Small. Fast. Reliable.
Choose any three.

Home Menu About Documentation Download License Support Purchase Search

SQLite Autoincrement

1. Summary

1. The AUTOINCREMENT keyword imposes extra CPU, memory, disk space, and disk I/O overhead and should be avoided if not strictly needed. It is usually not needed.
2. In SQLite, a column with type INTEGER PRIMARY KEY is an alias for the ROWID (except in WITHOUT ROWID tables) which is always a 64-bit signed integer.
3. On an INSERT, if the ROWID or INTEGER PRIMARY KEY column is not explicitly given a value, then it will be filled automatically with an unused integer, usually one more than the largest ROWID currently in use. This is true regardless of whether or not the AUTOINCREMENT keyword is used.
4. If the AUTOINCREMENT keyword appears after INTEGER PRIMARY KEY, that changes the automatic ROWID assignment algorithm to prevent the reuse of ROWIDs over the lifetime of the database. In other words, the purpose of AUTOINCREMENT is to prevent the reuse of ROWIDs from previously deleted rows.

2. Background

In SQLite, table rows normally have a 64-bit signed integer ROWID which is unique among all rows in the same table. (WITHOUT ROWID tables are the exception.)

You can access the ROWID of an SQLite table using one of the special column names ROWID, _ROWID_, or OID. Except if you declare an ordinary table column to use one of those special names, then the use of that name will refer to the declared column not to the internal ROWID.

If a table contains a column of type INTEGER PRIMARY KEY, then that column becomes an alias for the ROWID. You can then access the ROWID using any of four different names, the original three names described above or the name given to the INTEGER PRIMARY KEY column. All these names are aliases for one another and work equally well in any context.

When a new row is inserted into an SQLite table, the ROWID can either be specified as part of the INSERT statement or it can be assigned automatically by the database engine. To specify a ROWID manually, just include it in the list of values to be inserted. For example:

```
CREATE TABLE test1(a INT, b TEXT);
INSERT INTO test1(rowid, a, b) VALUES(123, 5, 'hello');
```

If no ROWID is specified on the insert, or if the specified ROWID has a value of NULL, then an appropriate ROWID is created automatically. The usual algorithm is to give the newly created row a ROWID that is one larger than the largest ROWID in the table prior to the insert. If the table is initially empty, then a ROWID of 1 is used. If the largest ROWID is equal to the largest possible integer (9223372036854775807) then the database engine starts picking positive candidate ROWIDs at random until it finds one that is not previously used. If no unused ROWID can be found after a reasonable number of attempts, the insert operation fails with an SQLITE_FULL error. If no negative ROWID values are inserted explicitly, then automatically generated ROWID values will always be greater than zero.

The normal ROWID selection algorithm described above will generate monotonically increasing unique ROWIDs as long as you never use the maximum ROWID value and you never delete the entry in the table with the largest ROWID. If you ever delete rows or if you ever create a row with the maximum possible ROWID, then ROWIDs from previously deleted rows might be reused when creating new rows and newly created ROWIDs might not be in strictly ascending order.

3. The AUTOINCREMENT Keyword

If a column has the type INTEGER PRIMARY KEY AUTOINCREMENT then a slightly different ROWID selection algorithm is used. The ROWID chosen for the new row is at least one larger than the largest ROWID that has ever before existed in that same table. If the table has never before contained any data, then a ROWID of 1 is used. If the largest possible ROWID has previously been inserted, then new INSERTs are not allowed and any attempt to insert a new row will fail with an SQLITE_FULL error. Only ROWID values from previous transactions that were committed are considered. ROWID values that were rolled back are ignored and can be reused.

SQLite keeps track of the largest ROWID using an internal table named "sqlite_sequence". The sqlite_sequence table is created and initialized automatically whenever a normal table that contains an AUTOINCREMENT column is created. The content of the sqlite_sequence table can be modified using ordinary UPDATE, INSERT, and DELETE statements. But making modifications to this table will likely perturb the AUTOINCREMENT key generation algorithm. Make sure you know what you are doing before you undertake such changes. The sqlite_sequence table does not track ROWID changes associated with UPDATE statement, only INSERT statements.

The behavior implemented by the AUTOINCREMENT keyword is subtly different from the default behavior. With AUTOINCREMENT, rows with automatically selected ROWIDs are guaranteed to have ROWIDs that have never been used before by the same table in the same database. And the automatically generated ROWIDs are guaranteed to be monotonically increasing. These are important properties in certain applications. But if your application does not need these properties, you should probably stay with the default behavior since the use of AUTOINCREMENT requires additional work to be done as each row is inserted and thus causes INSERTs to run a little slower.

Note that "monotonically increasing" does not imply that the ROWID always increases by exactly one. One is the usual increment. However, if an insert fails due to (for example) a uniqueness constraint, the ROWID of the failed insertion attempt might not be reused on subsequent inserts, resulting in gaps in the ROWID sequence. AUTOINCREMENT guarantees that automatically chosen ROWIDs will be increasing but not that they will be sequential.

Because AUTOINCREMENT keyword changes the behavior of the ROWID selection algorithm, AUTOINCREMENT is not allowed on WITHOUT ROWID tables or on any table column other than INTEGER PRIMARY KEY. Any attempt to use AUTOINCREMENT on a WITHOUT ROWID table or on a column other than the INTEGER PRIMARY KEY column results in an error.