

# SQLITE - PRAGMA

[http://www.tutorialspoint.com/sqlite/sqlite\\_pragma.htm](http://www.tutorialspoint.com/sqlite/sqlite_pragma.htm)

Copyright © tutorialspoint.com

The SQLite **PRAGMA** command is a special command to be used to control various environmental variables and state flags within the SQLite environment. A PRAGMA value can be read and it can also be set based on requirements.

## Syntax:

To query the current PRAGMA value, just provide the name of the pragma:

```
PRAGMA pragma_name;
```

To set a new value for PRAGMA, you will use the following syntax:

```
PRAGMA pragma_name = value;
```

The set mode can be either the name or the integer equivalent but the returned value will always be an integer.

## auto\_vacuum Pragma

The **auto\_vacuum** pragma gets or sets the auto-vacuum mode. Following is the simple syntax:

```
PRAGMA [database.]auto_vacuum;  
PRAGMA [database.]auto_vacuum = mode;
```

Where **mode** can be any of the following:

Pragma Value	Description
0 or NONE	Auto-vacuum is disabled. This is default mode which means that a database file will never shrink in size unless it is manually vacuumed using the VACUUM command.
1 or FULL	Auto-vacuum is enabled and fully automatic which allows a database file to shrink as data is removed from the database.
2 or INCREMENTAL	Auto-vacuum is enabled but must be manually activated. In this mode the reference data is maintained, but free pages are simply put on the free list. These pages can be recovered using the <b>incremental_vacuum pragma</b> any time.

## cache\_size Pragma

The **cache\_size** pragma can get or temporarily set the maximum size of the in-memory page cache. Following is the simple syntax:

```
PRAGMA [database.]cache_size;  
PRAGMA [database.]cache_size = pages;
```

The **pages** value represents the number of pages in the cache. The built-in page cache has a default size of 2,000 pages and a minimum size of 10 pages.

## case\_sensitive\_like Pragma

The **case\_sensitive\_like** pragma controls the case-sensitivity of the built-in LIKE expression. By default, this pragma is false which means that the built-in LIKE operator ignores letter case. Following is the simple syntax:

```
PRAGMA case_sensitive_like = [true|false];
```

There is no way to query for the current state of this pragma.

## count\_changes Pragma

The **count\_changes** pragma gets or sets the return value of data manipulation statements such as INSERT, UPDATE and DELETE. Following is the simple syntax:

```
PRAGMA count_changes;  
PRAGMA count_changes = [true|false];
```

By default, this pragma is false and these statements do not return anything. If set to true, each of the mentioned statement will return an one-column, one-row table consisting of a single integer value indicating impacted rows by the operation.

## database\_list Pragma

The **database\_list** pragma will be used to list down all the databases attached. Following is the simple syntax:

```
PRAGMA database_list;
```

This pragma will return a three-column table with one row per open or attached database giving database sequence number, its name and file associated.

## encoding Pragma

The **encoding** pragma controls how strings are encoded and stored in a database file. Following is the simple syntax:

```
PRAGMA encoding;  
PRAGMA encoding = format;
```

The format value can be one of UTF-8, UTF-16le, or UTF-16be.

## freelist\_count Pragma

The **freelist\_count** pragma returns a single integer indicating how many database pages are currently marked as free and available. Following is the simple syntax:

```
PRAGMA [database.]freelist_count;
```

The format value can be one of UTF-8, UTF-16le, or UTF-16be.

## index\_info Pragma

The **index\_info** pragma returns information about a database index. Following is the simple syntax:

```
PRAGMA [database.]index_info( index_name );
```

The result set will contain one row for each column contained in the index giving column sequence, column index with-in table and column name.

## index\_list Pragma

The **index\_list** pragma lists all of the indexes associated with a table. Following is the simple syntax:

```
PRAGMA [database.]index_list( table_name );
```

The result set will contain one row for each index giving index sequence, index name and flag indicating whether index is unique or not.

## journal\_mode Pragma

The **journal\_mode** pragma gets or sets the journal mode which controls how the journal file is stored and processed. Following is the simple syntax:

```
PRAGMA journal_mode;  
PRAGMA journal_mode = mode;  
PRAGMA database.journal_mode;  
PRAGMA database.journal_mode = mode;
```

There are five supported journal modes:

Pragma Value	Description
DELETE	This is default mode. Here at the conclusion of a transaction, the journal file is deleted.
TRUNCATE	The journal file is truncated to a length of zero bytes.
PERSIST	The journal file is left in place, but the header is overwritten to indicate the journal is no longer valid.
MEMORY	The journal record is held in memory, rather than on disk.
OFF	No journal record is kept.

## max\_page\_count Pragma

The **max\_page\_count** pragma gets or sets the maximum allowed page count for a database. Following is the simple syntax:

```
PRAGMA [database.]max_page_count;  
PRAGMA [database.]max_page_count = max_page;
```

The default value is 1,073,741,823 which is one giga-page which means if the default 1 KB page size, this allows databases to grow up to one terabyte.

## page\_count Pragma

The **page\_count** pragma returns the current number of pages in database. Following is the simple syntax:

```
PRAGMA [database.]page_count;
```

The size of the database file should be `page_count * page_size`.

## page\_size Pragma

The **page\_size** pragma gets or sets the size of the database pages. Following is the simple syntax:

```
PRAGMA [database.]page_size;  
PRAGMA [database.]page_size = bytes;
```

By default, the allowed sizes are 512, 1024, 2048, 4096, 8192, 16384, and 32768 bytes. The only way to alter the page size on an existing database is to set the page size and then immediately VACUUM the database.

## parser\_trace Pragma

The **parser\_trace** pragma controls printing the debugging state as it parses SQL commands. Following is the simple syntax:

```
PRAGMA parser_trace = [true|false];
```

By default it is set to false but when enabled by setting it to true, the SQL parser will print its state as it parses SQL commands.

## recursive\_triggers Pragma

The **recursive\_triggers** pragma gets or sets the recursive trigger functionality. If recursive triggers are not enabled, a trigger action will not fire another trigger. Following is the simple syntax:

```
PRAGMA recursive_triggers;  
PRAGMA recursive_triggers = [true|false];
```

## schema\_version Pragma

The **schema\_version** pragma gets or sets the schema version value that is stored in the database header. Following is the simple syntax:

```
PRAGMA [database.]schema_version;  
PRAGMA [database.]schema_version = number;
```

This is a 32-bit signed integer value that keeps track of schema changes. Whenever a schema-altering command is executed *like, CREATE... or DROP...*, this value is incremented.

## secure\_delete Pragma

The **secure\_delete** pragma is used to control how content is deleted from the database. Following is the simple syntax:

```
PRAGMA secure_delete;  
PRAGMA secure_delete = [true|false];  
PRAGMA database.secure_delete;  
PRAGMA database.secure_delete = [true|false];
```

The default value for the secure delete flag is normally off, but this can be changed with the SQLITE\_SECURE\_DELETE build option.

## sql\_trace Pragma

The **sql\_trace** pragma is used to dump SQL trace results to the screen. Following is the simple syntax:

```
PRAGMA sql_trace;  
PRAGMA sql_trace = [true|false];
```

SQLite must be compiled with the SQLITE\_DEBUG directive for this pragma to be included.

## synchronous Pragma

The **synchronous** pragma gets or sets the current disk synchronization mode which controls how aggressively SQLite will write data all the way out to physical storage. Following is the simple syntax:

```
PRAGMA [database.]synchronous;  
PRAGMA [database.]synchronous = mode;
```

SQLite supports the following synchronisation modes:

Pragma Value	Description
0 or OFF	No syncs at all
1 or NORMAL	Sync after each sequence of critical disk operations
2 or FULL	Sync after each critical disk operation

## temp\_store Pragma

The **temp\_store** pragma gets or sets the storage mode used by temporary database files. Following is the simple syntax:

```
PRAGMA temp_store;
PRAGMA temp_store = mode;
```

SQLite supports the following storage modes:

Pragma Value	Description
0 or DEFAULT	Use compile-time default. Normally FILE.
1 or FILE	Use file-based storage.
2 or MEMORY	Use memory-based storage.

## temp\_store\_directory Pragma

The **temp\_store\_directory** pragma gets or sets the location used for temporary database files. Following is the simple syntax:

```
PRAGMA temp_store_directory;
PRAGMA temp_store_directory = 'directory_path';
```

## user\_version Pragma

The **user\_version** pragma gets or sets the user-defined version value that is stored in the database header. Following is the simple syntax:

```
PRAGMA [database.]user_version;
PRAGMA [database.]user_version = number;
```

This is a 32-bit signed integer value which can be set by the developer for version tracking purpose.

## writable\_schema Pragma

The **writable\_schema** pragma gets or sets the ability to modify system tables. Following is the simple syntax:

```
PRAGMA writable_schema;
PRAGMA writable_schema = [true|false];
```

If this pragma is set, tables that start with `sqlite_` can be created and modified, including the `sqlite_master` table. Be careful while using pragma because it can lead to complete database corruption.

Loading [MathJax]/jax/output/HTML-CSS/jax.js