

SQL Style Guide

Purpose

- To provide clarity and guidance to product teams around the formatting of SQL statements and Stored Procedures in PostgreSQL databases
- To provide a common language for discussion of related topics

Scope

This document pertains primarily to the formatting of SQL statements and Stored Procedures for PostgreSQL use, but is not limited to PostgreSQL.

General

This style guide provides general guidelines for AWeber's use of SQL, including inside of stored procedures. These guidelines are designed to be compatible with Joe Celko's [SQL Programming Style](#) book to make adoption for teams who have already read that book easier.

- Use consistent and descriptive identifiers and names.
- Make judicious use of white space and indentation to make code easier to read.
- Store [ISO-8601](#) compliant time and date information (YYYY-MM-DD HH:MM:SS.SSSSST±hh:mm).
- Try to use only standard SQL functions instead of vendor specific functions for reasons of portability.
- Keep code succinct and devoid of redundant SQL—such as unnecessary quoting or parentheses or WHERE clauses that can otherwise be derived.
- Include comments in SQL code where necessary. Use the C style opening /* and closing */ where possible otherwise precede comments with -- and finish them with a new line.
- If you require a [surrogate key](#), use UUIDs instead of auto-generated SERIAL or SEQUENCE based counters.
- Prefer the use of SQL constants like CURRENT_TIMESTAMP over database functions like now() .

Examples

Do

```
1 SELECT file_hash -- stored ssdeep hash
2 FROM file_system
3 WHERE file_name = '.vimrc';
```

```
1  /* Updating the file record after writing to the file */
2  UPDATE file_system
3     SET file_modified_at = '1980-02-22 13:19:01.00000',
4         file_size = 209732
5  WHERE file_name = '.vimrc';
```

Avoid

- CamelCase—it is difficult to scan quickly.
- Descriptive prefixes or Hungarian notation such as `sp_` or `tbl`.
- Plurals—use the more natural collective term where possible instead. For example `staff` instead of `employees` or `people` instead of `individuals`.
- Quoted identifiers—if you must use them then stick to SQL92 double quotes for portability (you may need to configure your SQL server to support this depending on vendor).
- Object oriented design principles should not be applied to SQL or database structures.
- Creating [surrogate keys](#) for tables that have natural keys.

Naming conventions

General

- Ensure the name is unique and does not exist as a reserved keyword.
- Keep the length to a maximum of 30 bytes—in practice this is 30 characters unless you are using multi-byte character set.
- Names must begin with a letter and may not end with an underscore.
- Only use letters, numbers and underscores in names.
- Avoid the use of multiple consecutive underscores—these can be hard to read.
- Use underscores where you would naturally include a space in the name (first name becomes `first_name`).
- Avoid abbreviations and if you have to use them make sure they are commonly understood.
- Ensure your tables and fields have a semantic name that conveys the information it contains.
- If adding a common field like an account ID or subscriber ID, ensure the naming is consistent with other uses of the same data in the database (IE don't call it `list_id` in one place and `unit_id` in another).

Example

```
1  SELECT first_name
2  FROM staff;
```

Tables

- Use a collective name or, less ideally, a plural form. For example (in order of preference) `staff` and `employees`.
- Do not prefix with `tbl` or any other such descriptive prefix or Hungarian notation.
- Never give a table the same name as one of its columns and vice versa.
- Avoid, where possible, concatenating two table names together to create the name of a relationship table. Rather than `cars_mechanics` prefer `services`.

Columns

- Always use the singular name.
- Do not add a column with the same name as its table and vice versa.
- Always use lowercase except where it may make sense not to such as proper nouns.

Aliasing or correlations

- Should relate in some way to the object or expression they are aliasing.
- As a rule of thumb the correlation name should be the first letter of each word in the object's name.
- If there is already a correlation with the same name then append a number.
- Always include the `AS` keyword—makes it easier to read as it is explicit.
- For computed data (`SUM()` or `AVG()`) use the name you would give it were it a column defined in the schema.

Examples

```
1 SELECT first_name AS fn
2 FROM staff AS s1
3 JOIN students AS s2
4 ON s2.mentor_id = s1.staff_num;
```

```
1 SELECT SUM(s.monitor_tally) AS monitor_total
2 FROM staff AS s;
```

Stored procedures

- The name must contain a verb.
- Do not prefix with `sp_` or any other such descriptive prefix **or Hungarian notation**.

Uniform suffixes

The following suffixes have a universal meaning ensuring the columns can be read and understood easily from SQL code. Use the correct suffix where appropriate.

- `_status` —flag value or some other status of any type such as `publication_status` .
- `_total` —the total or sum of a collection of values.
- `_num` —denotes the field contains any kind of number.
- `_name` —signifies a name such as `first_name` .
- `_seq` —contains a contiguous sequence of values.
- `_at` —denotes a column that contains timestamp, such as `created_at`
- `_on` —denotes a column that contains the date of something.
- `_count` —a count.
- `_size` —the size of something such as a file size or clothing.
- `_addr` —an address for the record could be physical or intangible such as `ip_addr` .

Query syntax

Reserved words

Always use uppercase for the [reserved keywords](#) like `SELECT` and `WHERE` .

It is best to avoid the abbreviated keywords and use the full length ones where available (prefer `ABSOLUTE` to `ABS`).

Do not use database server specific keywords where an [ANSI SQL keyword](#) already exists performing the same function. This helps to make code more portable.

Example

```
1 SELECT model_num
2   FROM phones AS p
3  WHERE p.released_on >= '2014-09-30';
```

White space

To make the code easier to read it is important that the correct compliment of spacing is used. Do not crowd code or remove natural language spaces.

Spaces

Spaces should be used to line up the code so that the root keywords all end on the same character boundary. This forms a river down the middle making it easy for the readers eye to scan over the code and separate the keywords from the implementation detail. Rivers are [bad in typography](#), but helpful here.

Example

```
1 (SELECT f.species_name,
2     AVG(f.height) AS average_height, AVG(f.diameter) AS average_diameter
3   FROM flora AS f
4  WHERE f.species_name = 'Banksia')
```

```

5      OR f.species_name = 'Sheoak'
6      OR f.species_name = 'Wattle'
7  GROUP BY f.species_name, f.observation_date)
8
9  UNION ALL
10
11 (SELECT b.species_name,
12      AVG(b.height) AS average_height, AVG(b.diameter) AS average_diameter
13   FROM botanic_garden_flora AS b
14  WHERE b.species_name = 'Banksia'
15        OR b.species_name = 'Sheoak'
16        OR b.species_name = 'Wattle'
17  GROUP BY b.species_name, b.observation_date)

```

Notice that `SELECT`, `FROM`, etc. are all right aligned while the actual column names and implementation specific details are left aligned.

Although not exhaustive always include spaces:

- before and after equals (`=`)
- after commas (`,`)
- surrounding apostrophes (`'`) where not within parentheses or with a trailing comma or semicolon.

Example

```

1  SELECT a.title, a.released_on, a.recorded_on
2  FROM albums AS a
3  WHERE a.title = 'Charcoal Lane'
4        OR a.title = 'The New Danger';

```

Line spacing

Always include newlines/vertical space:

- before `AND` or `OR`
- after semicolons to separate queries for easier reading
- after each keyword definition
- after a comma when separating multiple columns into logical groups
- to separate code into related sections, which helps to ease the readability of large chunks of code.

Keeping all the keywords aligned to the righthand side and the values left aligned creates a uniform gap down the middle of query. It makes it much easier to scan the query definition over quickly too.

Examples

```

1  INSERT INTO albums (title, release_date, recording_date)
2  VALUES ('Charcoal Lane', '1990-01-01 01:01:01.00000', '1990-01-01 01:01:01.00000'),
3         ('The New Danger', '2008-01-01 01:01:01.00000', '1990-01-01 01:01:01.00000');

```

```

1 UPDATE albums
2   SET release_date = '1990-01-01 01:01:01.00000'
3   WHERE title = 'The New Danger';

```

```

1 SELECT a.title,
2        a.release_date, a.recording_date, a.production_date -- grouped dates together
3   FROM albums AS a
4  WHERE a.title = 'Charcoal Lane'
5        OR a.title = 'The New Danger';

```

Indentation

To ensure that SQL is readable it is important that standards of indentation are followed.

Joins

Joins should aligned with the river.

Example

```

1     SELECT r.last_name
2     FROM riders AS r
3  INNER JOIN bikes AS b
4     ON r.bike_vin_num = b.vin_num
5     AND b.engines > 2
6  INNER JOIN crew AS c
7     ON r.crew_chief_last_name = c.last_name
8     AND c.chief = 'Y';

```

Subqueries

Subqueries should be aligned to the right side of the river and then laid out using the same style as any other query. Sometimes it will make sense to have the closing parenthesis on a new line at the same character position as it's opening partner—this is especially true where you have nested subqueries.

Example

```

1 SELECT r.last_name,
2        (SELECT MAX(YEAR(championship_date))
3         FROM champions AS c
4         WHERE c.last_name = r.last_name
5         AND c.confirmed = 'Y') AS last_championship_year
6 FROM riders AS r
7 WHERE r.last_name IN
8        (SELECT c.last_name
9         FROM champions AS c
10        WHERE YEAR(championship_date) > '2008'
11        AND c.confirmed = 'Y');

```

Preferred formalisms

- Make use of `BETWEEN` where possible instead of combining multiple statements with `AND`.
- Similarly use `IN()` instead of multiple `OR` clauses.

- Where a value needs to be interpreted before leaving the database use the `CASE` expression. `CASE` statements can be nested to form more complex logical structures.
- Avoid the use of `UNION` clauses and temporary tables where possible. If the schema can be optimised to remove the reliance on these features then it most likely should be.

Example

```
1 SELECT CASE postcode
2     WHEN 'BN1' THEN 'Brighton'
3     WHEN 'EH1' THEN 'Edinburgh'
4     END AS city
5 FROM office_locations
6 WHERE country = 'United Kingdom'
7     AND opening_time BETWEEN 8 AND 9
8     AND postcode IN ('EH1', 'BN1', 'NN1', 'KW1')
```

Create syntax

When declaring schema information it is also important to maintain human readable code. To facilitate this ensure the column definitions are ordered and grouped where it makes sense to do so.

Indent column definitions by two (2) spaces within the `CREATE` definition.

Choosing data types

- Where possible do not use vendor specific data types — these are not portable and may not be available in older versions of the same vendor's software.
 - Within Postgres, use the following for guidance on data types to use:

Prefer	Over
TIMESTAMP WITH TIME ZONE	TIMESTAMPTZ
SMALLINT	INT2
INTEGER	INT4
BIGINT	INT8

- Only use `REAL` or `FLOAT` types where it is strictly necessary for floating point mathematics otherwise prefer `NUMERIC` and `DECIMAL` at all times. Floating point rounding errors are a

nuisance!

Specifying default values

- The default value must be the same type as the column—if a column is declared a `DECIMAL` do not provide an `INTEGER` default value.
- Default values must follow the data type declaration and come before any `NOT NULL` statement.

Data

Constraints and keys

Constraints and their subset, keys, are a very important component of any database definition. They can quickly become very difficult to read and reason about though so it is important that a standard set of guidelines are followed.

Choosing keys

Deciding the column(s) that will form the keys in the definition should be a carefully considered activity as it will effect performance and data integrity.

1. The key should be unique to some degree.
2. Consistency in terms of data type for the value across the schema and a lower likelihood of this changing in the future.
3. Can the value be validated against a standard format (such as one published by ISO)?
Encouraging conformity to point 2.
4. Keeping the key as simple as possible whilst not being scared to use compound keys where necessary.
5. Use natural keys when possible, avoiding surrogate keys.

It is a reasoned and considered balancing act to be performed at the definition of a database. Should requirements evolve in the future it is possible to make changes to the definitions to keep them up to date.

Defining constraints

Once the keys are decided it is possible to define them in the system using constraints along with field value validation.

General

- Tables must have at least one key to be complete and useful.
- Constraints should be given a custom name excepting `UNIQUE`, `PRIMARY KEY` and `FOREIGN KEY` where the database vendor will generally supply sufficiently intelligible names automatically.

Layout and order

- Specify the primary key first right after the `CREATE TABLE` statement.
- Constraints should be defined directly beneath the column they correspond to. Indent the constraint so that it aligns to the right of the column name.
- If it is a multi-column constraint then consider putting it as close to both column definitions as possible and where this is difficult as a last resort include them at the end of the `CREATE TABLE` definition.
- If it is a table level constraint that applies to the entire table then it should also appear at the end.
- Use alphabetical order where `ON DELETE` comes before `ON UPDATE`.
- If it make senses to do so align each aspect of the query on the same character position. For example all `NOT NULL` definitions could start at the same character position. This is not hard and fast, but it certainly makes the code much easier to scan and read.

Validation

- Use `LIKE` and `SIMILAR TO` constraints to ensure the integrity of strings where the format is known.
- Where the ultimate range of a numerical value is known it must be written as a range `CHECK()` to prevent incorrect values entering the database or the silent truncation of data too large to fit the column definition. In the least it should check that the value is greater than zero in most cases.
- `CHECK()` constraints should be kept in separate clauses to ease debugging.

Example

```
1 CREATE TABLE staff (  
2   staff_num      INTEGER NOT NULL,  
3   first_name     TEXT    NOT NULL,  
4   pens_in_drawer INT(2)  NOT NULL,  
5                 CONSTRAINT pens_in_drawer_range  
6                 CHECK(pens_in_drawer >= 1 AND pens_in_drawer < 100),  
7   PRIMARY KEY (staff_num));
```

Designs to avoid

- Object oriented design principles do not effectively translate to relational database designs—avoid this pitfall.
- Placing the value in one column and the units in another column. The column should make the units self evident to prevent the requirement to combine columns again later in the application. Use `CHECK()` to ensure valid data is inserted into the column.
- [EAV \(Entity Attribute Value\)](#) tables—use a specialist product intended for handling such schema-less data instead.

- Splitting up data that should be in one table across many because of arbitrary concerns such as time-based archiving or location in a multi-national organisation. Later queries must then work across multiple tables with `UNION` rather than just simply querying one table.

User-Defined Functions (Stored Procedures)

Language

The following list enumerates what languages user-defined functions should be written in, in order of preference:

1. [SQL](#)
2. [PL/pgSQL](#)
3. [PL/Python](#)

Avoid writing stored procedures in C or PL/Perl if possible.

General

- Queries within user-defined functions should adhere to the style outlined in this guide, without regard to the language the function is written in.
- The style outlined in this document should be applied to the DDL that is used for creating the function
- The DDL for creating the function should use `CREATE OR REPLACE FUNCTION`
- Dollar-quoting should be used to define the body of the function within the SQL statement
- Spacing rules should be applied on a per-language basis.
 - PL/pgSQL functions should indent using two spaces, starting at column 0.
 - PL/Python functions should indent using four spaces, following PEP-8 rules
- Functions that return one or more rows of data should return `RECORD` with a documented return structure, instead of creating per-function specific return `TYPE` s
- Avoid using temporary tables if possible

Examples

```
1 CREATE OR REPLACE FUNCTION daily_autoblocked(character varying(50)) RETURNS SETOF character
2 varying(50) AS $$
3 SELECT email_user
4 FROM blocked_emails
5 WHERE email_domain = $1
6 AND added_at > CURRENT_TIMESTAMP - INTERVAL '1 days'
7 AND unit_id = 0 AND note ~ 'Autoblock'
8 ORDER BY email_user;
9 $$ LANGUAGE SQL
10 SECURITY DEFINER
SET SEARCH_PATH TO public, pg_temp;
```

```

1 CREATE OR REPLACE FUNCTION mikkoo.reset_queues() RETURNS void AS $$
2 DECLARE
3     queues text[] := ARRAY['public_accounts', 'public_leads', 'public_lead_data_desc'];
4 BEGIN
5     FOR i IN array_lower(queues, 1) .. array_upper(queues, 1) LOOP
6         BEGIN
7             PERFORM * FROM pgq.drop_queue(queues[i]);
8             EXCEPTION WHEN raise_exception THEN
9                 RAISE NOTICE '% does not exist', queues[i];
10        END;
11        PERFORM * FROM pgq.create_queue(queues[i]);
12    END LOOP;
13 END;
14 $$ LANGUAGE plpgsql;

```

Appendix

Reserved keyword reference

A list of ANSI SQL (92, 99 and 2003), MySQL 3 to 5.x, PostgreSQL 8.1, MS SQL Server 2000, MS ODBC and Oracle 10.2 reserved keywords.

Reserved Keywords

```

1 A
2 ABORT
3 ABS
4 ABSOLUTE
5 ACCESS
6 ACTION
7 ADA
8 ADD
9 ADMIN
10 AFTER
11 AGGREGATE
12 ALIAS
13 ALL
14 ALLOCATE
15 ALSO
16 ALTER
17 ALWAYS
18 ANALYSE
19 ANALYZE
20 AND
21 ANY
22 ARE
23 ARRAY
24 AS
25 ASC
26 ASENSITIVE
27 ASSERTION
28 ASSIGNMENT
29 ASYMMETRIC
30 AT
31 ATOMIC
32 ATTRIBUTE
33 ATTRIBUTES
34 AUDIT
35 AUTHORIZATION
36 AUTO_INCREMENT
37 AVG

```

38	AVG_ROW_LENGTH
39	BACKUP
40	BACKWARD
41	BEFORE
42	BEGIN
43	BERNOULLI
44	BETWEEN
45	BIGINT
46	BINARY
47	BIT
48	BIT_LENGTH
49	BITVAR
50	BLOB
51	BOOL
52	BOOLEAN
53	BOTH
54	BREADTH
55	BREAK
56	BROWSE
57	BULK
58	BY
59	C
60	CACHE
61	CALL
62	CALLED
63	CARDINALITY
64	CASCADE
65	CASCADEED
66	CASE
67	CAST
68	CATALOG
69	CATALOG_NAME
70	CEIL
71	CEILING
72	CHAIN
73	CHANGE
74	CHAR
75	CHAR_LENGTH
76	CHARACTER
77	CHARACTER_LENGTH
78	CHARACTER_SET_CATALOG
79	CHARACTER_SET_NAME
80	CHARACTER_SET_SCHEMA
81	CHARACTERISTICS
82	CHARACTERS
83	CHECK
84	CHECKED
85	CHECKPOINT
86	CHECKSUM
87	CLASS
88	CLASS_ORIGIN
89	CLOB
90	CLOSE
91	CLUSTER
92	CLUSTERED
93	COALESCE
94	COBOL
95	COLLATE
96	COLLATION
97	COLLATION_CATALOG
98	COLLATION_NAME
99	COLLATION_SCHEMA
100	COLLECT
101	COLUMN
102	COLUMN_NAME
103	COLUMNS
104	COMMAND_FUNCTION

105	COMMAND_FUNCTION_CODE
106	COMMENT
107	COMMIT
108	COMMITTED
109	COMPLETION
110	COMPRESS
111	COMPUTE
112	CONDITION
113	CONDITION_NUMBER
114	CONNECT
115	CONNECTION
116	CONNECTION_NAME
117	CONSTRAINT
118	CONSTRAINT_CATALOG
119	CONSTRAINT_NAME
120	CONSTRAINT_SCHEMA
121	CONSTRAINTS
122	CONSTRUCTOR
123	CONTAINS
124	CONTAINSTABLE
125	CONTINUE
126	CONVERSION
127	CONVERT
128	COPY
129	CORR
130	CORRESPONDING
131	COUNT
132	COVAR_POP
133	COVAR_SAMP
134	CREATE
135	CREATEDB
136	CREATEROLE
137	CREATEUSER
138	CROSS
139	CSV
140	CUBE
141	CUME_DIST
142	CURRENT
143	CURRENT_DATE
144	CURRENT_DEFAULT_TRANSFORM_GROUP
145	CURRENT_PATH
146	CURRENT_ROLE
147	CURRENT_TIME
148	CURRENT_TIMESTAMP
149	CURRENT_TRANSFORM_GROUP_FOR_TYPE
150	CURRENT_USER
151	CURSOR
152	CURSOR_NAME
153	CYCLE
154	DATA
155	DATABASE
156	DATABASES
157	DATE
158	DATETIME
159	DATETIME_INTERVAL_CODE
160	DATETIME_INTERVAL_PRECISION
161	DAY
162	DAY_HOUR
163	DAY_MICROSECOND
164	DAY_MINUTE
165	DAY_SECOND
166	DAYOFMONTH
167	DAYOFWEEK
168	DAYOFYEAR
169	DBCC
170	DEALLOCATE
171	DEC

172	DECIMAL
173	DECLARE
174	DEFAULT
175	DEFAULTS
176	DEFERRABLE
177	DEFERRED
178	DEFINED
179	DEFINER
180	DEGREE
181	DELAY_KEY_WRITE
182	DELAYED
183	DELETE
184	DELIMITER
185	DELIMITERS
186	DENSE_RANK
187	DENY
188	DEPTH
189	DEREF
190	DERIVED
191	DESC
192	DESCRIBE
193	DESCRIPTOR
194	DESTROY
195	DESTRUCTOR
196	DETERMINISTIC
197	DIAGNOSTICS
198	DICTIONARY
199	DISABLE
200	DISCONNECT
201	DISK
202	DISPATCH
203	DISTINCT
204	DISTINCTROW
205	DISTRIBUTED
206	DIV
207	DO
208	DOMAIN
209	DOUBLE
210	DROP
211	DUAL
212	DUMMY
213	DUMP
214	DYNAMIC
215	DYNAMIC_FUNCTION
216	DYNAMIC_FUNCTION_CODE
217	EACH
218	ELEMENT
219	ELSE
220	ELSEIF
221	ENABLE
222	ENCLOSED
223	ENCODING
224	ENCRYPTED
225	END
226	END-EXEC
227	ENUM
228	EQUALS
229	ERRLVL
230	ESCAPE
231	ESCAPED
232	EVERY
233	EXCEPT
234	EXCEPTION
235	EXCLUDE
236	EXCLUDING
237	EXCLUSIVE
238	EXEC

239	EXECUTE
240	EXISTING
241	EXISTS
242	EXIT
243	EXP
244	EXPLAIN
245	EXTERNAL
246	EXTRACT
247	FALSE
248	FETCH
249	FIELDS
250	FILE
251	FILLFACTOR
252	FILTER
253	FINAL
254	FIRST
255	FLOAT
256	FLOAT4
257	FLOAT8
258	FLOOR
259	FLUSH
260	FOLLOWING
261	FOR
262	FORCE
263	FOREIGN
264	FORTRAN
265	FORWARD
266	FOUND
267	FREE
268	FREETEXT
269	FREETEXTTABLE
270	FREEZE
271	FROM
272	FULL
273	FULLTEXT
274	FUNCTION
275	FUSION
276	G
277	GENERAL
278	GENERATED
279	GET
280	GLOBAL
281	GO
282	GOTO
283	GRANT
284	GRANTED
285	GRANTS
286	GREATEST
287	GROUP
288	GROUPING
289	HANDLER
290	HAVING
291	HEADER
292	HEAP
293	HIERARCHY
294	HIGH_PRIORITY
295	HOLD
296	HOLDLOCK
297	HOST
298	HOSTS
299	HOURL
300	HOURL_MICROSECOND
301	HOURL_MINUTE
302	HOURL_SECOND
303	IDENTIFIED
304	IDENTITY
305	IDENTITY_INSERT

306	IDENTITYCOL
307	IF
308	IGNORE
309	ILIKE
310	IMMEDIATE
311	IMMUTABLE
312	IMPLEMENTATION
313	IMPLICIT
314	IN
315	INCLUDE
316	INCLUDING
317	INCREMENT
318	INDEX
319	INDICATOR
320	INFILE
321	INFIX
322	INHERIT
323	INHERITS
324	INITIAL
325	INITIALIZE
326	INITIALLY
327	INNER
328	INOUT
329	INPUT
330	INSENSITIVE
331	INSERT
332	INSERT_ID
333	INSTANCE
334	INSTANTIABLE
335	INSTEAD
336	INT
337	INT1
338	INT2
339	INT3
340	INT4
341	INT8
342	INTEGER
343	INTERSECT
344	INTERSECTION
345	INTERVAL
346	INTO
347	INVOKER
348	IS
349	ISAM
350	ISNULL
351	ISOLATION
352	ITERATE
353	JOIN
354	K
355	KEY
356	KEY_MEMBER
357	KEY_TYPE
358	KEYS
359	KILL
360	LANCOMPILER
361	LANGUAGE
362	LARGE
363	LAST
364	LAST_INSERT_ID
365	LATERAL
366	LEADING
367	LEAST
368	LEAVE
369	LEFT
370	LENGTH
371	LESS
372	LEVEL

373	LIKE
374	LIMIT
375	LINENO
376	LINES
377	LISTEN
378	LN
379	LOAD
380	LOCAL
381	LOCALTIME
382	LOCALTIMESTAMP
383	LOCATION
384	LOCATOR
385	LOCK
386	LOGIN
387	LOGS
388	LONG
389	LONGBLOB
390	LONGTEXT
391	LOOP
392	LOW_PRIORITY
393	LOWER
394	M
395	MAP
396	MATCH
397	MATCHED
398	MAX
399	MAX_ROWS
400	MAXEXTENTS
401	MAXVALUE
402	MEDIUMBLOB
403	MEDIUMINT
404	MEDIUMTEXT
405	MEMBER
406	MERGE
407	MESSAGE_LENGTH
408	MESSAGE_OCTET_LENGTH
409	MESSAGE_TEXT
410	METHOD
411	MIDDLEINT
412	MIN
413	MIN_ROWS
414	MINUS
415	MINUTE
416	MINUTE_MICROSECOND
417	MINUTE_SECOND
418	MINVALUE
419	MLSLABEL
420	MOD
421	MODE
422	MODIFIES
423	MODIFY
424	MODULE
425	MONTH
426	MONTHNAME
427	MORE
428	MOVE
429	MULTISET
430	MUMPS
431	MYISAM
432	NAME
433	NAMES
434	NATIONAL
435	NATURAL
436	NCHAR
437	NCLOB
438	NESTING
439	NEW

440	NEXT
441	NO
442	NO_WRITE_TO_BINLOG
443	NOAUDIT
444	NOCHECK
445	NOCOMPRESS
446	NOCREATEDB
447	NOCREATEROLE
448	NOCREATEUSER
449	NOINHERIT
450	NOLOGIN
451	NONCLUSTERED
452	NONE
453	NORMALIZE
454	NORMALIZED
455	NOSUPERUSER
456	NOT
457	NOTHING
458	NOTIFY
459	NOTNULL
460	NOWAIT
461	NULL
462	NULLABLE
463	NULLIF
464	NULLS
465	NUMBER
466	NUMERIC
467	OBJECT
468	OCTET_LENGTH
469	OCTETS
470	OF
471	OFF
472	OFFLINE
473	OFFSET
474	OFFSETS
475	OIDS
476	OLD
477	ON
478	ONLINE
479	ONLY
480	OPEN
481	OPENDATASOURCE
482	OPENQUERY
483	OPENROWSET
484	OPENXML
485	OPERATION
486	OPERATOR
487	OPTIMIZE
488	OPTION
489	OPTIONALLY
490	OPTIONS
491	OR
492	ORDER
493	ORDERING
494	ORDINALITY
495	OTHERS
496	OUT
497	OUTER
498	OUTFILE
499	OUTPUT
500	OVER
501	OVERLAPS
502	OVERLAY
503	OVERRIDING
504	OWNER
505	PACK_KEYS
506	PAD

507	PARAMETER
508	PARAMETER_MODE
509	PARAMETER_NAME
510	PARAMETER_ORDINAL_POSITION
511	PARAMETER_SPECIFIC_CATALOG
512	PARAMETER_SPECIFIC_NAME
513	PARAMETER_SPECIFIC_SCHEMA
514	PARAMETERS
515	PARTIAL
516	PARTITION
517	PASCAL
518	PASSWORD
519	PATH
520	PCTFREE
521	PERCENT
522	PERCENT_RANK
523	PERCENTILE_CONT
524	PERCENTILE_DISC
525	PLACING
526	PLAN
527	PLI
528	POSITION
529	POSTFIX
530	POWER
531	PRECEDING
532	PRECISION
533	PREFIX
534	PREORDER
535	PREPARE
536	PREPARED
537	PRESERVE
538	PRIMARY
539	PRINT
540	PRIOR
541	PRIVILEGES
542	PROC
543	PROCEDURAL
544	PROCEDURE
545	PROCESS
546	PROCESSLIST
547	PUBLIC
548	PURGE
549	QUOTE
550	RAID0
551	RAISERROR
552	RANGE
553	RANK
554	RAW
555	READ
556	READS
557	READTEXT
558	REAL
559	RECHECK
560	RECONFIGURE
561	RECURSIVE
562	REF
563	REFERENCES
564	REFERENCING
565	REGEXP
566	REGR_AVGX
567	REGR_AVGY
568	REGR_COUNT
569	REGR_INTERCEPT
570	REGR_R2
571	REGR_SLOPE
572	REGR_SXX
573	REGR_SXY

574	REGR_SYX
575	REINDEX
576	RELATIVE
577	RELEASE
578	RELOAD
579	RENAME
580	REPEAT
581	REPEATABLE
582	REPLACE
583	REPLICATION
584	REQUIRE
585	RESET
586	RESIGNAL
587	RESOURCE
588	RESTART
589	RESTORE
590	RESTRICT
591	RESULT
592	RETURN
593	RETURNED_CARDINALITY
594	RETURNED_LENGTH
595	RETURNED_OCTET_LENGTH
596	RETURNED_SQLSTATE
597	RETURNS
598	REVOKE
599	RIGHT
600	RLIKE
601	ROLE
602	ROLLBACK
603	ROLLUP
604	ROUTINE
605	ROUTINE_CATALOG
606	ROUTINE_NAME
607	ROUTINE_SCHEMA
608	ROW
609	ROW_COUNT
610	ROW_NUMBER
611	ROWCOUNT
612	ROWGUIDCOL
613	ROWID
614	ROWNUM
615	ROWS
616	RULE
617	SAVE
618	SAVEPOINT
619	SCALE
620	SCHEMA
621	SCHEMA_NAME
622	SCHEMAS
623	SCOPE
624	SCOPE_CATALOG
625	SCOPE_NAME
626	SCOPE_SCHEMA
627	SCROLL
628	SEARCH
629	SECOND
630	SECOND_MICROSECOND
631	SECTION
632	SECURITY
633	SELECT
634	SELF
635	SENSITIVE
636	SEPARATOR
637	SEQUENCE
638	SERIALIZABLE
639	SERVER_NAME
640	SESSION

641	SESSION_USER
642	SET
643	SETOF
644	SETS
645	SETUSER
646	SHARE
647	SHOW
648	SHUTDOWN
649	SIGNAL
650	SIMILAR
651	SIMPLE
652	SIZE
653	SMALLINT
654	SOME
655	SONAME
656	SOURCE
657	SPACE
658	SPATIAL
659	SPECIFIC
660	SPECIFIC_NAME
661	SPECIFICTYPE
662	SQL
663	SQL_BIG_RESULT
664	SQL_BIG_SELECTS
665	SQL_BIG_TABLES
666	SQL_CALC_FOUND_ROWS
667	SQL_LOG_OFF
668	SQL_LOG_UPDATE
669	SQL_LOW_PRIORITY_UPDATES
670	SQL_SELECT_LIMIT
671	SQL_SMALL_RESULT
672	SQL_WARNINGS
673	SQLCA
674	SQLCODE
675	SQLERROR
676	SQLEXCEPTION
677	SQLSTATE
678	SQLWARNING
679	SQRT
680	SSL
681	STABLE
682	START
683	STARTING
684	STATE
685	STATEMENT
686	STATIC
687	STATISTICS
688	STATUS
689	STDDEV_POP
690	STDDEV_SAMP
691	STDIN
692	STDOUT
693	STORAGE
694	STRAIGHT_JOIN
695	STRICT
696	STRING
697	STRUCTURE
698	STYLE
699	SUBCLASS_ORIGIN
700	SUBLIST
701	SUBMULTISET
702	SUBSTRING
703	SUCCESSFUL
704	SUM
705	SUPERUSER
706	SYMMETRIC
707	SYNONYM

708	SYSDATE
709	SYSID
710	SYSTEM
711	SYSTEM_USER
712	TABLE
713	TABLE_NAME
714	TABLES
715	TABLESAMPLE
716	TABLESPACE
717	TEMP
718	TEMPLATE
719	TEMPORARY
720	TERMINATE
721	TERMINATED
722	TEXT
723	TEXTSIZE
724	THAN
725	THEN
726	TIES
727	TIME
728	TIMESTAMP
729	TIMEZONE_HOUR
730	TIMEZONE_MINUTE
731	TINYBLOB
732	TINYINT
733	TINYTEXT
734	TO
735	TOAST
736	TOP
737	TOP_LEVEL_COUNT
738	TRAILING
739	TRAN
740	TRANSACTION
741	TRANSACTION_ACTIVE
742	TRANSACTIONS_COMMITTED
743	TRANSACTIONS_ROLLED_BACK
744	TRANSFORM
745	TRANSFORMS
746	TRANSLATE
747	TRANSLATION
748	TREAT
749	TRIGGER
750	TRIGGER_CATALOG
751	TRIGGER_NAME
752	TRIGGER_SCHEMA
753	TRIM
754	TRUE
755	TRUNCATE
756	TRUSTED
757	TSEQUAL
758	TYPE
759	UESCAPE
760	UID
761	UNBOUNDED
762	UNCOMMITTED
763	UNDER
764	UNDO
765	UNENCRYPTED
766	UNION
767	UNIQUE
768	UNKNOWN
769	UNLISTEN
770	UNLOCK
771	UNNAMED
772	UNNEST
773	UNSIGNED
774	UNTIL

```
775 UPDATE
776 UPDATETEXT
777 UPPER
778 USAGE
779 USE
780 USER
781 USER_DEFINED_TYPE_CATALOG
782 USER_DEFINED_TYPE_CODE
783 USER_DEFINED_TYPE_NAME
784 USER_DEFINED_TYPE_SCHEMA
785 USING
786 UTC_DATE
787 UTC_TIME
788 UTC_TIMESTAMP
789 VACUUM
790 VALID
791 VALIDATE
792 VALIDATOR
793 VALUE
794 VALUES
795 VAR_POP
796 VAR_SAMP
797 VARBINARY
798 VARCHAR
799 VARCHAR2
800 VARCHARACTER
801 VARIABLE
802 VARIABLES
803 VARYING
804 VERBOSE
805 VIEW
806 VOLATILE
807 WAITFOR
808 WHEN
809 WHENEVER
810 WHERE
811 WHILE
812 WIDTH_BUCKET
813 WINDOW
814 WITH
815 WITHIN
816 WITHOUT
817 WORK
818 WRITE
819 WRITETEXT
820 X509
821 XOR
822 YEAR
823 YEAR_MONTH
824 ZEROFILL
825 ZONE
```

Attribution

This guide was originally based on the SQL Style Guide by [Simon Holywell](#) is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#), which is available at <http://www.sqlstyle.guide>.