Text in Databases

Charles Severance

www.pg4e.com/lectures/04-Text.sql

Generating Test Data

Exploring performance

- We can't really explore performance if we only have 5 records
- So before we play a bit with performance, we need to make up some data

Generating lots of Random Data

- We use **repeat()** to generate long strings (horizontal)
- We use **generate_series()** to generate lots of rows (vertical)
  - Like Python's range
- We use **random()** to make rows unique
  - Floating point 0 <= random() <= 1.0

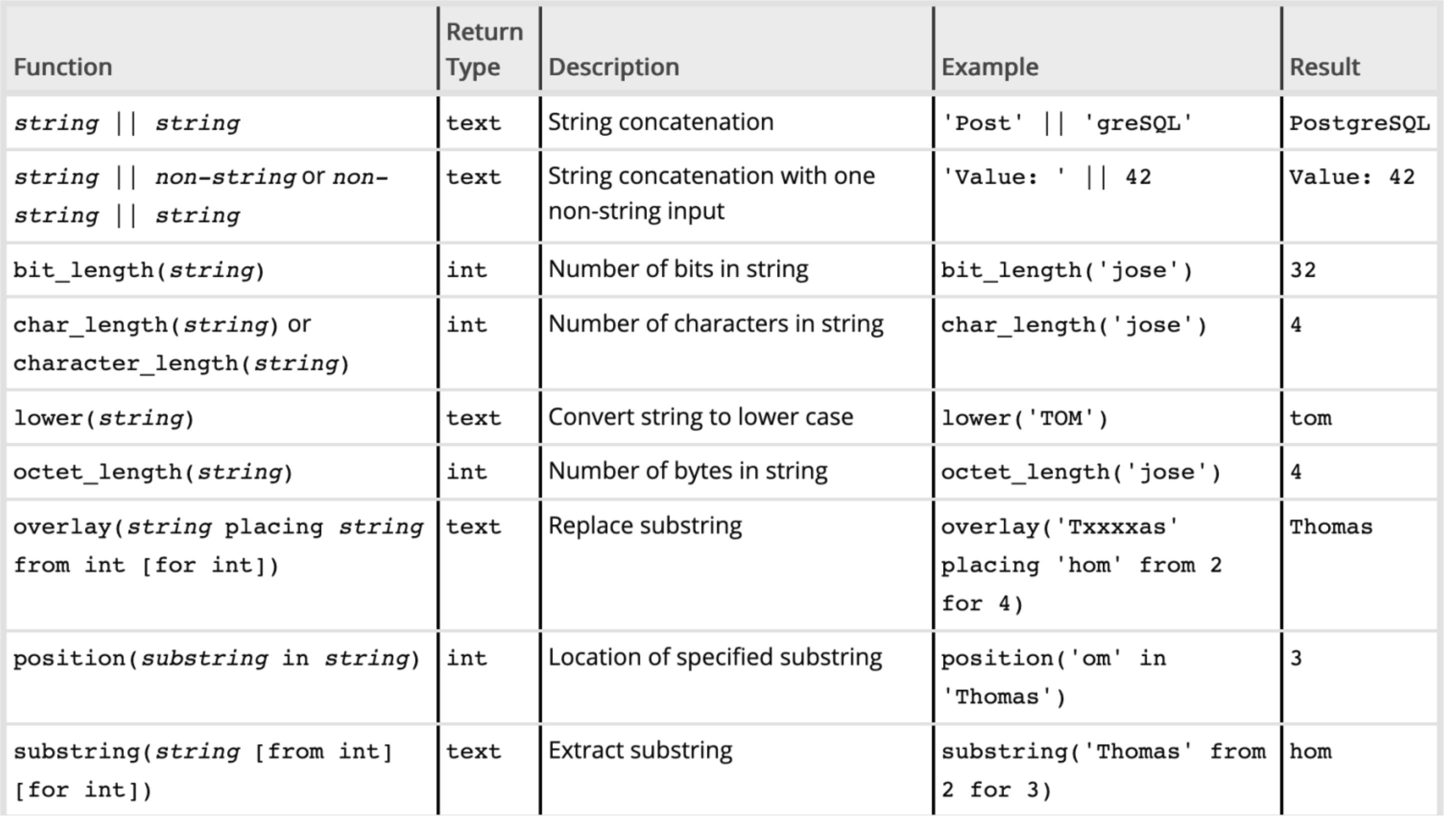
```
discuss=> select random(), random(), trunc(random()*100);
      random       |      random       | trunc
-------------------+-------------------+-------
 0.192553216125816 | 0.751528221182525 |    91

discuss=> select repeat('Neon ', 5);
          repeat
---------------------------
 Neon Neon Neon Neon Neon

discuss=> select generate_series(1,5);
 generate_series
-----------------
               1
               2
               3
               4
               5
```

```
discuss=> select 'https://sql4e.com/neon/' ||
discuss->   trunc(random()*1000000) || repeat('Lemon', 5) ||
discuss->   generate_series(1,5);

                        ?column?
---------------------------------------------------------
 https://sql4e.com/neon/225845LemonLemonLemonLemonLemon1
 https://sql4e.com/neon/679405LemonLemonLemonLemonLemon2
 https://sql4e.com/neon/603925LemonLemonLemonLemonLemon3
 https://sql4e.com/neon/917014LemonLemonLemonLemonLemon4
 https://sql4e.com/neon/428156LemonLemonLemonLemonLemon5
```

Text Functions

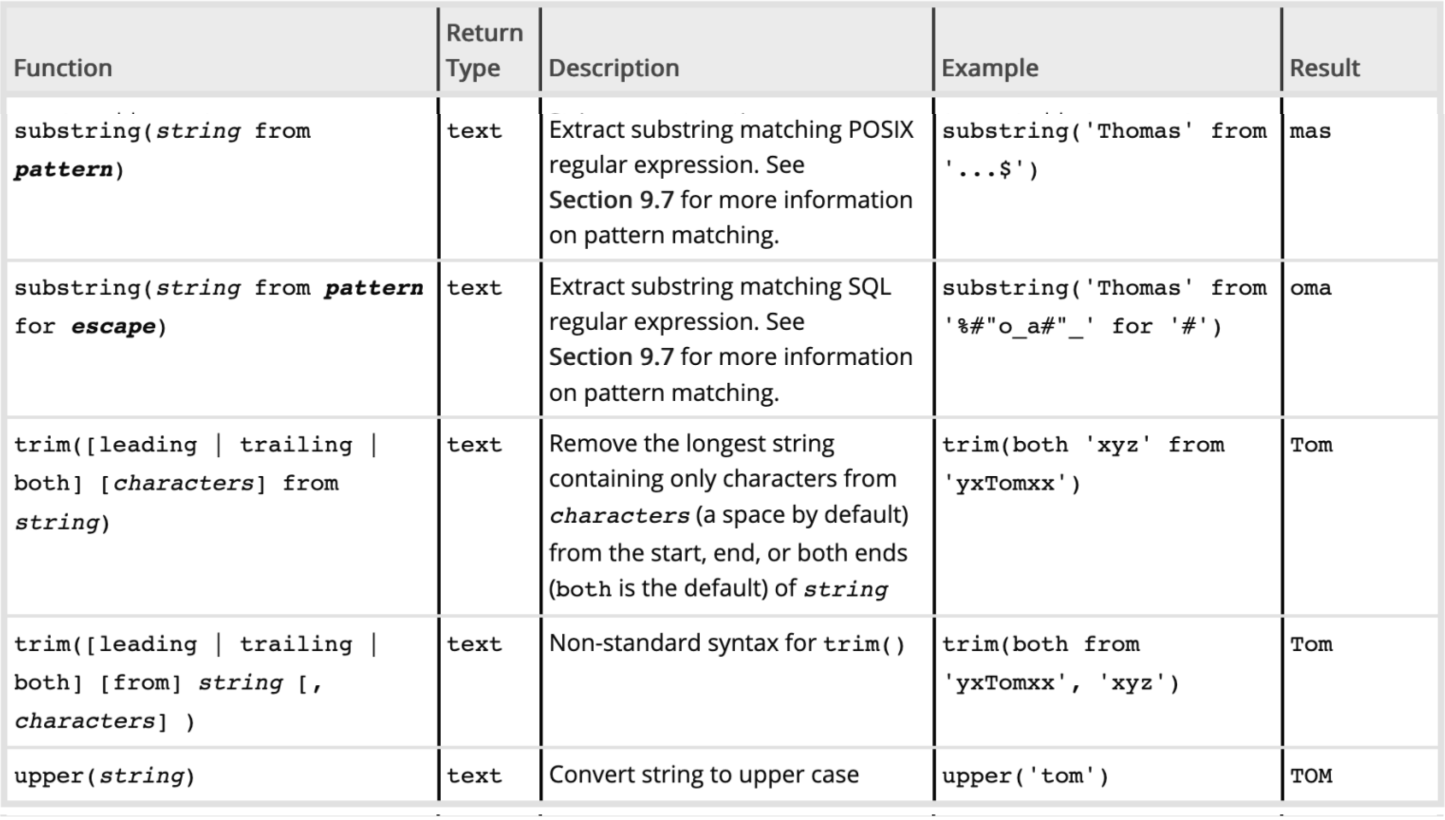
Many Text Functions

- Where Clause Operators
  - LIKE / ILIKE / NOT LIKE / NOT ILIKE
  - SIMILAR TO/ NOT SIMILAR TO (cover later as regular expressions)
  - **= > < >= <= BETWEEN IN**
- Manipulate SELECT Results / WHERE clause
  - lower(), upper()



Function	Return Type	Description	Example	Result
string || *string*	text	String concatenation	'Post' || 'greSQL'	PostgreSQL
string || *non-string* or *non-string* || *string*	text	String concatenation with one non-string input	'Value: ' || 42	Value: 42
bit_length(*string*)	int	Number of bits in string	bit_length('jose')	32
char_length(*string*) or character_length(*string*)	int	Number of characters in string	char_length('jose')	4
lower(*string*)	text	Convert string to lower case	lower('TOM')	tom
octet_length(*string*)	int	Number of bytes in string	octet_length('jose')	4
overlay(*string* placing *string* from int [for int])	text	Replace substring	overlay('Txxxxas' placing 'hom' from 2 for 4)	Thomas
position(*substring* in *string*)	int	Location of specified substring	position('om' in 'Thomas')	3
substring(*string* [from int] [for int])	text	Extract substring	substring('Thomas' from 2 for 3)	hom

https://www.postgresql.org/docs/11/functions-string.html



Function	Return Type	Description	Example	Result
substring(*string* from ***pattern***)	text	Extract substring matching POSIX regular expression. See **Section 9.7** for more information on pattern matching.	substring('Thomas' from '...$')	mas
substring(*string* from ***pattern*** for ***escape***)	text	Extract substring matching SQL regular expression. See **Section 9.7** for more information on pattern matching.	substring('Thomas' from '%#"o_a#"_' for '#')	oma
trim([leading | trailing | both] [*characters*] from *string*)	text	Remove the longest string containing only characters from *characters* (a space by default) from the start, end, or both ends (both is the default) of *string*	trim(both 'xyz' from 'yxTomxx')	Tom
trim([leading | trailing | both] [from] *string* [, *characters*])	text	Non-standard syntax for trim()	trim(both from 'yxTomxx', 'xyz')	Tom
upper(*string*)	text	Convert string to upper case	upper('tom')	TOM

https://www.postgresql.org/docs/11/functions-string.html

```
CREATE TABLE textfun (
  content TEXT
);

CREATE INDEX textfun_b ON textfun (content);
```

```
discuss=> SELECT pg_relation_size('textfun'), pg_indexes_size('textfun');
 pg_relation_size | pg_indexes_size
------------------+-----------------
                0 |            8192
```

```
discuss=> SELECT pg_relation_size('textfun'), pg_indexes_size('textfun');
 pg_relation_size | pg_indexes_size
------------------+-----------------
                0 |            8192
```

```
INSERT INTO textfun (content)
SELECT (CASE WHEN (random() < 0.5)
         THEN 'https://www.pg4e.com/neon/'
         ELSE 'http://www.pg4e.com/LEMONS/'
         END) || generate_series(100000,200000);
```

```
discuss=> SELECT pg_relation_size('textfun'), pg_indexes_size('textfun');
 pg_relation_size | pg_indexes_size
------------------+-----------------
           6832128|          8585216
```

```
discuss=> SELECT content FROM textfun LIMIT 5;
             content
---------------------------------
 http://www.pg4e.com/LEMONS/100000
 http://www.pg4e.com/LEMONS/100001
 https://www.pg4e.com/neon/100002
 http://www.pg4e.com/LEMONS/100003
 http://www.pg4e.com/LEMONS/100004

discuss=> SELECT pg_relation_size('textfun'), pg_indexes_size('textfun');
 pg_relation_size | pg_indexes_size
------------------+-----------------
          6832128 |         5931008
```

```
INSERT INTO textfun (content)
SELECT (CASE WHEN (random() < 0.5)
         THEN 'https://www.pg4e.com/neon/'
         ELSE 'http://www.pg4e.com/LEMONS/'
         END) || generate_series(100000,200000);
```

Text Functions

```
SELECT content FROM textfun WHERE content LIKE '%150000%';
--  https://www.pg4e.com/neon/150000
SELECT upper(content) FROM textfun WHERE content LIKE '%150000%';
--  HTTPS://WWW.PG4E.COM/NEON/150000
SELECT lower(content) FROM textfun WHERE content LIKE '%150000%';
--  https://www.pg4e.com/neon/150000
SELECT right(content, 4) FROM textfun WHERE content LIKE '%150000%';
-- 0000
SELECT left(content, 4) FROM textfun WHERE content LIKE '%150000%';
-- http
```

Moar Text Functions

```
SELECT content FROM textfun WHERE content LIKE '%150000%';
--  https://www.pg4e.com/neon/150000
SELECT strpos(content, 'ttps://') FROM textfun WHERE …
-- 2
SELECT substr(content, 2, 4) FROM textfun WHERE …
-- ttps
SELECT split_part(content, '/', 4) FROM textfun WHERE …
-- neon
SELECT translate(content, 'th.p/', 'TH!P_') FROM textfun …
--  HTTPs:__www!Pg4e!com_neon_150000
```

B-Tree Index performance

```
discuss=> explain analyze SELECT content FROM textfun WHERE content LIKE 'racing%';
Index Only Scan using textfun_b on textfun
   Index Cond: ((content >= 'racing'::text) AND (content < 'racinh'::text))
   Filter: (content ~~ 'racing%'::text)
   Heap Fetches: 0
Execution Time: 0.011 ms

discuss=> explain analyze SELECT content FROM textfun WHERE content LIKE '%racing%';
Seq Scan on textfun
Filter: (content ~~ '%racing%'::text)
   Rows Removed by Filter: 100001
Execution Time: 10.271 ms

discuss=> explain analyze SELECT content FROM textfun WHERE content ILIKE 'racing%';
Seq Scan on textfun
   Filter: (content ~~* 'racing%'::text)
   Rows Removed by Filter: 100001
Execution Time: 29.958 ms
```

```
CREATE INDEX textfun_b ON textfun (content);
```

```
discuss=> explain analyze SELECT content FROM textfun WHERE content
discuss->   LIKE '%150000%';
 Seq Scan on textfun
   Filter: (content ~~ '%150000%'::text)
   Rows Removed by Filter: 100000
Execution Time: 14.923 ms

discuss=> explain analyze SELECT content FROM textfun WHERE content
discuss->    LIKE '%150000%' LIMIT 1;
Limit  (cost=0.00..208.40 rows=1 width=33)
   ->  Seq Scan on textfun
         Filter: (content ~~ '%150000%'::text)
         Rows Removed by Filter: 50000
 Planning Time: 0.116 ms
 Execution Time: 8.732 ms
```

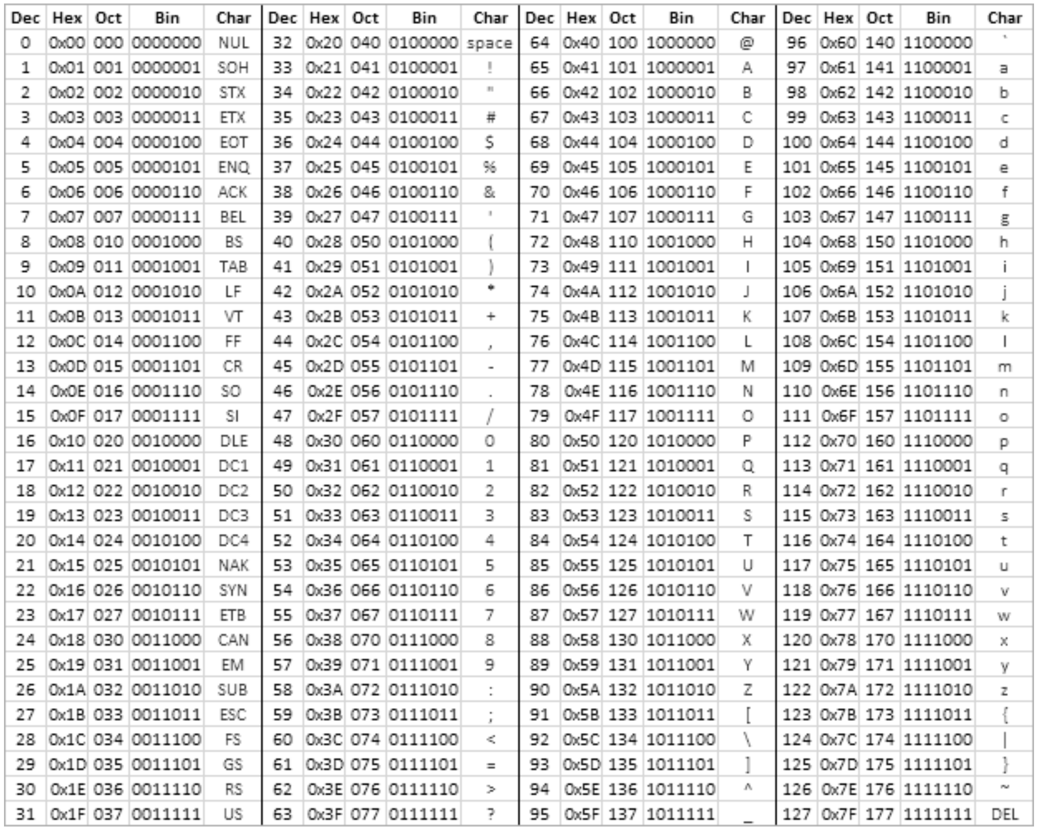
```
discuss=> explain analyze SELECT content FROM textfun
discuss-> WHERE content IN ('http://www.pg4e.com/neon/150000',
diacuss-> 'https://www.pg4e.com/neon/150000');

 Index Only Scan using textfun_b on textfun
   Index Cond: (content = ANY ('{http://www.pg4e.com/neon/150000,
                              https://www.pg4e.com/neon/150000}'::text[]))
 Execution Time: 0.036 ms

discuss=> explain analyze SELECT content FROM textfun
discuss-> WHERE content
discuss-> IN (SELECT content FROM textfun WHERE content LIKE '%150000%');
 Nested Loop
   ->  HashAggregate
         Group Key: textfun_1.content
         ->  Seq Scan on textfun textfun_1
               Filter: (content ~~ '%150000%'::text)
               Rows Removed by Filter: 100000
   ->  Index Only Scan using textfun_b on textfun
         Index Cond: (content = textfun_1.content)
 Execution Time: 14.302 ms
```

Character Sets

https://www.py4e.com/lessons/network

ASCII

- American Standard Code for Information Interchange



Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

https://en.wikipedia.org/wiki/ASCII
http://www.catonmat.net/download/ascii-cheat-sheet.png

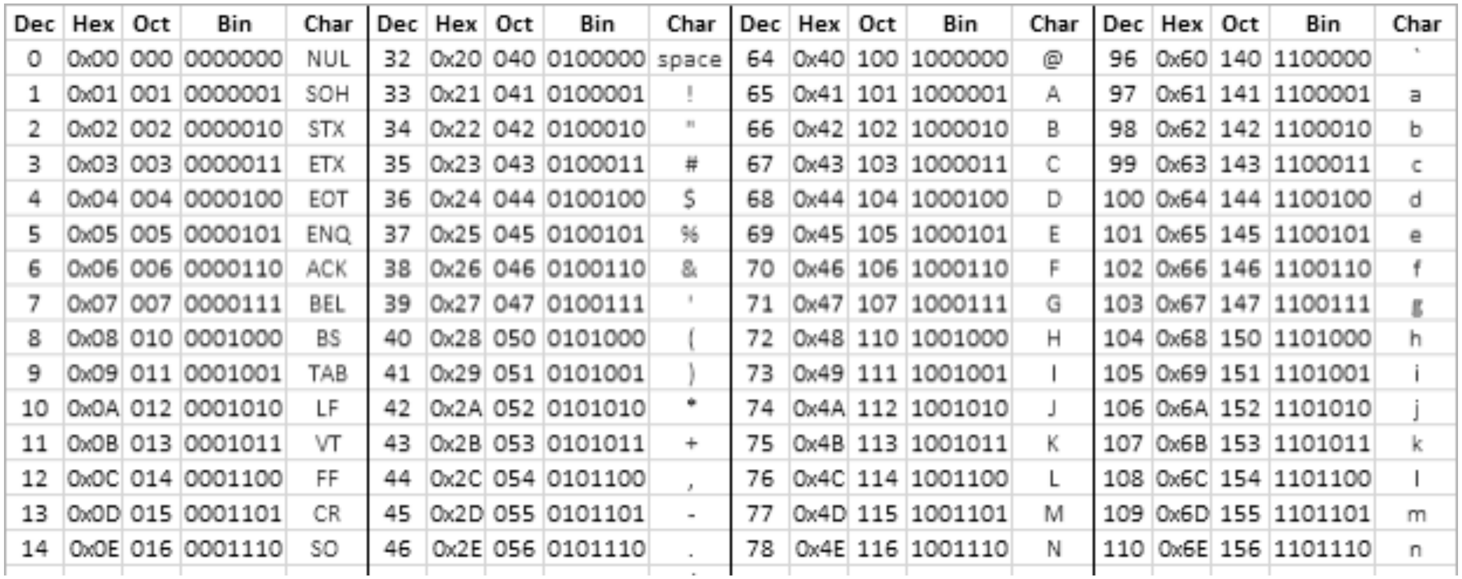
Representing Simple Strings

- "In the old days" - each character is represented by a number between 0 and 127 stored in 8 bits of memory
- We refer to "8 bits of memory as a "byte" of memory
- The ascii() function tells us the numeric value of a single ASCII character
- The chr() function maps from an integer to a character

```
discuss=> select ascii('H'), ascii('e'), ascii('l'), chr(72), chr(42);
 ascii | ascii | ascii | chr | chr
-------+-------+-------+-----+-----
    72 |   101 |   108 | H   | *
```

```
discuss=> select ascii('H'), ascii('e'), ascii('l'), chr(72), chr(42);
 ascii | ascii | ascii | chr | chr
-------+-------+-------+-----+-----
    72 |   101 |   108 | H   | *
```

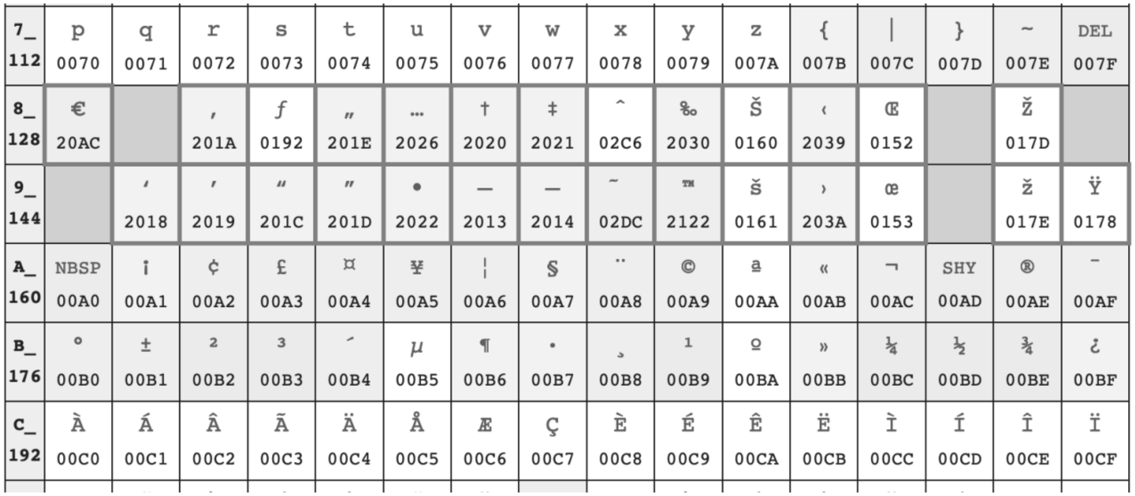
In the 1960s and 1970s, we just assumed that one byte was one character



Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n

Beyond 127…

- To be "more international" they defined characters 128-255 but inconsistently
  - https://en.wikipedia.org/wiki/ISO/IEC_8859-1 (latin1)
  - https://en.m.wikipedia.org/wiki/Windows-1252



7_ 112	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
8_ 128	€ 20AC		‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	Š 0160	‹ 2039	Œ 0152		Ž 017D	
9_ 144		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	› 203A	œ 0153		ž 017E	Ÿ 0178
A_ 160	NBSP 00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	SHY 00AD	® 00AE	¯ 00AF
B_ 176	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C_ 192	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF

Don't cut and paste code or text from PDFs ☹

Overlapping character sets

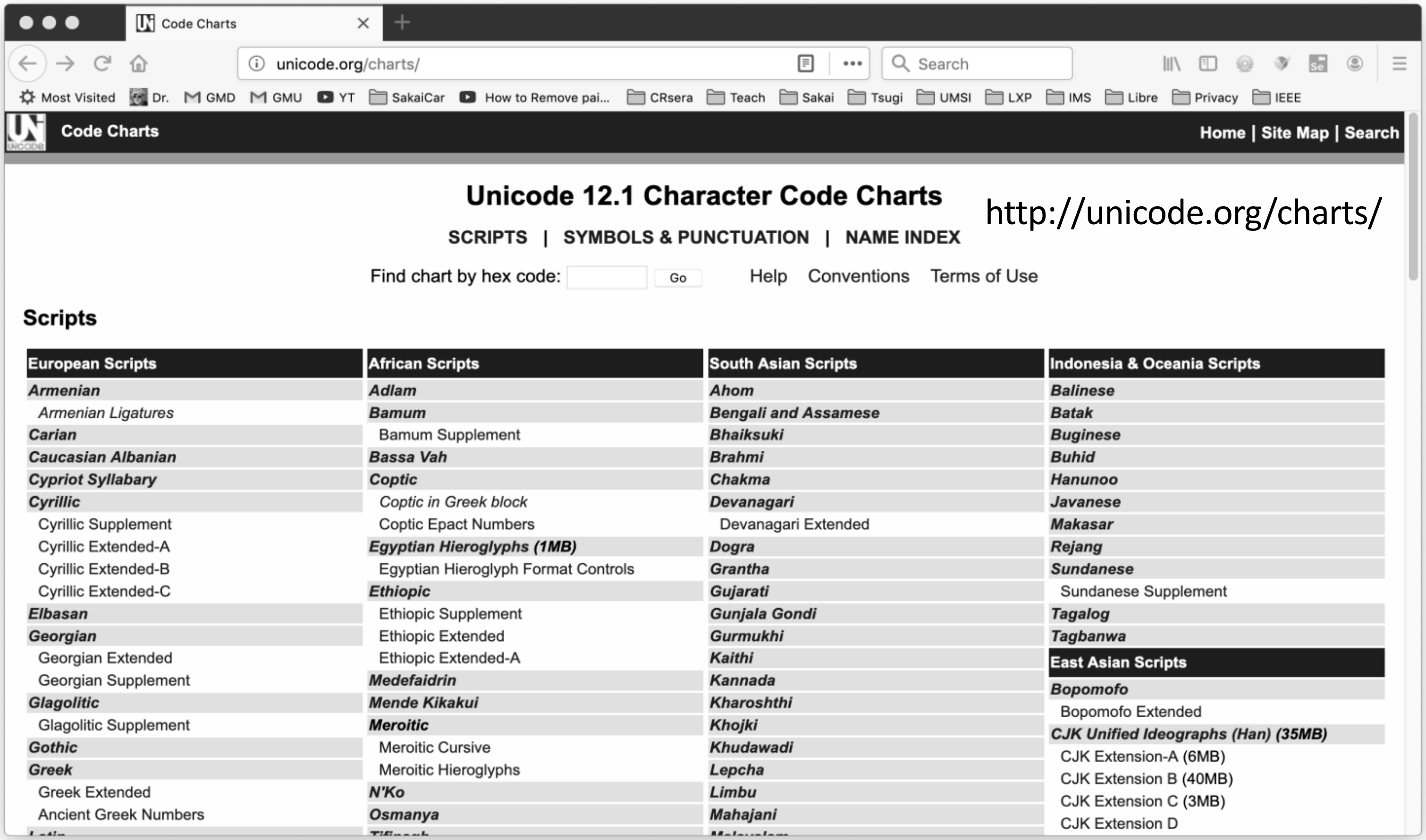
- We needed more than 128 new characters globally so 128-255 could mean different things based on context
  - ISO 8859-2 for Eastern European languages
  - ISO 8859-3 for Turkish, Maltese and Esperanto
  - ISO 8859-5 for Cyrillic
- But these were not self-documenting you needed to know the character set outside the data of the file ---- Confusion

Unicode – All Characters in One Set

- Unicode is 32 / 21 bits (long story)
- Unicode 12.1
  - 137,000 characters
  - 150 character sets

```
discuss=> select chr(72), chr(231), chr(20013);
 chr | chr | chr
-----+-----+-----
 H   | ç   | 中
```

https://en.wikipedia.org/wiki/List_of_Unicode_characters



http://unicode.org/charts/

Unicode 12.1 Character Code Charts

SCRIPTS | SYMBOLS & PUNCTUATION | NAME INDEX

Find chart by hex code: Go   Help   Conventions   Terms of Use

Scripts

European Scripts

- ***Armenian***
- *Armenian Ligatures*
- ***Carian***
- ***Caucasian Albanian***
- ***Cypriot Syllabary***
- ***Cyrillic***
- Cyrillic Supplement
- Cyrillic Extended-A
- Cyrillic Extended-B
- Cyrillic Extended-C
- ***Elbasan***
- ***Georgian***
- Georgian Extended
- Georgian Supplement
- ***Glagolitic***
- Glagolitic Supplement
- ***Gothic***
- ***Greek***
- Greek Extended
- Ancient Greek Numbers
- ***Latin***

African Scripts

- ***Adlam***
- ***Bamum***
- Bamum Supplement
- ***Bassa Vah***
- ***Coptic***
- *Coptic in Greek block*
- Coptic Epact Numbers
- ***Egyptian Hieroglyphs (1MB)***
- Egyptian Hieroglyph Format Controls
- ***Ethiopic***
- Ethiopic Supplement
- Ethiopic Extended
- Ethiopic Extended-A
- ***Medefaidrin***
- ***Mende Kikakui***
- ***Meroitic***
- Meroitic Cursive
- Meroitic Hieroglyphs
- ***N'Ko***
- ***Osmanya***
- ***Tifinagh***

South Asian Scripts

- ***Ahom***
- ***Bengali and Assamese***
- ***Bhaiksuki***
- ***Brahmi***
- ***Chakma***
- ***Devanagari***
- Devanagari Extended
- ***Dogra***
- ***Grantha***
- ***Gujarati***
- ***Gunjala Gondi***
- ***Gurmukhi***
- ***Kaithi***
- ***Kannada***
- ***Kharoshthi***
- ***Khojki***
- ***Khudawadi***
- ***Lepcha***
- ***Limbu***
- ***Mahajani***
- ***Malayalam***

Indonesia & Oceania Scripts

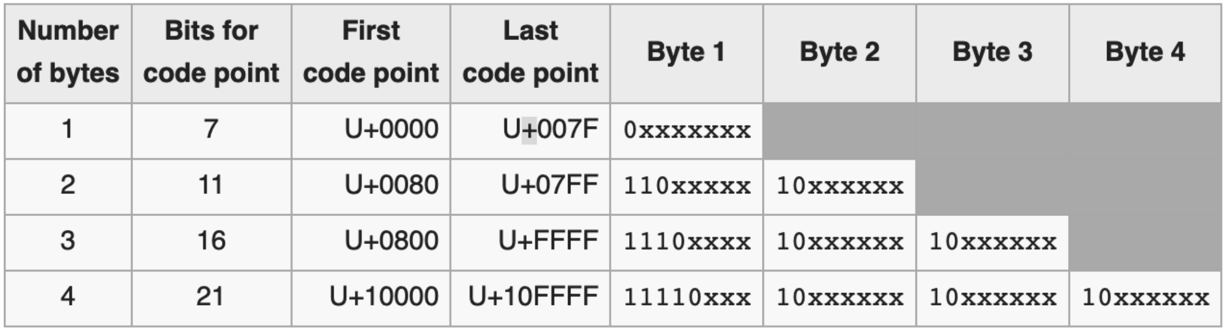
- ***Balinese***
- ***Batak***
- ***Buginese***
- ***Buhid***
- ***Hanunoo***
- ***Javanese***
- ***Makasar***
- ***Rejang***
- ***Sundanese***
- Sundanese Supplement
- ***Tagalog***
- ***Tagbanwa***

East Asian Scripts

- ***Bopomofo***
- Bopomofo Extended
- ***CJK Unified Ideographs (Han) (35MB)***
- CJK Extension-A (6MB)
- CJK Extension B (40MB)
- CJK Extension C (3MB)
- CJK Extension D

We can't afford 32 bit characters

- UTF-8 is a compression scheme for Unicode
  - Represents 21 bits in 8-32 bits
  - 0-128 are ASCII
  - 128-255 are signals



Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

https://en.wikipedia.org/wiki/UTF-8

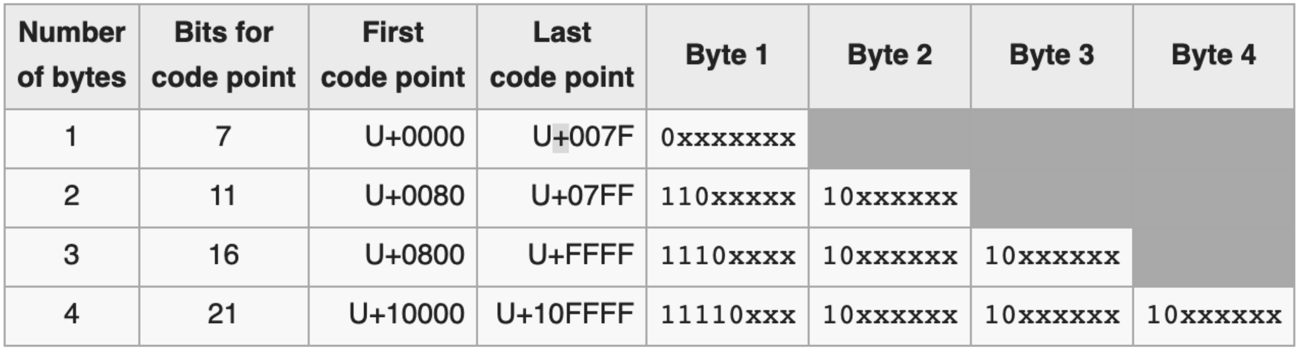
Space – the final frontier…

```
discuss=> SELECT char_length('学习管理'), octet_length('学习管理'),
discuss->   bit_length('学习管理'), ascii('学');

 char_length | octet_length | bit_length | ascii
-------------+--------------+------------+-------
           4 |           12 |         96 | 23398
```

… for Unicode

UTF-8 Designed for Transition

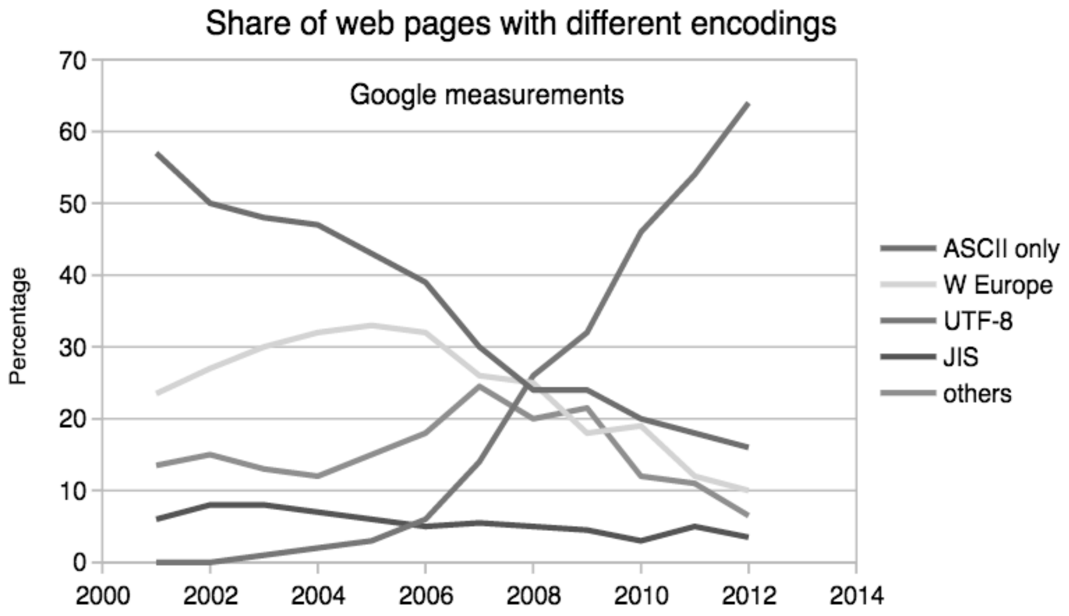
- Pure ASCII is UTF-8 / no conversion
- Partial auto detect/convert of
  - Latin-1 variants
  - 1252 variants



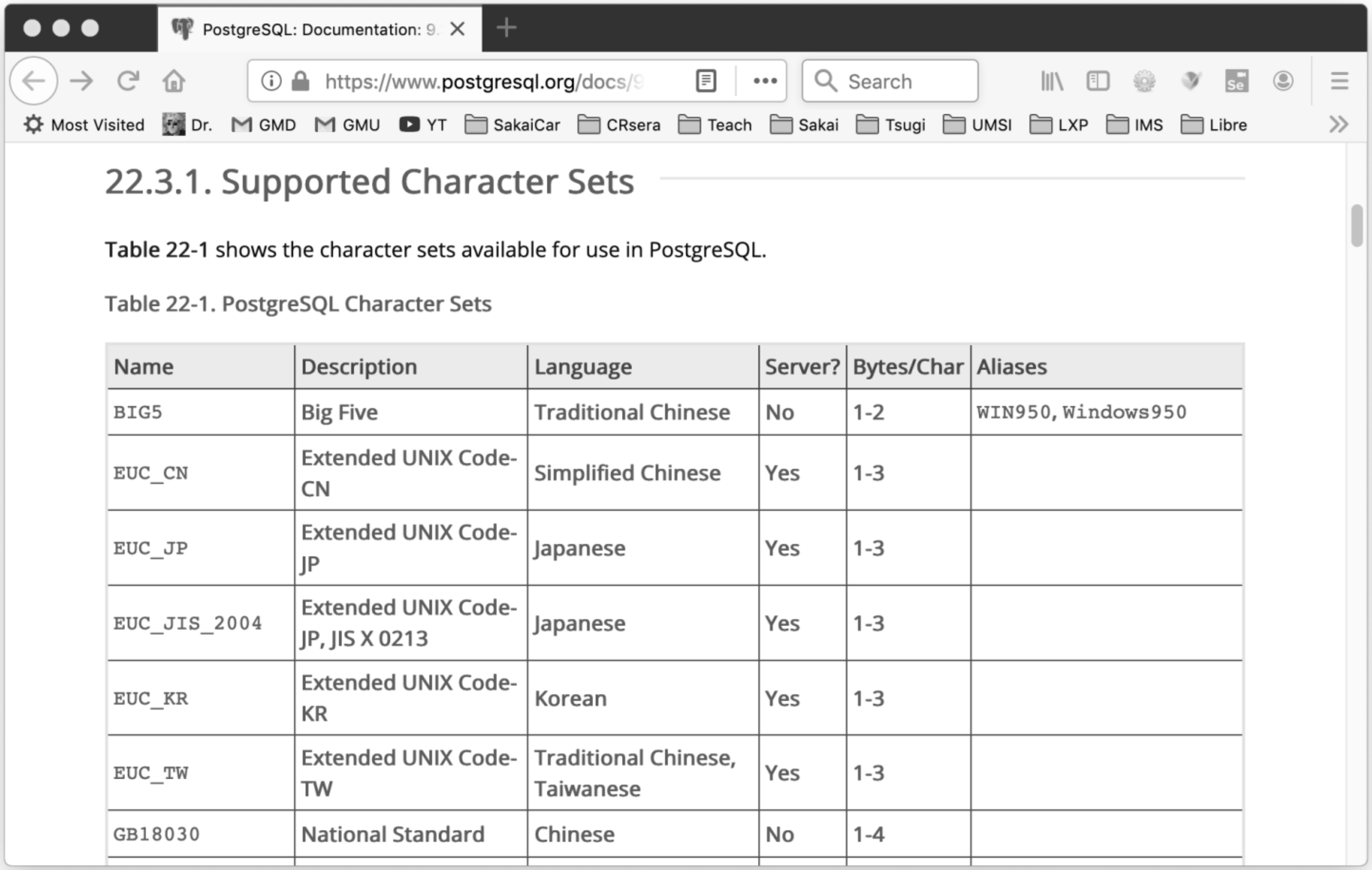
Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-8 Is Dominant

- Rapid uptake after 2004
- UTF-8 is 94% of all web pages in 2019



https://w3techs.com/technologies/details/en-utf8/all/all



22.3.1. Supported Character Sets

Table 22-1 shows the character sets available for use in PostgreSQL.

Table 22-1. PostgreSQL Character Sets

Name	Description	Language	Server?	Bytes/Char	Aliases
`BIG5`	Big Five	Traditional Chinese	No	1-2	`WIN950`, `Windows950`
`EUC_CN`	Extended UNIX Code-CN	Simplified Chinese	Yes	1-3	
`EUC_JP`	Extended UNIX Code-JP	Japanese	Yes	1-3	
`EUC_JIS_2004`	Extended UNIX Code-JP, JIS X 0213	Japanese	Yes	1-3	
`EUC_KR`	Extended UNIX Code-KR	Korean	Yes	1-3	
`EUC_TW`	Extended UNIX Code-TW	Traditional Chinese, Taiwanese	Yes	1-3	
`GB18030`	National Standard	Chinese	No	1-4	

```
discuss=> SHOW SERVER_ENCODING;
 server_encoding
-----------------
 UTF8
(1 row)
```

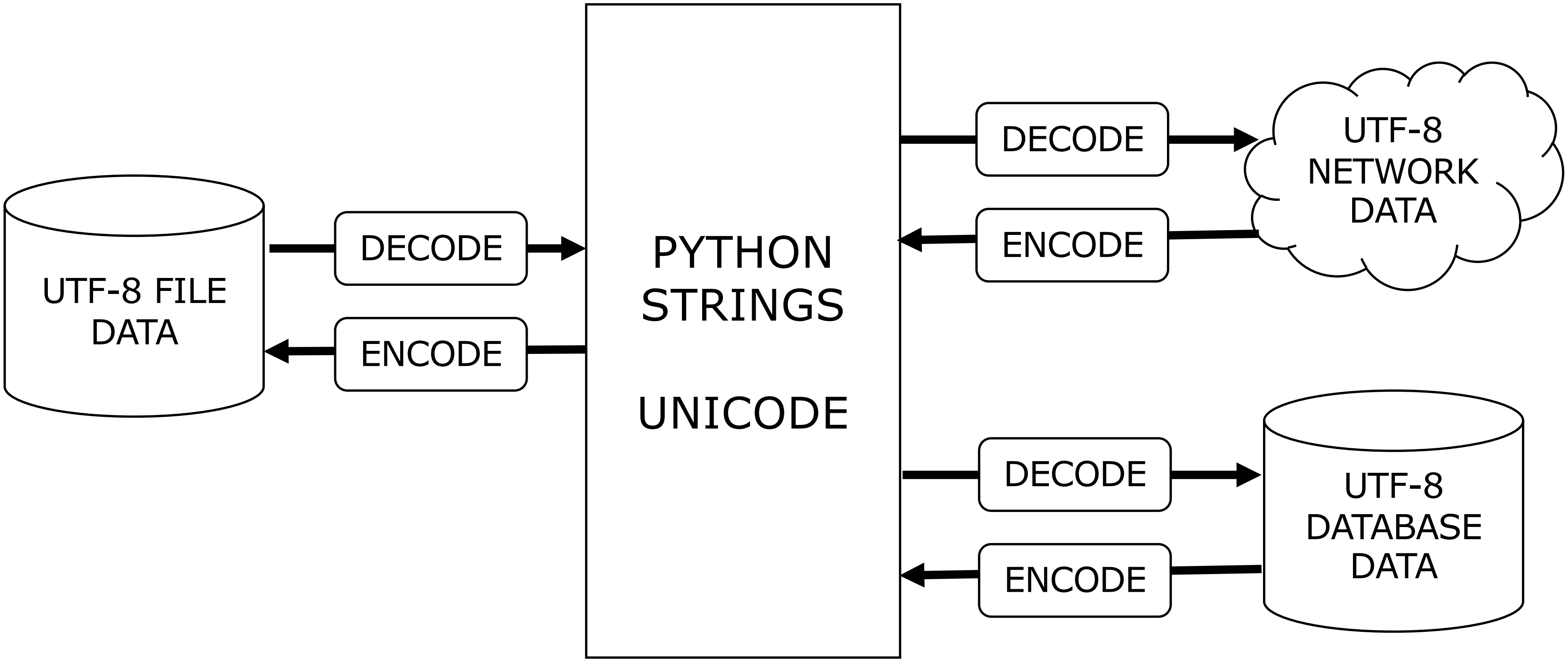
Character Sets In Python

https://www.py4e.com/lessons/network

Python 3 and Unicode

- Strings in memory are Unicode
- The "bytes" type is for 8-bit characters
- Strings "at rest" are generally stored UTF-8 for space and interoperability
  - Files
  - Network resources
  - Database tables

```
>>> x = b'abc'
>>> type(x)
<class 'bytes'>
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
```



UTF-8 FILE DATA

DECODE

ENCODE

PYTHON STRINGS

UNICODE

DECODE

ENCODE

UTF-8 NETWORK DATA

DECODE

ENCODE

UTF-8 DATABASE DATA

Opening a File

```
open(file, mode='r', buffering=-1,
 encoding=None, errors=None, newline=None,
 closefd=True, opener=None)
```

encoding is the name of the encoding used to decode or encode the file. This should only be used in text mode. The default encoding is platform dependent (whatever locale.getpreferredencoding() returns), but any text encoding supported by Python can be used. See the codecs module for the list of supported encodings.

https://docs.python.org/3/library/functions.html#open

Reading Network Data

- When we read data from an network resource, we must decode it based on the character set so it is properly represented in Python 3 as a UNICODE string

```
while True:
    data = mysock.recv(512)
    if ( len(data) < 1 ) :
        break
    mystring = data.decode()
    print(mystring)
```

Database Data

- When you interact with a database from Python all conversion between Unicode and UTF-8 is done implicitly
- The Python database connector (i.e. psycopg2) knows the internal storage format of the database and automatically handles all conversion

Inside Hashes

A **hash function** is any function that can be used to map data of arbitrary size onto data of a fixed size.

https://en.wikipedia.org/wiki/Hash_function

Uses of Hashes

- Checksum – see if a message was altered in transit
- Cryptography / Signature – See if a message came from a trusted source
- Good functions enable fast lookup of data
  - Python dictionaries
  - Database tables

Good Hash Functions

- Deterministic – There can be no randomness – must get the same output for the same input
- Uniform Distribution – Should have an equal chance of generating any value with the range of its outputs – values don't cluster or collide
- Sensitive – Any change in input should provide a change in output
- One-way – You should not be able to derive the input from the output (cannot reverse)

Special Math for Hash Computation

- Bitwise operators
  - << left shift
  - ^ Exclusive or
  - & And

https://www.pg4e.com/code/hashmath.py

```
x = 15
y = ord('H')
print('x', x, format(x, '08b'))
print('y', y, format(y, '08b'))
print('x^y ', format(x^y, '08b'));
print('x&y ', format(x&y, '08b'));
print('x<<1', format(x<<1, '08b'));
```

```
$ python3 hashmath.py
x 15 00001111
y 72 01001000
x^y  01000111
x&y  00001000
x<<1 00011110
```

```
while True:
    txt = input("Enter a string: ")
    if len(txt) < 1: break

    hv = 0;
    for let in txt:

        hv = ((hv << 1) ^ ord(let)) & 0xffffff;

        if ( hv < 2000 ) :
            print(let,
              format(ord(let), '08b'),
              format(hv,'16b'),
              format(ord(let), '03d'), hv)
    print(format(hv, '08x'), hv)
```

```
$ python3 simplehash.py
Enter a string: Hello
H 01001000          1001000 072 72
e 01100101         11110101 101 245
l 01101100        110000110 108 390
l 01101100       1101100000 108 864
o 01101111      11010101111 111 1711
000006af 1711
Enter a string: hello
h 01101000          1101000 104 104
e 01100101         10110101 101 181
l 01101100        100000110 108 262
l 01101100       1001100000 108 608
o 01101111      10010101111 111 1199
000004af 1199
Enter a string: ehllo
e 01100101          1100101 101 101
h 01101000         10100010 104 162
l 01101100        100101000 108 296
l 01101100       1000111100 108 572
o 01101111      10000010111 111 1047
00000417 1047
```

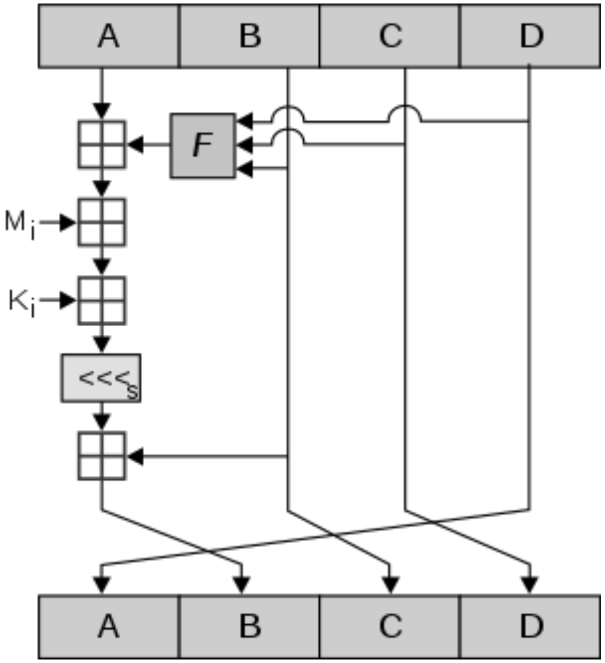
https://www.pg4e.com/code/simplehash.py

The Science/Math of Hashing

- Designing Hash Computations is serious work
- National Institute of Standards and Technology (NIST) runs multi-year "competitions" when new hashing algorithms are needed
- Sometimes algorithms have flaws that are detected years later and we deprecate them

The "Classic" Hash - MD5

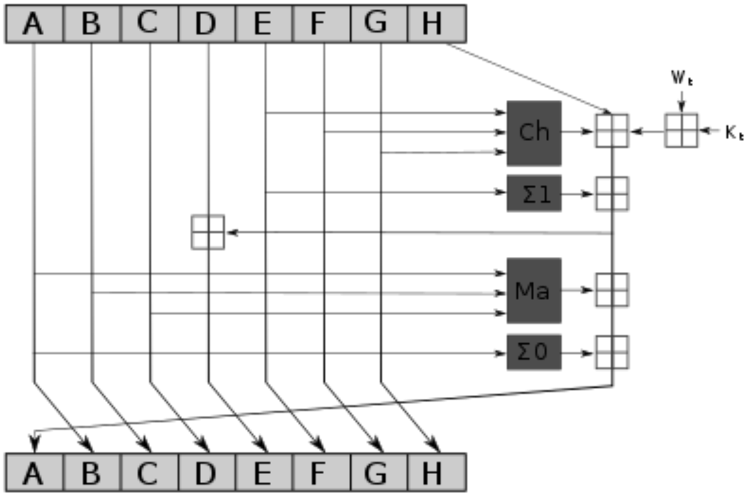
- 128 bit hash
- Widely implemented
- Broken for cryptography
  - Can alter data in transit without breaking a signature
  - Rainbow tables use forward computation and storage to reverse MD5 for short input strings (Password hashing)



https://en.wikipedia.org/wiki/MD5

SHA-256 – A Modern Hash

- A family of related hashes called "SHA-2"
- Created in 2001



```
discuss=> select md5('hello');
               md5
----------------------------------
 5d41402abc4b2a76b9719d911017c592

discuss=> select sha256('hello');
                               sha256
--------------------------------------------------------------------
 \x2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

Indexes and Performance

Lets Build a Web Crawler

- Approach
  - Retrieve a web page from our queue of web pages
  - Store the web page and look for outgoing links
  - Add the links we have not already read to a queue

- This table will be large and we will look up URLs quite often (100s per retrieved page)

How long is a URL?

```
CREATE TABLE cr1 (
  id SERIAL,
  url VARCHAR(128) UNIQUE,
  content TEXT
);
```

```
discuss=> insert into cr1(url)
discuss-> select repeat('Neon', 1000) || generate_series(1,5000);
ERROR:  value too long for type character varying(128)
discuss=>
```

```
CREATE TABLE cr2 (
  id SERIAL,
  url TEXT,
  content TEXT
);
```

```
discuss=> insert into cr2 (url)
discuss-> select repeat('Neon', 1000) || generate_series(1,5000);
INSERT 0 5000
discuss=> select pg_relation_size('cr2'), pg_indexes_size('cr2');
 pg_relation_size | pg_indexes_size
------------------+-----------------
           507904 |               0

discuss=> create unique index cr2_unique on cr2 (url);
CREATE INDEX
discuss=> select pg_relation_size('cr2'), pg_indexes_size('cr2');
 pg_relation_size | pg_indexes_size
------------------+-----------------
           507904 |          450560
```

```
discuss=> drop index cr2_unique;
DROP INDEX

discuss=> create unique index cr2_md5 on cr2 (md5(url));
CREATE INDEX
discuss=> select pg_relation_size('cr2'), pg_indexes_size('cr2');
 pg_relation_size | pg_indexes_size
------------------+-----------------
           507904 |          311296

discuss=> explain select * from cr2 where url='lemons';
 Seq Scan on cr2  (cost=0.00..124.50 rows=1 width=99)
   Filter: (url = 'lemons'::text)

discuss=> explain select * from cr2 where md5(url)=md5('lemons');
 Index Scan using cr2_md5 on cr2  (cost=0.28..8.30 rows=1 width=99)
   Index Cond: (md5(url) = '238ad51a7f1d25d991e6b51879d6b66d'::text)
```

```
discuss=> explain analyze select * from cr2 where md5(url)=md5('lemons');

Index Scan using cr2_md5 on cr2  (cost=0.28..8.30 rows=1 width=99)
 (actual time=0.118..0.118 rows=0 loops=1)
   Index Cond: (md5(url) = '238ad51a7f1d25d991e6b51879d6b66d'::text)
 Planning Time: 0.116 ms
 Execution Time: 0.142 ms

discuss=> explain analyze select * from cr2 where url='lemons';

Seq Scan on cr2  (cost=0.00..124.50 rows=1 width=99)
       (actual time=1.764..1.764 rows=0 loops=1)
   Filter: (url = 'lemons'::text)
   Rows Removed by Filter: 5000
 Planning Time: 0.067 ms
 Execution Time: 1.784 ms
```

Hashing with a separate column

```
CREATE TABLE cr3 (
  id SERIAL,
  url TEXT,
  url_md5 uuid unique,
  content TEXT
);
```

```
discuss=> insert into cr3 (url)
discuss-> select repeat('Neon', 1000) || generate_series(1,5000);
INSERT 0 5000
discuss=> update cr3 set url_md5 = md5(url)::uuid;
UPDATE 5000
discuss=> select pg_relation_size('cr3'), pg_indexes_size('cr3');
 pg_relation_size | pg_indexes_size
------------------+-----------------
          1097728 |          368640

discuss=> explain analyze select * from cr3 where url_md5=md5('lemons')::uuid;
  Index Scan using cr3_url_md5_key on cr3
   Index Cond: (url_md5 = '238ad51a-7f1d-25d9-91e6-b51879d6b66d'::uuid)
 Planning Time: 0.110 ms
 Execution Time: 0.030 ms
```

Index Strategies

```
CREATE TABLE cr2 (
  id SERIAL,
  url TEXT,
  content TEXT
);

No INDEX

Relation Size  507904
Index Size          0

SELECT       1.784 ms
```

```
CREATE TABLE cr2 (
  id SERIAL,
  url TEXT,
  content TEXT
);

MD5 Index on url

Relation Size  507904
Index Size     311296

SELECT        0.142ms
```

```
CREATE TABLE cr3 (
  id SERIAL,
  url TEXT,
  url_md5 uuid unique,
  content TEXT
);


Relation Size  1097728
Index Size      368640

SELECT        0.030 ms
```

The speed is for *exact match* SELECT statements – like one might do for a logical key on a table.

PostgreSQL Index Types

- B-Tree – Maintains order – Usually preferred
  - Helps on exact lookup, prefix lookup, <, >, range, sort
- HASH
  - Smaller - helps only on exact lookup
  - Not recommended before PostgreSQL 10

```
CREATE TABLE cr4 (
  id SERIAL,
  url TEXT,
  content TEXT
);

create index cr4_hash on cr4 using hash (url);
```

```
discuss=> select pg_relation_size('cr5'), pg_indexes_size('cr5');
 pg_relation_size | pg_indexes_size
------------------+-----------------
           507904 |          278528
(1 row)

discuss=> explain analyze select * from cr5 where url='lemons';
Bitmap Heap Scan on cr5
   Recheck Cond: (url = 'lemons'::text)
   ->  Bitmap Index Scan on cr5_hash
         Index Cond: (url = 'lemons'::text)
 Planning Time: 0.131 ms
 Execution Time: 0.045 ms
```

Hash Versus B-Tree

```
CREATE TABLE cr3 (
  id SERIAL,
  url TEXT,
  url_md5 uuid unique,
  content TEXT
);


Relation Size  1097728
Index Size      368640

SELECT        0.030 ms
```

```
CREATE TABLE cr4 (
  id SERIAL,
  url TEXT,
  content TEXT
);

Index HASH (url)

Relation Size   507904
Index Size      278528

SELECT        0.045 ms
```

The speed is for *exact match* SELECT statements – also HASH index cannot be unique.

Regular Expressions

https://www.py4e.com/lessons/regex

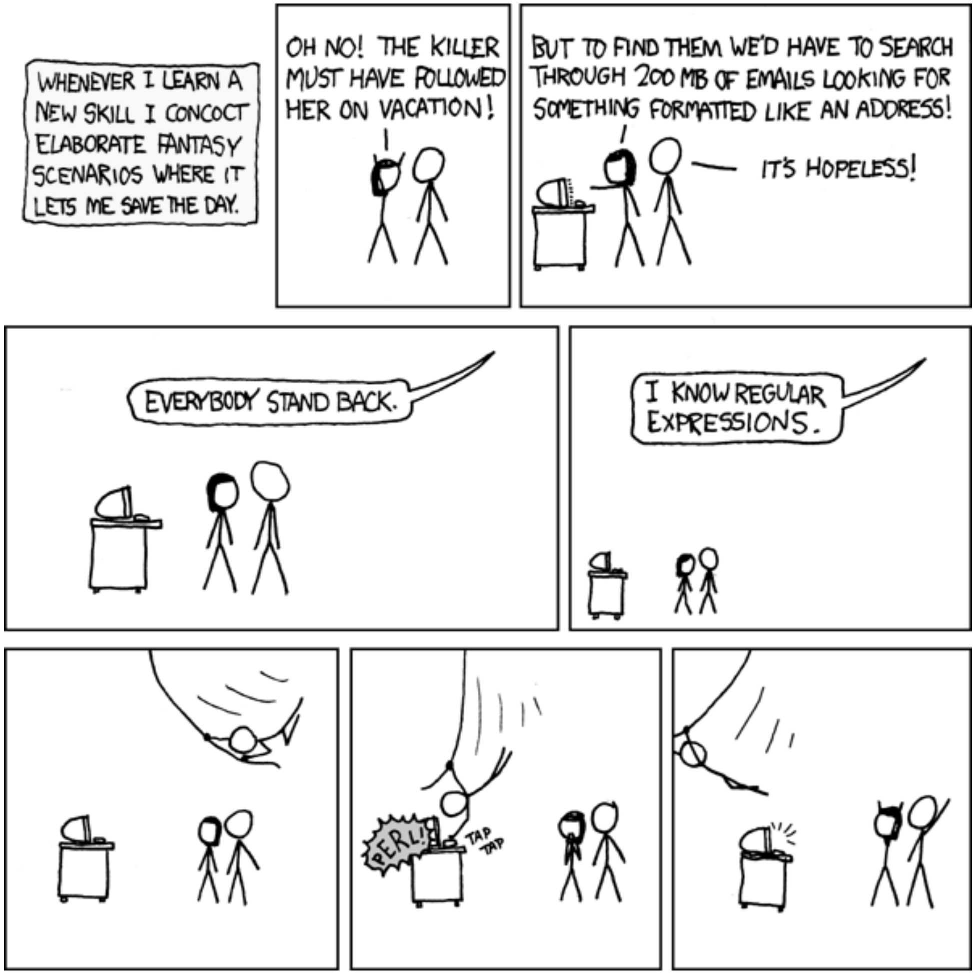
Regular Expressions

- A text based programming language
- Clever wild-card strings for matching and parsing text
- Widely available
  - Unix commands like "grep"
  - Virtually every programming language
  - Subtle differences across implementations
- PostgreSQL uses the POSIX variant

http://en.wikipedia.org/wiki/Regular_expression

Understanding Regular Expressions

- Very powerful and quite cryptic at first
- Fun once you understand them
- It is like learning a new programming language where marker characters are keywords
- It is kind of a throwback to the 1970's – very compact
- Lots of StackOverflow posts to look at ☺


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

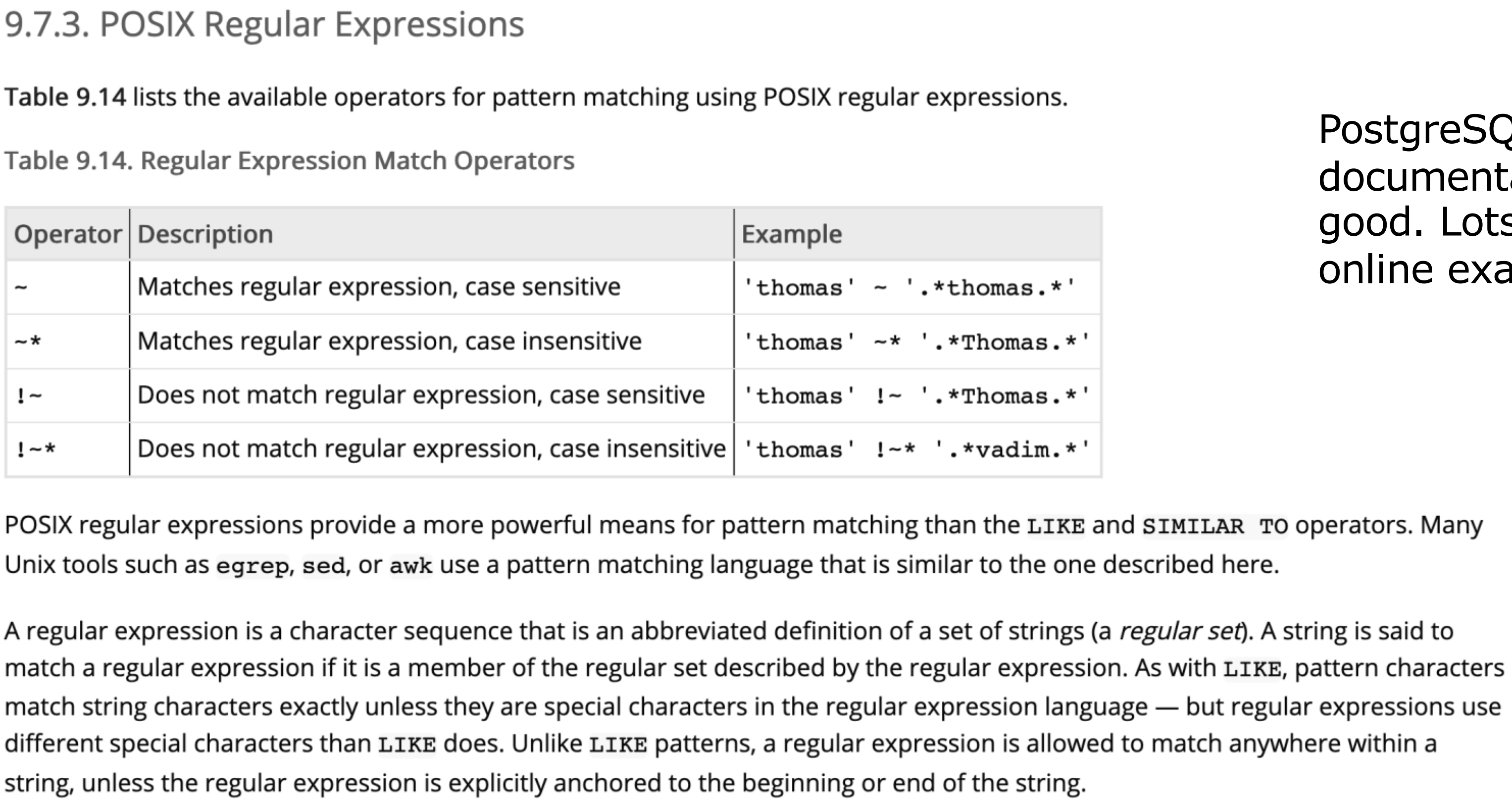
There is an XKCD for Everything

http://xkcd.com/208/

Regular Expression Quick Guide

```
^        Matches the beginning of a line
$        Matches the end of the line
.        Matches any character
*        Repeats a character zero or more times
*?       Repeats a character zero or more times (non-greedy)
+        Repeats a character one or more times
+?       Repeats a character one or more times (non-greedy)
[aeiou]  Matches a single character in the listed set
[^XYZ]   Matches a single character not in the listed set
[a-z0-9] The set of characters can include a range
(        Indicates where string extraction is to start
)        Indicates where string extraction is to end
```

https://www.pg4e.com/lectures/04-Text-Regex-Handout.txt



9.7.3. POSIX Regular Expressions

Table 9.14 lists the available operators for pattern matching using POSIX regular expressions.

Table 9.14. Regular Expression Match Operators

Operator	Description	Example
`~`	Matches regular expression, case sensitive	`'thomas' ~ '.*thomas.*'`
`~*`	Matches regular expression, case insensitive	`'thomas' ~* '.*Thomas.*'`
`!~`	Does not match regular expression, case sensitive	`'thomas' !~ '.*Thomas.*'`
`!~*`	Does not match regular expression, case insensitive	`'thomas' !~* '.*vadim.*'`

POSIX regular expressions provide a more powerful means for pattern matching than the `LIKE` and `SIMILAR TO` operators. Many Unix tools such as `egrep`, `sed`, or `awk` use a pattern matching language that is similar to the one described here.

A regular expression is a character sequence that is an abbreviated definition of a set of strings (a *regular set*). A string is said to match a regular expression if it is a member of the regular set described by the regular expression. As with `LIKE`, pattern characters match string characters exactly unless they are special characters in the regular expression language — but regular expressions use different special characters than `LIKE` does. Unlike `LIKE` patterns, a regular expression is allowed to match anywhere within a string, unless the regular expression is explicitly anchored to the beginning or end of the string.

PostgreSQL documentation is good. Lots of online examples.

https://www.postgresql.org/docs/11/functions-matching.html#FUNCTIONS-POSIX-REGEXP

Where Clause Operators

- ~ Matches
- ~* Matches case insensitive
- !~ Does not match
- !~* Does not match case insensitive
- Different than LIKE – Match anywhere
  - tweet ~ 'UMSI'
  - tweet LIKE '%UMSI%'

https://www.postgresql.org/docs/11/functions-matching.html#FUNCTIONS-POSIX-REGEXP

The simplest regex is like LIKE

```
CREATE TABLE em (id serial, primary key(id), email text);

INSERT INTO em (email) VALUES ('csev@umich.edu');
INSERT INTO em (email) VALUES ('coleen@umich.edu');
INSERT INTO em (email) VALUES ('sally@uiuc.edu');
INSERT INTO em (email) VALUES ('ted79@umuc.edu');
INSERT INTO em (email) VALUES ('glenn1@apple.com');
INSERT INTO em (email) VALUES ('nbody@apple.com');
```

```
discuss=> SELECT email FROM em WHERE email ~ 'umich';
      email
------------------
 csev@umich.edu
 coleen@umich.edu
```

```
discuss=> SELECT email from em;
      email
------------------
 csev@umich.edu
 coleen@umich.edu
 sally@uiuc.edu
 ted79@umuc.edu
 glenn1@apple.com
 nbody@apple.com
```

```
discuss=> SELECT email FROM em WHERE email ~ 'umich';
      email
------------------
 csev@umich.edu
 coleen@umich.edu

discuss=> SELECT email FROM em WHERE email ~ '^c';
      email
------------------
 csev@umich.edu
 coleen@umich.edu
```

```
discuss=> SELECT email from em;
      email
------------------
 csev@umich.edu
 coleen@umich.edu
 sally@uiuc.edu
 ted79@umuc.edu
 glenn1@apple.com
 nbody@apple.com
```

```
discuss=> SELECT email FROM em WHERE email ~ 'edu$';
      email
------------------
 csev@umich.edu
 coleen@umich.edu
 sally@uiuc.edu
 ted79@umuc.edu

discuss=> SELECT email FROM em WHERE email ~ '^[gnt]';
      email
-----------------
 ted79@umuc.edu
 glenn@apple.com
 nbody@apple.com
```

```
discuss=> SELECT email from em;
      email
------------------
 csev@umich.edu
 coleen@umich.edu
 sally@uiuc.edu
 ted79@umuc.edu
 glenn1@apple.com
 nbody@apple.com
```

```
discuss=> SELECT email FROM em WHERE email ~ '[0-9]';
      email
------------------
 ted79@umuc.edu
 glenn1@apple.com
(2 rows)

discuss=> SELECT email FROM em
discuss->    WHERE email ~ '[0-9][0-9]';
     email
----------------
 ted79@umuc.edu
```

```
discuss=> SELECT substring(email FROM '[0-9]+')
discuss->   FROM em WHERE email ~ '[0-9]';
 substring
-----------
 79
 1

discuss=> SELECT substring(email FROM '.+@(.*)$') FROM em;
 substring
-----------
 umich.edu
 umich.edu
 uiuc.edu
 umuc.edu
 apple.com
 apple.com
```

```
discuss=> SELECT DISTINCT substring(email FROM '.+@(.*)$') FROM em;
 substring
-----------
 apple.com
 uiuc.edu
 umuc.edu
 umich.edu

discuss=> SELECT substring(email FROM '.+@(.*)$'),
discuss->     count(substring(email FROM '.+@(.*)$'))
discuss-> FROM em GROUP BY substring(email FROM '.+@(.*)$');
 substring | count
-----------+-------
 apple.com |     2
 uiuc.edu  |     1
 umuc.edu  |     1
 umich.edu |     2
```

Multiple Matches

- The substring() gets the first match in a text column
- We can get an array of matches using regexp_matches()

```
CREATE TABLE tw (id serial, primary key(id), tweet text);

INSERT INTO tw (tweet) VALUES ('This is #SQL and #FUN stuff');
INSERT INTO tw (tweet) VALUES ('More people should learn #SQL from #UMSI');
INSERT INTO tw (tweet) VALUES ('#UMSI also teaches #PYTHON');
```

```
discuss=> SELECT tweet FROM tw;
                  tweet
------------------------------------------
 This is #SQL and #FUN stuff
 More people should learn #SQL from #UMSI
 #UMSI also teaches #PYTHON
```

```
discuss=> SELECT id, tweet FROM tw WHERE tweet ~ '#SQL';
 id |                  tweet
----+------------------------------------------
  1 | This is #SQL and #FUN stuff
  2 | More people should learn #SQL from #UMSI
```

```
discuss=> SELECT regexp_matches(tweet,'#([A-Za-z0-9_]+)', 'g') FROM tw;
 regexp_matches
----------------
 {SQL}
 {FUN}
 {SQL}
 {UMSI}
 {UMSI}
 {PYTHON}

discuss=> SELECT DISTINCT regexp_matches(tweet,'#([A-Za-z0-9_]+)', 'g')
discuss->   FROM tw;
 regexp_matches
----------------
 {FUN}
 {UMSI}
 {SQL}
 {PYTHON}
```

```
discuss=> SELECT tweet FROM tw;
                  tweet
------------------------------------------
 This is #SQL and #FUN stuff
 More people should learn #SQL from #UMSI
 #UMSI also teaches #PYTHON
(3 rows)

discuss=> SELECT id, regexp_matches(tweet,'#([A-Za-z0-9_]+)', 'g')
discuss->    FROM tw;
 id | regexp_matches
----+----------------
  1 | {SQL}
  1 | {FUN}
  2 | {SQL}
  2 | {UMSI}
  3 | {UMSI}
  3 | {PYTHON}
```

DEMO Reading Email

```
-- https://www.pg4e.com/lectures/mbox-short.txt

CREATE TABLE mbox (line TEXT);
\copy mbox FROM 'mbox-short.txt' with delimiter E'\007';

SELECT line FROM mbox WHERE line ~ '^From ';

SELECT substring(line, ' (.+@[^ ]+) ') FROM mbox WHERE line ~ '^From ';

SELECT substring(line, ' (.+@[^ ]+) '), count(substring(line, ' (.+@[^ ]+) '))
FROM mbox WHERE line ~ '^From '
GROUP BY substring(line, ' (.+@[^ ]+) ')
ORDER BY count(substring(line, ' (.+@[^ ]+) ')) DESC;

SELECT email, count(email) FROM
( SELECT substring(line, ' (.+@[^ ]+) ') AS email
  FROM mbox WHERE line ~ '^From '
) AS badsub
GROUP BY email ORDER BY count(email) DESC;
```

Summary

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Insert new Contributors and Translators here including names and dates

Continue new Contributors and Translators here