅÅ001 AUNING
BA001 karup
BA002 KYBEHUSE
ÅÅ002 Kaastrup
AB002 Ålestrup
AB001 Aarhus
```

```
select store_id, city
  from stores
 where instr(city, 'h') > 0
 order by city;
```

```
STORE CITY
----- --------------------
BA002 KYBEHUSE
AA001 København
AB001 Aarhus
```

12.2 LIKE operator quirk?

The AA=Å rule of DANISH_M is not quite consistently obeyed by LIKE operator
Test your own special language rules whether they are implemented well

```
select store_id, city
  from stores
 where city like 'a%'
 order by city;

select store_id, city
  from stores
 where city like 'å%'
 order by city;

select store_id, city
  from stores
 where city like 'aa%'
 order by city
```

```
STORE CITY
----- --------------------
BB002 andst
ÅÅ001 AUNING

STORE CITY
----- --------------------
AB002 Ålestrup


STORE CITY
----- --------------------
BB001 AALBORG
AB001 Aarhus
```

Database Migration Assistant for Unicode (DMU)

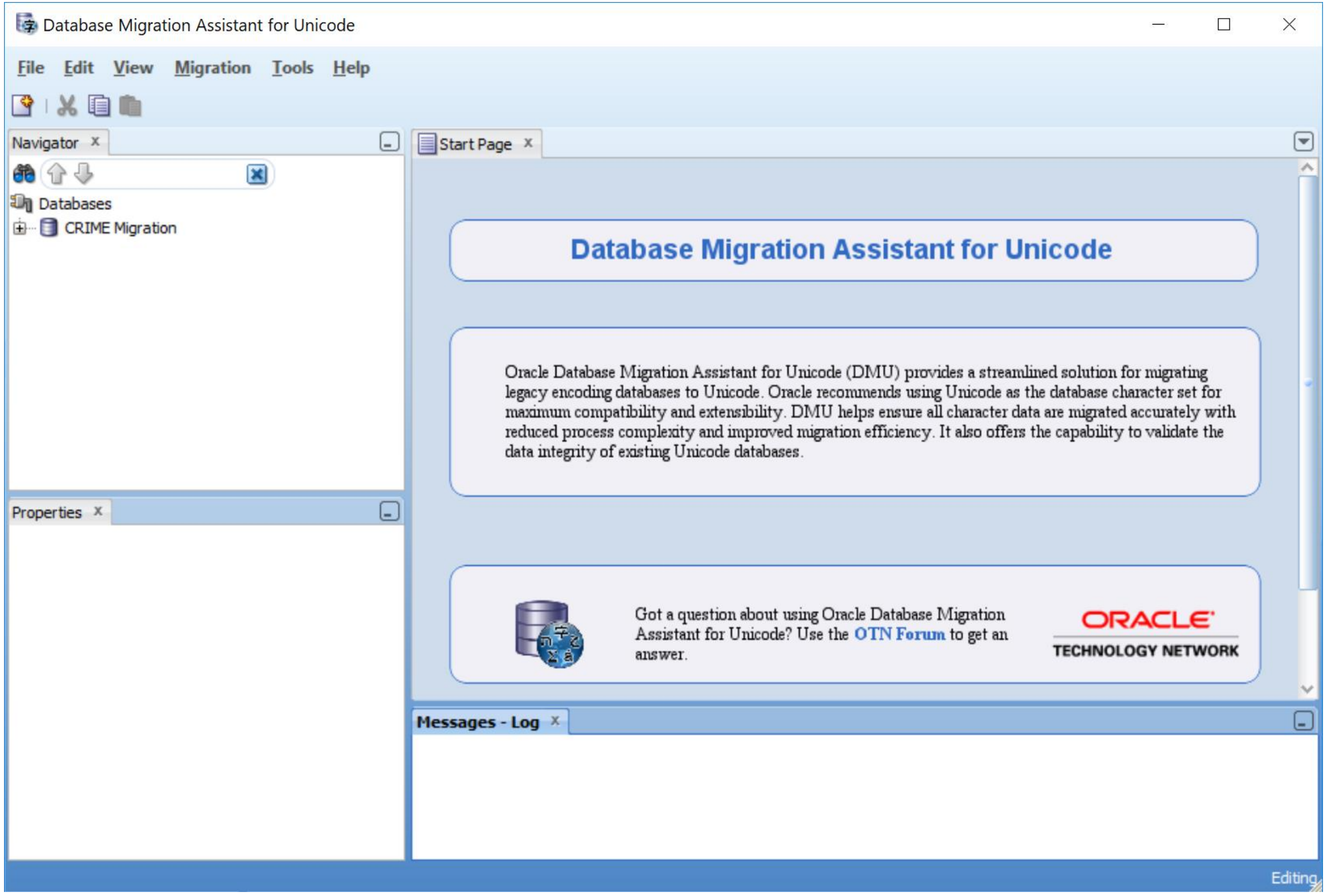
"Yo guys, let's migrate to Unicode"

Database Migration Assistant for Unicode

- Tool with GUI that can do
  - Convert a database from single-byte charset to AL32UTF8
  - Scan single-byte charset databases for invalid data or other problems that would cause trouble at conversion
  - Scan AL32UTF8 charset databases for invalid data
- MOS Note: Doc ID 1272374.1 - Explains most about the tool
- DMU docs: https://docs.oracle.com/cd/E64126_01/index.htm
- DMU FAQ: http://www.oracle.com/technetwork/database/database-technologies/globalization/dmu/learnmore/faq-345828.html
- MOS Note: Doc ID 1900712.2 - DMU troubleshooting guide

DMU s

- Java GUI
- Run either
  - On DB server
  - On client
- Download
  - MOS (sup.)
  - OTN



Database Migration Assistant for Unicode

Oracle Database Migration Assistant for Unicode (DMU) provides a streamlined solution for migrating legacy encoding databases to Unicode. Oracle recommends using Unicode as the database character set for maximum compatibility and extensibility. DMU helps ensure all character data are migrated accurately with reduced process complexity and improved migration efficiency. It also offers the capability to validate the data integrity of existing Unicode databases.

Got a question about using Oracle Database Migration Assistant for Unicode? Use the OTN Forum to get an answer.

Working with DMU

1. Scan the database (repeat until no problems reported)

   – DMU will report anything that cannot be converted
     - for example invalid byte values, data that will be >4000 bytes after conversion, non-ASCII or non-ISO values in data dictionary, etc.

   – DMU has tools for repairing some of the problems
     - for example convert columns from BYTE semantics to CHAR semantics, replace invalid byte values, etc.

2. Convert / migrate the data

   – If no ROWID dependencies, consider setting parameter "Use CTAS" in guide

3. Scan the result to verify correct migration to AL32UTF8

Possible issues that can block DMU

- DMU will generally not touch data in the data dictionary (SYS and other schemas)
- So for example column names or procedure parameter names like `ZURÜCK` cannot be converted - they must be changed before DMU can do conversion
- Most source code, though, DMU can handle. An exception is object type specifications - for example if a type spec includes a comment like `/* Author: Schrödinger */` , then DMU cannot convert the database
- VARCHAR2 attributes in object types that are used in tables / queues and need to be changed from BYTE to CHAR semantics cannot be changed without dropping table
  - This can be problematic even by manual datapump export, drop table, drop type and recreate with CHAR semantics => datapump import fails as type signature has changed (this even if type recreated with same object ID)

Possible issues... (continued)

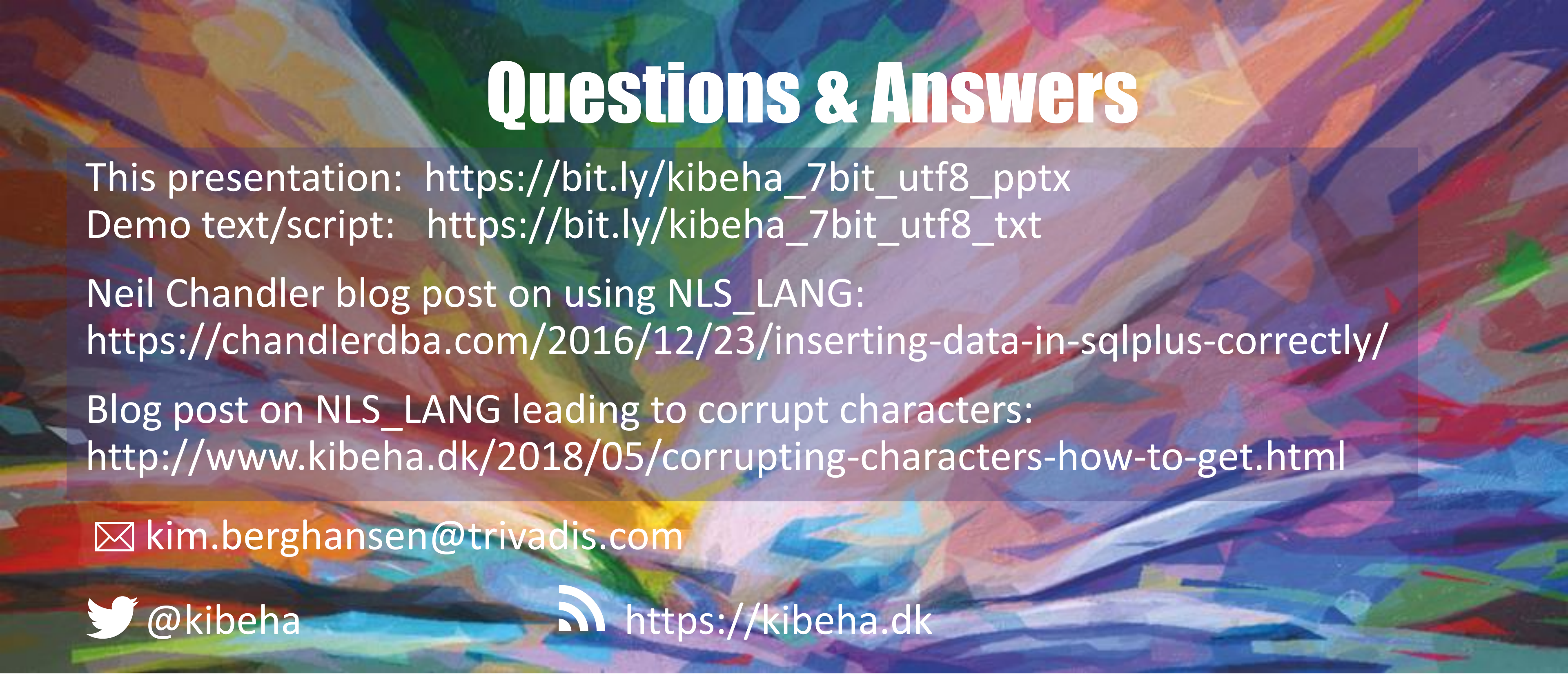
- Some columns might after conversion lead to an index becoming too long
- Some data might after conversion no longer fit in VARCHAR2(4000 / 32K)
- Oracle Text metadata in CTXSYS schema cannot be touched by DMU
  - For example if PRINTJOINS for a text index has § character, DMU won't work
- Workload statistics in WRH$_SQLSTAT contain session ACTION and MODULE, so when German PL/SQL Developer IDE sets MODULE to "Fenster für SQL", then DMU won't work until the workload stats has been purged
- etc...

Lessons learned

- DMU great tool for finding out issues in the database
- Depending on results of those findings, you can choose either
  - fix/workaround the issues and do the conversion with the DMU
  - or build new AL32UTF8 database and move the data (for example datapump)
- There can be so many variants of small issues that it is not realistic to fully automate, manual work in the preparation phase is needed

- Can be good idea to scan DB with DMU even if not migrating characterset
  - Locating rows/columns with bad text helps find clients using wrong NLS_LANG
  - Can fix corrupt text

The last bit



Questions & Answers

This presentation: https://bit.ly/kibeha_7bit_utf8_pptx
Demo text/script: https://bit.ly/kibeha_7bit_utf8_txt

Neil Chandler blog post on using NLS_LANG:
https://chandlerdba.com/2016/12/23/inserting-data-in-sqlplus-correctly/

Blog post on NLS_LANG leading to corrupt characters:
http://www.kibeha.dk/2018/05/corrupting-characters-how-to-get.html

✉ kim.berghansen@trivadis.com

@kibeha

https://kibeha.dk