



*Small. Fast. Reliable.  
Choose any three.*

[Home](#) [Menu](#) [About](#) [Documentation](#) [Download](#) [License](#) [Support](#)

[Purchase](#)

[Search](#)

# Result and Error Codes

## ► Table Of Contents

### Overview

Many of the routines in the SQLite [C-language Interface](#) return numeric result codes indicating either success or failure, and in the event of a failure, providing some idea of the cause of the failure. This document strives to explain what each of those numeric result codes means.

## 1. Result Codes versus Error Codes

"Error codes" are a subset of "result codes" that indicate that something has gone wrong. There are only a few non-error result codes: [SQLITE\\_OK](#), [SQLITE\\_ROW](#), and [SQLITE\\_DONE](#). The term "error code" means any result code other than these three.

## 2. Primary Result Codes versus Extended Result Codes

Result codes are signed 32-bit integers. The least significant 8 bits of the result code define a broad category and are called the "primary result code". More significant bits provide more detailed information about the error and are called the "extended result code"

Note that the primary result code is always a part of the extended result code. Given a full 32-bit extended result code, the application can always find the corresponding primary result code merely by extracting the least significant 8 bits of the extended result code.

All extended result codes are also error codes. Hence the terms "extended result code" and "extended error code" are interchangeable.

For historic compatibility, the C-language interfaces return primary result codes by default. The extended result code for the most recent error can be retrieved using the [sqlite3\\_extended\\_errcode\(\)](#) interface. The [sqlite3\\_extended\\_result\\_codes\(\)](#) interface can be used to put a [database connection](#) into a mode where it returns the extended result codes instead of the primary result codes.

## 3. Definitions

All result codes are integers. Symbolic names for all result codes are created using "#define" macros in the `sqlite3.h` header file. There are separate sections in the `sqlite3.h` header file for the [result code definitions](#) and the [extended result code definitions](#).

Primary result code symbolic names are of the form "SQLITE\_XXXXXX" where XXXXXX is a sequence of uppercase alphabetic characters. Extended result code names are of the form "SQLITE\_XXXXXX\_YYYYYYY" where the XXXXXX part is the corresponding primary result code and the YYYYYYY is an extension that further classifies the result code.

The names and numeric values for existing result codes are fixed and unchanging. However, new result codes, and especially new extended result codes, might appear in future releases of SQLite.

## 4. Primary Result Code List

The 31 result codes are [defined in `sqlite3.h`](#) and are listed in alphabetical order below:

<a href="#">SQLITE_ABORT (4).</a>	<a href="#">SQLITE_MISUSE (21).</a>
<a href="#">SQLITE_AUTH (23).</a>	<a href="#">SQLITE_NOLFS (22).</a>
<a href="#">SQLITE_BUSY (5).</a>	<a href="#">SQLITE_NOMEM (7).</a>
<a href="#">SQLITE_CANTOPEN (14).</a>	<a href="#">SQLITE_NOTADB (26).</a>
<a href="#">SQLITE_CONSTRAINT (19).</a>	<a href="#">SQLITE_NOTFOUND (12).</a>
<a href="#">SQLITE_CORRUPT (11).</a>	<a href="#">SQLITE_NOTICE (27).</a>
<a href="#">SQLITE_DONE (101).</a>	<a href="#">SQLITE_OK (0).</a>
<a href="#">SQLITE_EMPTY (16).</a>	<a href="#">SQLITE_PERM (3).</a>
<a href="#">SQLITE_ERROR (1).</a>	<a href="#">SQLITE_PROTOCOL (15).</a>
<a href="#">SQLITE_FORMAT (24).</a>	<a href="#">SQLITE_RANGE (25).</a>
<a href="#">SQLITE_FULL (13).</a>	<a href="#">SQLITE_READONLY (8).</a>
<a href="#">SQLITE_INTERNAL (2).</a>	<a href="#">SQLITE_ROW (100).</a>
<a href="#">SQLITE_INTERRUPT (9).</a>	<a href="#">SQLITE_SCHEMA (17).</a>
<a href="#">SQLITE_IOERR (10).</a>	<a href="#">SQLITE_TOOBIG (18).</a>
<a href="#">SQLITE_LOCKED (6).</a>	<a href="#">SQLITE_WARNING (28).</a>
<a href="#">SQLITE_MISMATCH (20).</a>	

## 5. Extended Result Code List

The 71 extended result codes are [defined in `sqlite3.h`](#) and are listed in alphabetical order below:

[SQLITE\\_ABORT\\_ROLLBACK \(516\).](#)  
[SQLITE\\_BUSY\\_RECOVERY \(261\).](#)  
[SQLITE\\_BUSY\\_SNAPSHOT \(517\).](#)  
[SQLITE\\_BUSY\\_TIMEOUT \(773\).](#)  
[SQLITE\\_CANTOPEN\\_CONVPATH \(1038\).](#)  
[SQLITE\\_CANTOPEN\\_DIRTYWAL \(1294\).](#)  
[SQLITE\\_CANTOPEN\\_FULLPATH \(782\).](#)  
[SQLITE\\_CANTOPEN\\_ISDIR \(526\).](#)

[SQLITE\\_CANTOPEN\\_NOTEMPDIR \(270\)](#).  
[SQLITE\\_CANTOPEN\\_SYMLINK \(1550\)](#).  
[SQLITE\\_CONSTRAINT\\_CHECK \(275\)](#).  
[SQLITE\\_CONSTRAINT\\_COMMITHOOK \(531\)](#).  
[SQLITE\\_CONSTRAINT\\_FOREIGNKEY \(787\)](#).  
[SQLITE\\_CONSTRAINT\\_FUNCTION \(1043\)](#).  
[SQLITE\\_CONSTRAINT\\_NOTNULL \(1299\)](#).  
[SQLITE\\_CONSTRAINT\\_PINNED \(2835\)](#).  
[SQLITE\\_CONSTRAINT\\_PRIMARYKEY \(1555\)](#).  
[SQLITE\\_CONSTRAINT\\_ROWID \(2579\)](#).  
[SQLITE\\_CONSTRAINT\\_TRIGGER \(1811\)](#).  
[SQLITE\\_CONSTRAINT\\_UNIQUE \(2067\)](#).  
[SQLITE\\_CONSTRAINT\\_VTAB \(2323\)](#).  
[SQLITE\\_CORRUPT\\_INDEX \(779\)](#).  
[SQLITE\\_CORRUPT\\_SEQUENCE \(523\)](#).  
[SQLITE\\_CORRUPT\\_VTAB \(267\)](#).  
[SQLITE\\_ERROR\\_MISSING\\_COLLSEQ \(257\)](#).  
[SQLITE\\_ERROR\\_RETRY \(513\)](#).  
[SQLITE\\_ERROR\\_SNAPSHOT \(769\)](#).  
[SQLITE\\_IOERR\\_ACCESS \(3338\)](#).  
[SQLITE\\_IOERR\\_AUTH \(7178\)](#).  
[SQLITE\\_IOERR\\_BEGIN\\_ATOMIC \(7434\)](#).  
[SQLITE\\_IOERR\\_BLOCKED \(2826\)](#).  
[SQLITE\\_IOERR\\_CHECKRESERVEDLOCK \(3594\)](#).  
[SQLITE\\_IOERR\\_CLOSE \(4106\)](#).  
[SQLITE\\_IOERR\\_COMMIT\\_ATOMIC \(7690\)](#).  
[SQLITE\\_IOERR\\_CONVPATH \(6666\)](#).  
[SQLITE\\_IOERR\\_DATA \(8202\)](#).  
[SQLITE\\_IOERR\\_DELETE \(2570\)](#).  
[SQLITE\\_IOERR\\_DELETE\\_NOENT \(5898\)](#).  
[SQLITE\\_IOERR\\_DIR\\_CLOSE \(4362\)](#).  
[SQLITE\\_IOERR\\_DIR\\_FSYNC \(1290\)](#).  
[SQLITE\\_IOERR\\_FSTAT \(1802\)](#).  
[SQLITE\\_IOERR\\_FSYNC \(1034\)](#).  
[SQLITE\\_IOERR\\_GETTEMPPATH \(6410\)](#).  
[SQLITE\\_IOERR\\_LOCK \(3850\)](#).  
[SQLITE\\_IOERR\\_MMAP \(6154\)](#).  
[SQLITE\\_IOERR\\_NOMEM \(3082\)](#).  
[SQLITE\\_IOERR\\_RDLOCK \(2314\)](#).  
[SQLITE\\_IOERR\\_READ \(266\)](#).  
[SQLITE\\_IOERR\\_ROLLBACK\\_ATOMIC \(7946\)](#).  
[SQLITE\\_IOERR\\_SEEK \(5642\)](#).  
[SQLITE\\_IOERR\\_SHMLOCK \(5130\)](#).  
[SQLITE\\_IOERR\\_SHMMAP \(5386\)](#).  
[SQLITE\\_IOERR\\_SHMOPEN \(4618\)](#).  
[SQLITE\\_IOERR\\_SHMSIZE \(4874\)](#).  
[SQLITE\\_IOERR\\_SHORT\\_READ \(522\)](#).  
[SQLITE\\_IOERR\\_TRUNCATE \(1546\)](#).  
[SQLITE\\_IOERR\\_UNLOCK \(2058\)](#).  
[SQLITE\\_IOERR\\_VNODE \(6922\)](#).  
[SQLITE\\_IOERR\\_WRITE \(778\)](#).  
[SQLITE\\_LOCKED\\_SHARED\\_CACHE \(262\)](#).  
[SQLITE\\_LOCKED\\_VTAB \(518\)](#).  
[SQLITE\\_NOTICE\\_RECOVER\\_ROLLBACK \(539\)](#).  
[SQLITE\\_NOTICE\\_RECOVER\\_WAL \(283\)](#).

[SQLITE\\_OK\\_LOAD\\_PERMANENTLY \(256\).](#)  
[SQLITE\\_READONLY\\_CANTINIT \(1288\).](#)  
[SQLITE\\_READONLY\\_CANTLOCK \(520\).](#)  
[SQLITE\\_READONLY\\_DBMOVED \(1032\).](#)  
[SQLITE\\_READONLY\\_DIRECTORY \(1544\).](#)  
[SQLITE\\_READONLY\\_RECOVERY \(264\).](#)  
[SQLITE\\_READONLY\\_ROLLBACK \(776\).](#)  
[SQLITE\\_WARNING\\_AUTOINDEX \(284\).](#)

## 6. Result Code Meanings

The meanings for all 102 result code values are shown below, in numeric order.

### (0) SQLITE\_OK

The SQLITE\_OK result code means that the operation was successful and that there were no errors. Most other result codes indicate an error.

### (1) SQLITE\_ERROR

The SQLITE\_ERROR result code is a generic error code that is used when no other more specific error code is available.

### (2) SQLITE\_INTERNAL

The SQLITE\_INTERNAL result code indicates an internal malfunction. In a working version of SQLite, an application should never see this result code. If application does encounter this result code, it shows that there is a bug in the database engine.

SQLite does not currently generate this result code. However, [application-defined SQL functions](#) or [virtual tables](#), or [VFSes](#), or other extensions might cause this result code to be returned.

### (3) SQLITE\_PERM

The SQLITE\_PERM result code indicates that the requested access mode for a newly created database could not be provided.

### (4) SQLITE\_ABORT

The SQLITE\_ABORT result code indicates that an operation was aborted prior to completion, usually by application request. See also: [SQLITE\\_INTERRUPT](#).

If the callback function to [sqlite3\\_exec\(\)](#) returns non-zero, then [sqlite3\\_exec\(\)](#) will return SQLITE\_ABORT.

If a [ROLLBACK](#) operation occurs on the same [database connection](#) as a pending read or write, then the pending read or write may fail with an SQLITE\_ABORT or [SQLITE\\_ABORT\\_ROLLBACK](#) error.

In addition to being a result code, the SQLITE\_ABORT value is also used as a [conflict resolution mode](#) returned from the [sqlite3\\_vtab\\_on\\_conflict\(\)](#) interface.

## (5) SQLITE\_BUSY

The SQLITE\_BUSY result code indicates that the database file could not be written (or in some cases read) because of concurrent activity by some other [database connection](#), usually a database connection in a separate process.

For example, if process A is in the middle of a large write transaction and at the same time process B attempts to start a new write transaction, process B will get back an SQLITE\_BUSY result because SQLite only supports one writer at a time. Process B will need to wait for process A to finish its transaction before starting a new transaction. The [sqlite3\\_busy\\_timeout\(\)](#) and [sqlite3\\_busy\\_handler\(\)](#) interfaces and the [busy\\_timeout pragma](#) are available to process B to help it deal with SQLITE\_BUSY errors.

An SQLITE\_BUSY error can occur at any point in a transaction: when the transaction is first started, during any write or update operations, or when the transaction commits. To avoid encountering SQLITE\_BUSY errors in the middle of a transaction, the application can use [BEGIN IMMEDIATE](#) instead of just [BEGIN](#) to start a transaction. The [BEGIN IMMEDIATE](#) command might itself return SQLITE\_BUSY, but if it succeeds, then SQLite guarantees that no subsequent operations on the same database through the next [COMMIT](#) will return SQLITE\_BUSY.

See also: [SQLITE\\_BUSY\\_RECOVERY](#) and [SQLITE\\_BUSY\\_SNAPSHOT](#).

The SQLITE\_BUSY result code differs from [SQLITE\\_LOCKED](#) in that SQLITE\_BUSY indicates a conflict with a separate [database connection](#), probably in a separate process, whereas [SQLITE\\_LOCKED](#) indicates a conflict within the same [database connection](#) (or sometimes a database connection with a [shared cache](#)).

## (6) SQLITE\_LOCKED

The SQLITE\_LOCKED result code indicates that a write operation could not continue because of a conflict within the same [database connection](#) or a conflict with a different database connection that uses a [shared cache](#).

For example, a [DROP TABLE](#) statement cannot be run while another thread is reading from that table on the same [database connection](#) because dropping the table would delete the table out from under the concurrent reader.

The SQLITE\_LOCKED result code differs from [SQLITE\\_BUSY](#) in that SQLITE\_LOCKED indicates a conflict on the same [database connection](#) (or on a connection with a [shared cache](#)) whereas [SQLITE\\_BUSY](#) indicates a conflict with a different database connection, probably in a different process.

## (7) SQLITE\_NOMEM

The SQLITE\_NOMEM result code indicates that SQLite was unable to allocate all the memory it needed to complete the operation. In other words, an internal call to [sqlite3\\_malloc\(\)](#) or [sqlite3\\_realloc\(\)](#) has failed in a case where the memory being allocated was required in order to continue the operation.

## (8) SQLITE\_READONLY

The SQLITE\_READONLY result code is returned when an attempt is made to alter some data for which the current database connection does not have write permission.

## (9) SQLITE\_INTERRUPT

The SQLITE\_INTERRUPT result code indicates that an operation was interrupted by the [sqlite3\\_interrupt\(\)](#) interface. See also: [SQLITE\\_ABORT](#)

## (10) SQLITE\_IOERR

The SQLITE\_IOERR result code says that the operation could not finish because the operating system reported an I/O error.

A full disk drive will normally give an [SQLITE\\_FULL](#) error rather than an SQLITE\_IOERR error.

There are many different extended result codes for I/O errors that identify the specific I/O operation that failed.

## (11) SQLITE\_CORRUPT

The SQLITE\_CORRUPT result code indicates that the database file has been corrupted. See the [How To Corrupt Your Database Files](#) for further discussion on how corruption can occur.

## (12) SQLITE\_NOTFOUND

The SQLITE\_NOTFOUND result code is used in two contexts. SQLITE\_NOTFOUND can be returned by the [sqlite3\\_file\\_control\(\)](#) interface to indicate that the [file control opcode](#) passed as the third argument was not recognized by the underlying [VFS](#). SQLITE\_NOTFOUND can also be returned by the xSetSystemCall() method of an [sqlite3\\_vfs](#) object.

The SQLITE\_NOTFOUND result code is also used internally by the SQLite implementation, but those internal uses are not exposed to the application.

## (13) SQLITE\_FULL

The SQLITE\_FULL result code indicates that a write could not complete because the disk is full. Note that this error can occur when trying to write information into the main database file, or it can also occur when writing into [temporary disk files](#).

Sometimes applications encounter this error even though there is an abundance of primary disk space because the error occurs when writing into [temporary disk files](#) on a system where temporary files are stored on a separate partition with much less space than the primary disk.

## (14) SQLITE\_CANTOPEN

The SQLITE\_CANTOPEN result code indicates that SQLite was unable to open a file. The file in question might be a primary database file or one of several [temporary disk files](#).

## (15) SQLITE\_PROTOCOL

The SQLITE\_PROTOCOL result code indicates a problem with the file locking protocol used by SQLite. The SQLITE\_PROTOCOL error is currently only returned when using



[WAL mode](#) and attempting to start a new transaction. There is a race condition that can occur when two separate [database connections](#) both try to start a transaction at the same time in [WAL mode](#). The loser of the race backs off and tries again, after a brief delay. If the same connection loses the locking race dozens of times over a span of multiple seconds, it will eventually give up and return `SQLITE_PROTOCOL`. The `SQLITE_PROTOCOL` error should appear in practice very, very rarely, and only when there are many separate processes all competing intensely to write to the same database.

## (16) `SQLITE_EMPTY`

The `SQLITE_EMPTY` result code is not currently used.

## (17) `SQLITE_SCHEMA`

The `SQLITE_SCHEMA` result code indicates that the database schema has changed. This result code can be returned from [sqlite3\\_step\(\)](#) for a [prepared statement](#) that was generated using [sqlite3\\_prepare\(\)](#) or [sqlite3\\_prepare16\(\)](#). If the database schema was changed by some other process in between the time that the statement was prepared and the time the statement was run, this error can result.

If a [prepared statement](#) is generated from [sqlite3\\_prepare\\_v2\(\)](#) then the statement is automatically re-prepared if the schema changes, up to [SQLITE\\_MAX\\_SCHEMA\\_RETRY](#) times (default: 50). The [sqlite3\\_step\(\)](#) interface will only return `SQLITE_SCHEMA` back to the application if the failure persists after these many retries.

## (18) `SQLITE_TOOBIG`

The `SQLITE_TOOBIG` error code indicates that a string or BLOB was too large. The default maximum length of a string or BLOB in SQLite is 1,000,000,000 bytes. This maximum length can be changed at compile-time using the [SQLITE\\_MAX\\_LENGTH](#) compile-time option, or at run-time using the [sqlite3\\_limit\(db,SQLITE\\_LIMIT\\_LENGTH,...\)](#) interface. The `SQLITE_TOOBIG` error results when SQLite encounters a string or BLOB that exceeds the compile-time or run-time limit.

The `SQLITE_TOOBIG` error code can also result when an oversized SQL statement is passed into one of the [sqlite3\\_prepare\\_v2\(\)](#) interfaces. The maximum length of an SQL statement defaults to a much smaller value of 1,000,000 bytes. The maximum SQL statement length can be set at compile-time using [SQLITE\\_MAX\\_SQL\\_LENGTH](#) or at run-time using [sqlite3\\_limit\(db,SQLITE\\_LIMIT\\_SQL\\_LENGTH,...\)](#).

## (19) `SQLITE_CONSTRAINT`

The `SQLITE_CONSTRAINT` error code means that an SQL constraint violation occurred while trying to process an SQL statement. Additional information about the failed constraint can be found by consulting the accompanying error message (returned via [sqlite3\\_errmsg\(\)](#) or [sqlite3\\_errmsg16\(\)](#)) or by looking at the [extended error code](#).

The `SQLITE_CONSTRAINT` code can also be used as the return value from the [xBestIndex\(\)](#) method of a [virtual table](#) implementation. When [xBestIndex\(\)](#) returns `SQLITE_CONSTRAINT`, that indicates that the particular combination of inputs submitted to [xBestIndex\(\)](#) cannot result in a usable query plan and should not be given further consideration.

## (20) SQLITE\_MISMATCH

The SQLITE\_MISMATCH error code indicates a datatype mismatch.

SQLite is normally very forgiving about mismatches between the type of a value and the declared type of the container in which that value is to be stored. For example, SQLite allows the application to store a large BLOB in a column with a declared type of BOOLEAN. But in a few cases, SQLite is strict about types. The SQLITE\_MISMATCH error is returned in those few cases when the types do not match.

The [rowid](#) of a table must be an integer. Attempt to set the [rowid](#) to anything other than an integer (or a NULL which will be automatically converted into the next available integer rowid) results in an SQLITE\_MISMATCH error.

## (21) SQLITE\_MISUSE

The SQLITE\_MISUSE return code might be returned if the application uses any SQLite interface in a way that is undefined or unsupported. For example, using a [prepared statement](#) after that prepared statement has been [finalized](#) might result in an SQLITE\_MISUSE error.

SQLite tries to detect misuse and report the misuse using this result code. However, there is no guarantee that the detection of misuse will be successful. Misuse detection is probabilistic. Applications should never depend on an SQLITE\_MISUSE return value.

If SQLite ever returns SQLITE\_MISUSE from any interface, that means that the application is incorrectly coded and needs to be fixed. Do not ship an application that sometimes returns SQLITE\_MISUSE from a standard SQLite interface because that application contains potentially serious bugs.

## (22) SQLITE\_NOLFS

The SQLITE\_NOLFS error can be returned on systems that do not support large files when the database grows to be larger than what the filesystem can handle. "NOLFS" stands for "NO Large File Support".

## (23) SQLITE\_AUTH

The SQLITE\_AUTH error is returned when the [authorizer callback](#) indicates that an SQL statement being prepared is not authorized.

## (24) SQLITE\_FORMAT

The SQLITE\_FORMAT error code is not currently used by SQLite.

## (25) SQLITE\_RANGE

The SQLITE\_RANGE error indicates that the parameter number argument to one of the [sqlite3\\_bind](#) routines or the column number in one of the [sqlite3\\_column](#) routines is out of range.

## (26) SQLITE\_NOTADB



When attempting to open a file, the `SQLITE_NOTADB` error indicates that the file being opened does not appear to be an SQLite database file.

## (27) `SQLITE_NOTICE`

The `SQLITE_NOTICE` result code is not returned by any C/C++ interface. However, `SQLITE_NOTICE` (or rather one of its [extended error codes](#)) is sometimes used as the first argument in an [sqlite3\\_log\(\)](#) callback to indicate that an unusual operation is taking place.

## (28) `SQLITE_WARNING`

The `SQLITE_WARNING` result code is not returned by any C/C++ interface. However, `SQLITE_WARNING` (or rather one of its [extended error codes](#)) is sometimes used as the first argument in an [sqlite3\\_log\(\)](#) callback to indicate that an unusual and possibly ill-advised operation is taking place.

## (100) `SQLITE_ROW`

The `SQLITE_ROW` result code returned by [sqlite3\\_step\(\)](#) indicates that another row of output is available.

## (101) `SQLITE_DONE`

The `SQLITE_DONE` result code indicates that an operation has completed. The `SQLITE_DONE` result code is most commonly seen as a return value from [sqlite3\\_step\(\)](#) indicating that the SQL statement has run to completion. But `SQLITE_DONE` can also be returned by other multi-step interfaces such as [sqlite3\\_backup\\_step\(\)](#).

## (256) `SQLITE_OK_LOAD_PERMANENTLY`

The [sqlite3\\_load\\_extension\(\)](#) interface loads an [extension](#) into a single database connection. The default behavior is for that extension to be automatically unloaded when the database connection closes. However, if the extension entry point returns `SQLITE_OK_LOAD_PERMANENTLY` instead of `SQLITE_OK`, then the extension remains loaded into the process address space after the database connection closes. In other words, the `xDlClose` methods of the [sqlite3\\_vfs](#) object is not called for the extension when the database connection closes.

The `SQLITE_OK_LOAD_PERMANENTLY` return code is useful to [loadable extensions](#) that register new [VFSes](#), for example.

## (257) `SQLITE_ERROR_MISSING_COLLSEQ`

The `SQLITE_ERROR_MISSING_COLLSEQ` result code means that an SQL statement could not be prepared because a collating sequence named in that SQL statement could not be located.

Sometimes when this error code is encountered, the [sqlite3\\_prepare\\_v2\(\)](#) routine will convert the error into [SQLITE\\_ERROR\\_RETRY](#) and try again to prepare the SQL statement using a different query plan that does not require the use of the unknown collating sequence.

## (261) SQLITE\_BUSY\_RECOVERY

The SQLITE\_BUSY\_RECOVERY error code is an [extended error code](#) for [SQLITE\\_BUSY](#) that indicates that an operation could not continue because another process is busy recovering a [WAL mode](#) database file following a crash. The SQLITE\_BUSY\_RECOVERY error code only occurs on [WAL mode](#) databases.

## (262) SQLITE\_LOCKED\_SHAREDCACHE

The SQLITE\_LOCKED\_SHAREDCACHE result code indicates that access to an SQLite data record is blocked by another database connection that is using the same record in [shared cache mode](#). When two or more database connections share the same cache and one of the connections is in the middle of modifying a record in that cache, then other connections are blocked from accessing that data while the modifications are on-going in order to prevent the readers from seeing a corrupt or partially completed change.

## (264) SQLITE\_READONLY\_RECOVERY

The SQLITE\_READONLY\_RECOVERY error code is an [extended error code](#) for [SQLITE\\_READONLY](#). The SQLITE\_READONLY\_RECOVERY error code indicates that a [WAL mode](#) database cannot be opened because the database file needs to be recovered and recovery requires write access but only read access is available.

## (266) SQLITE\_IOERR\_READ

The SQLITE\_IOERR\_READ error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error in the [VFS](#) layer while trying to read from a file on disk. This error might result from a hardware malfunction or because a filesystem came unmounted while the file was open.

## (267) SQLITE\_CORRUPT\_VTAB

The SQLITE\_CORRUPT\_VTAB error code is an [extended error code](#) for [SQLITE\\_CORRUPT](#) used by [virtual tables](#). A [virtual table](#) might return SQLITE\_CORRUPT\_VTAB to indicate that content in the virtual table is corrupt.

## (270) SQLITE\_CANTOPEN\_NOTEMPDIR

The SQLITE\_CANTOPEN\_NOTEMPDIR error code is no longer used.

## (275) SQLITE\_CONSTRAINT\_CHECK

The SQLITE\_CONSTRAINT\_CHECK error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that a [CHECK constraint](#) failed.

## (283) SQLITE\_NOTICE\_RECOVER\_WAL

The SQLITE\_NOTICE\_RECOVER\_WAL result code is passed to the callback of [sqlite3\\_log\(\)](#) when a [WAL mode](#) database file is recovered.

## (284) SQLITE\_WARNING\_AUTOINDEX

The `SQLITE_WARNING_AUTOINDEX` result code is passed to the callback of [sqlite3\\_log\(\)](#) whenever [automatic indexing](#) is used. This can serve as a warning to application designers that the database might benefit from additional indexes.

### (513) `SQLITE_ERROR_RETRY`

The `SQLITE_ERROR_RETRY` is used internally to provoke [sqlite3\\_prepare\\_v2\(\)](#) (or one of its sibling routines for creating prepared statements) to try again to prepare a statement that failed with an error on the previous attempt.

### (516) `SQLITE_ABORT_ROLLBACK`

The `SQLITE_ABORT_ROLLBACK` error code is an [extended error code](#) for [SQLITE\\_ABORT](#) indicating that an SQL statement aborted because the transaction that was active when the SQL statement first started was rolled back. Pending write operations always fail with this error when a rollback occurs. A [ROLLBACK](#) will cause a pending read operation to fail only if the schema was changed within the transaction being rolled back.

### (517) `SQLITE_BUSY_SNAPSHOT`

The `SQLITE_BUSY_SNAPSHOT` error code is an [extended error code](#) for [SQLITE\\_BUSY](#) that occurs on [WAL mode](#) databases when a database connection tries to promote a read transaction into a write transaction but finds that another [database connection](#) has already written to the database and thus invalidated prior reads.

The following scenario illustrates how an `SQLITE_BUSY_SNAPSHOT` error might arise:

1. Process A starts a read transaction on the database and does one or more `SELECT` statement. Process A keeps the transaction open.
2. Process B updates the database, changing values previous read by process A.
3. Process A now tries to write to the database. But process A's view of the database content is now obsolete because process B has modified the database file after process A read from it. Hence process A gets an `SQLITE_BUSY_SNAPSHOT` error.

### (518) `SQLITE_LOCKED_VTAB`

The `SQLITE_LOCKED_VTAB` result code is not used by the SQLite core, but it is available for use by extensions. Virtual table implementations can return this result code to indicate that they cannot complete the current operation because of locks held by other threads or processes.

The [R-Tree extension](#) returns this result code when an attempt is made to update the R-Tree while another prepared statement is actively reading the R-Tree. The update cannot proceed because any change to an R-Tree might involve reshuffling and rebalancing of nodes, which would disrupt read cursors, causing some rows to be repeated and other rows to be omitted.

### (520) `SQLITE_READONLY_CANTLOCK`

The `SQLITE_READONLY_CANTLOCK` error code is an [extended error code](#) for [SQLITE\\_READONLY](#). The `SQLITE_READONLY_CANTLOCK` error code indicates that

SQLite is unable to obtain a read lock on a [WAL mode](#) database because the shared-memory file associated with that database is read-only.

## (522) SQLITE\_IOERR\_SHORT\_READ

The SQLITE\_IOERR\_SHORT\_READ error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating that a read attempt in the [VFS](#) layer was unable to obtain as many bytes as was requested. This might be due to a truncated file.

## (523) SQLITE\_CORRUPT\_SEQUENCE

The SQLITE\_CORRUPT\_SEQUENCE result code means that the schema of the `sqlite_sequence` table is corrupt. The `sqlite_sequence` table is used to help implement the [AUTOINCREMENT](#) feature. The `sqlite_sequence` table should have the following format:

```
CREATE TABLE sqlite_sequence(name,seq);
```

If SQLite discovers that the `sqlite_sequence` table has any other format, it returns the SQLITE\_CORRUPT\_SEQUENCE error.

## (526) SQLITE\_CANTOPEN\_ISDIR

The SQLITE\_CANTOPEN\_ISDIR error code is an [extended error code](#) for [SQLITE\\_CANTOPEN](#) indicating that a file open operation failed because the file is really a directory.

## (531) SQLITE\_CONSTRAINT\_COMMITHOOK

The SQLITE\_CONSTRAINT\_COMMITHOOK error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that a [commit hook callback](#) returned non-zero that thus caused the SQL statement to be rolled back.

## (539) SQLITE\_NOTICE\_RECOVER\_ROLLBACK

The SQLITE\_NOTICE\_RECOVER\_ROLLBACK result code is passed to the callback of [sqlite3\\_log\(\)](#) when a [hot journal](#) is rolled back.

## (769) SQLITE\_ERROR\_SNAPSHOT

The SQLITE\_ERROR\_SNAPSHOT result code might be returned when attempting to start a read transaction on an historical version of the database by using the [sqlite3\\_snapshot\\_open\(\)](#) interface. If the historical snapshot is no longer available, then the read transaction will fail with the SQLITE\_ERROR\_SNAPSHOT. This error code is only possible if SQLite is compiled with [-DSQLITE\\_ENABLE\\_SNAPSHOT](#).

## (773) SQLITE\_BUSY\_TIMEOUT

The SQLITE\_BUSY\_TIMEOUT error code indicates that a blocking Posix advisory file lock request in the VFS layer failed due to a timeout. Blocking Posix advisory locks are only available as a proprietary SQLite extension and even then are only supported if SQLite is compiled with the [SQLITE\\_ENABLE\\_SETLK\\_TIMEOUT](#) compile-time option.

## (776) SQLITE\_READONLY\_ROLLBACK

The SQLITE\_READONLY\_ROLLBACK error code is an [extended error code](#) for [SQLITE\\_READONLY](#). The SQLITE\_READONLY\_ROLLBACK error code indicates that a database cannot be opened because it has a [hot journal](#) that needs to be rolled back but cannot because the database is readonly.

## (778) SQLITE\_IOERR\_WRITE

The SQLITE\_IOERR\_WRITE error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error in the [VFS](#) layer while trying to write into a file on disk. This error might result from a hardware malfunction or because a filesystem came unmounted while the file was open. This error should not occur if the filesystem is full as there is a separate error code (SQLITE\_FULL) for that purpose.

## (779) SQLITE\_CORRUPT\_INDEX

The SQLITE\_CORRUPT\_INDEX result code means that SQLite detected an entry is or was missing from an index. This is a special case of the [SQLITE\\_CORRUPT](#) error code that suggests that the problem might be resolved by running the [REINDEX](#) command, assuming no other problems exist elsewhere in the database file.

## (782) SQLITE\_CANTOPEN\_FULLPATH

The SQLITE\_CANTOPEN\_FULLPATH error code is an [extended error code](#) for [SQLITE\\_CANTOPEN](#) indicating that a file open operation failed because the operating system was unable to convert the filename into a full pathname.

## (787) SQLITE\_CONSTRAINT\_FOREIGNKEY

The SQLITE\_CONSTRAINT\_FOREIGNKEY error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that a [foreign key constraint](#) failed.

## (1032) SQLITE\_READONLY\_DBMOVED

The SQLITE\_READONLY\_DBMOVED error code is an [extended error code](#) for [SQLITE\\_READONLY](#). The SQLITE\_READONLY\_DBMOVED error code indicates that a database cannot be modified because the database file has been moved since it was opened, and so any attempt to modify the database might result in database corruption if the processes crashes because the [rollback journal](#) would not be correctly named.

## (1034) SQLITE\_IOERR\_FSYNC

The SQLITE\_IOERR\_FSYNC error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error in the [VFS](#) layer while trying to flush previously written content out of OS and/or disk-control buffers and into persistent storage. In other words, this code indicates a problem with the fsync() system call in unix or the FlushFileBuffers() system call in windows.

## (1038) SQLITE\_CANTOPEN\_CONVPATH

The SQLITE\_CANTOPEN\_CONVPATH error code is an [extended error code](#) for [SQLITE\\_CANTOPEN](#) used only by Cygwin [VFS](#) and indicating that the `cygwin_conv_path()` system call failed while trying to open a file. See also: [SQLITE\\_IOERR\\_CONVPATH](#)

### (1043) SQLITE\_CONSTRAINT\_FUNCTION

The SQLITE\_CONSTRAINT\_FUNCTION error code is not currently used by the SQLite core. However, this error code is available for use by extension functions.

### (1288) SQLITE\_READONLY\_CANTINIT

The SQLITE\_READONLY\_CANTINIT result code originates in the `xShmMap` method of a [VFS](#) to indicate that the shared memory region used by [WAL mode](#) exists but its content is unreliable and unusable by the current process since the current process does not have write permission on the shared memory region. (The shared memory region for WAL mode is normally a file with a "-wal" suffix that is mmap'ed into the process space. If the current process does not have write permission on that file, then it cannot write into shared memory.)

Higher level logic within SQLite will normally intercept the error code and create a temporary in-memory shared memory region so that the current process can at least read the content of the database. This result code should not reach the application interface layer.

### (1290) SQLITE\_IOERR\_DIR\_FSYNC

The SQLITE\_IOERR\_DIR\_FSYNC error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error in the [VFS](#) layer while trying to invoke `fsync()` on a directory. The unix [VFS](#) attempts to `fsync()` directories after creating or deleting certain files to ensure that those files will still appear in the filesystem following a power loss or system crash. This error code indicates a problem attempting to perform that `fsync()`.

### (1294) SQLITE\_CANTOPEN\_DIRTYWAL

The SQLITE\_CANTOPEN\_DIRTYWAL result code is not used at this time.

### (1299) SQLITE\_CONSTRAINT\_NOTNULL

The SQLITE\_CONSTRAINT\_NOTNULL error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that a [NOT NULL constraint](#) failed.

### (1544) SQLITE\_READONLY\_DIRECTORY

The SQLITE\_READONLY\_DIRECTORY result code indicates that the database is read-only because process does not have permission to create a journal file in the same directory as the database and the creation of a journal file is a prerequisite for writing.

### (1546) SQLITE\_IOERR\_TRUNCATE

The SQLITE\_IOERR\_TRUNCATE error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error in the [VFS](#) layer while trying to truncate a file



to a smaller size.

## (1550) SQLITE\_CANTOPEN\_SYMLINK

The SQLITE\_CANTOPEN\_SYMLINK result code is returned by the [sqlite3\\_open\(\)](#) interface and its siblings when the [SQLITE\\_OPEN\\_NOFOLLOW](#) flag is used and the database file is a symbolic link.

## (1555) SQLITE\_CONSTRAINT\_PRIMARYKEY

The SQLITE\_CONSTRAINT\_PRIMARYKEY error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that a [PRIMARY KEY constraint](#) failed.

## (1802) SQLITE\_IOERR\_FSTAT

The SQLITE\_IOERR\_FSTAT error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error in the [VFS](#) layer while trying to invoke `fstat()` (or the equivalent) on a file in order to determine information such as the file size or access permissions.

## (1811) SQLITE\_CONSTRAINT\_TRIGGER

The SQLITE\_CONSTRAINT\_TRIGGER error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that a [RAISE function](#) within a [trigger](#) fired, causing the SQL statement to abort.

## (2058) SQLITE\_IOERR\_UNLOCK

The SQLITE\_IOERR\_UNLOCK error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within `xUnlock` method on the [sqlite3\\_io\\_methods](#) object.

## (2067) SQLITE\_CONSTRAINT\_UNIQUE

The SQLITE\_CONSTRAINT\_UNIQUE error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that a [UNIQUE constraint](#) failed.

## (2314) SQLITE\_IOERR\_RDLOCK

The SQLITE\_IOERR\_UNLOCK error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within `xLock` method on the [sqlite3\\_io\\_methods](#) object while trying to obtain a read lock.

## (2323) SQLITE\_CONSTRAINT\_VTAB

The SQLITE\_CONSTRAINT\_VTAB error code is not currently used by the SQLite core. However, this error code is available for use by application-defined [virtual tables](#).

## (2570) SQLITE\_IOERR\_DELETE

The SQLITE\_IOERR\_UNLOCK error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within `xDelete` method on the [sqlite3\\_vfs](#) object.

## (2579) SQLITE\_CONSTRAINT\_ROWID

The SQLITE\_CONSTRAINT\_ROWID error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that a [rowid](#) is not unique.

## (2826) SQLITE\_IOERR\_BLOCKED

The SQLITE\_IOERR\_BLOCKED error code is no longer used.

## (2835) SQLITE\_CONSTRAINT\_PINNED

The SQLITE\_CONSTRAINT\_PINNED error code is an [extended error code](#) for [SQLITE\\_CONSTRAINT](#) indicating that an [UPDATE trigger](#) attempted to delete the row that was being updated in the middle of the update.

## (3082) SQLITE\_IOERR\_NOMEM

The SQLITE\_IOERR\_NOMEM error code is sometimes returned by the [VFS](#) layer to indicate that an operation could not be completed due to the inability to allocate sufficient memory. This error code is normally converted into [SQLITE\\_NOMEM](#) by the higher layers of SQLite before being returned to the application.

## (3338) SQLITE\_IOERR\_ACCESS

The SQLITE\_IOERR\_ACCESS error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within the xAccess method on the [sqlite3\\_vfs](#) object.

## (3594) SQLITE\_IOERR\_CHECKRESERVEDLOCK

The SQLITE\_IOERR\_CHECKRESERVEDLOCK error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within the xCheckReservedLock method on the [sqlite3\\_io\\_methods](#) object.

## (3850) SQLITE\_IOERR\_LOCK

The SQLITE\_IOERR\_LOCK error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error in the advisory file locking logic. Usually an SQLITE\_IOERR\_LOCK error indicates a problem obtaining a [PENDING lock](#). However it can also indicate miscellaneous locking errors on some of the specialized [VFSES](#) used on Macs.

## (4106) SQLITE\_IOERR\_CLOSE

The SQLITE\_IOERR\_CLOSE error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within the xClose method on the [sqlite3\\_io\\_methods](#) object.

## (4362) SQLITE\_IOERR\_DIR\_CLOSE

The SQLITE\_IOERR\_DIR\_CLOSE error code is no longer used.

## (4618) SQLITE\_IOERR\_SHMOPEN

The SQLITE\_IOERR\_SHMOPEN error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within the xShmMap method on the [sqlite3\\_io\\_methods](#) object while trying to open a new shared memory segment.

## (4874) SQLITE\_IOERR\_SHMSIZE

The SQLITE\_IOERR\_SHMSIZE error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within the xShmMap method on the [sqlite3\\_io\\_methods](#) object while trying to enlarge a ["shm" file](#) as part of [WAL mode](#) transaction processing. This error may indicate that the underlying filesystem volume is out of space.

## (5130) SQLITE\_IOERR\_SHMLOCK

The SQLITE\_IOERR\_SHMLOCK error code is no longer used.

## (5386) SQLITE\_IOERR\_SHMMAP

The SQLITE\_IOERR\_SHMMAP error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within the xShmMap method on the [sqlite3\\_io\\_methods](#) object while trying to map a shared memory segment into the process address space.

## (5642) SQLITE\_IOERR\_SEEK

The SQLITE\_IOERR\_SEEK error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within the xRead or xWrite methods on the [sqlite3\\_io\\_methods](#) object while trying to seek a file descriptor to the beginning point of the file where the read or write is to occur.

## (5898) SQLITE\_IOERR\_DELETE\_NOENT

The SQLITE\_IOERR\_DELETE\_NOENT error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating that the xDelete method on the [sqlite3\\_vfs](#) object failed because the file being deleted does not exist.

## (6154) SQLITE\_IOERR\_MMAP

The SQLITE\_IOERR\_MMAP error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating an I/O error within the xFetch or xUnfetch methods on the [sqlite3\\_io\\_methods](#) object while trying to map or unmap part of the database file into the process address space.

## (6410) SQLITE\_IOERR\_GETTEMPPATH

The SQLITE\_IOERR\_GETTEMPPATH error code is an [extended error code](#) for [SQLITE\\_IOERR](#) indicating that the [VFS](#) is unable to determine a suitable directory in which to place temporary files.

## (6666) SQLITE\_IOERR\_CONVPATH

The SQLITE\_IOERR\_CONVPATH error code is an [extended error code](#) for [SQLITE\\_IOERR](#) used only by Cygwin [VFS](#) and indicating that the `cygwin_conv_path()` system call failed. See also: [SQLITE\\_CANTOPEN\\_CONVPATH](#)

## (6922) SQLITE\_IOERR\_VNODE

The SQLITE\_IOERR\_VNODE error code is a code reserved for use by extensions. It is not used by the SQLite core.

## (7178) SQLITE\_IOERR\_AUTH

The SQLITE\_IOERR\_AUTH error code is a code reserved for use by extensions. It is not used by the SQLite core.

## (7434) SQLITE\_IOERR\_BEGIN\_ATOMIC

The SQLITE\_IOERR\_BEGIN\_ATOMIC error code indicates that the underlying operating system reported an error on the [SQLITE\\_FCNTL\\_BEGIN\\_ATOMIC\\_WRITE](#) file-control. This only comes up when [SQLITE\\_ENABLE\\_ATOMIC\\_WRITE](#) is enabled and the database is hosted on a filesystem that supports atomic writes.

## (7690) SQLITE\_IOERR\_COMMIT\_ATOMIC

The SQLITE\_IOERR\_COMMIT\_ATOMIC error code indicates that the underlying operating system reported an error on the [SQLITE\\_FCNTL\\_COMMIT\\_ATOMIC\\_WRITE](#) file-control. This only comes up when [SQLITE\\_ENABLE\\_ATOMIC\\_WRITE](#) is enabled and the database is hosted on a filesystem that supports atomic writes.

## (7946) SQLITE\_IOERR\_ROLLBACK\_ATOMIC

The SQLITE\_IOERR\_ROLLBACK\_ATOMIC error code indicates that the underlying operating system reported an error on the [SQLITE\\_FCNTL\\_ROLLBACK\\_ATOMIC\\_WRITE](#) file-control. This only comes up when [SQLITE\\_ENABLE\\_ATOMIC\\_WRITE](#) is enabled and the database is hosted on a filesystem that supports atomic writes.

## (8202) SQLITE\_IOERR\_DATA

The SQLITE\_IOERR\_DATA error code is an [extended error code](#) for [SQLITE\\_IOERR](#) used only by [checksum VFS shim](#) to indicate that the checksum on a page of the database file is incorrect.