




SQLite



**The (un)known
Super Ant**



Advantages

- **Zero conf, serverless**
- **Free** for all use (public domain)
- **Compact** and **fast**
- The most **implemented** in the world, **standard**
- **Reliable**, very well tested





SQLite

SQLite uses

When use SQLite?

- Data analysis
- As proprietary file format
- Cache, temporary, in memory
- Single user version, education or experimental
- Embedded, mobile, IoT

When **NOT** use SQLite?

- Many concurrent writes
- GIANT databases (more than 281 Tb)
- Accessed by network



SQLite

Best practices with Python

SQLite y Python: **Connecting**

```
con = sqlite3.connect(database[,...])
con.row_factory = sqlite3.Row
row = con.execute("SELECT 1 as a")
      .fetchone()
col1 = row["a"]
```



SQLite y Python: **Use context managers**

```
con = sqlite3.connect(database[,...])  
with con:  
    con.execute("sql")
```



SQLite y Python: **PLACE HOLDERS**

- **Place holders,**

```
sql = "INSERT INTO tb (a, b) VALUES (?, ?)"
```

```
con.execute(sql, (var_A, var_B))
```

- Avoid conversion errors.
- Prevent sql injection.



SQLite y Python: **Optimize**

- Prefer using connection to cursors

```
con.execute("sql", (variables,))
```

```
con.executemany("sql", [n-tuples])
```

```
con.executescript("sql; sql; ...")
```



SQLite y Python: **PRAGMAS**

```
con.execute("PRAGMA  
           schema.option = value")
```

- To see and configure
- Schema refers to default database.





SQLite

Transactions
Isolation
Concurrency

Transactions

- All or none.
 - Writes all the data in the transaction or nothing.

BEGIN

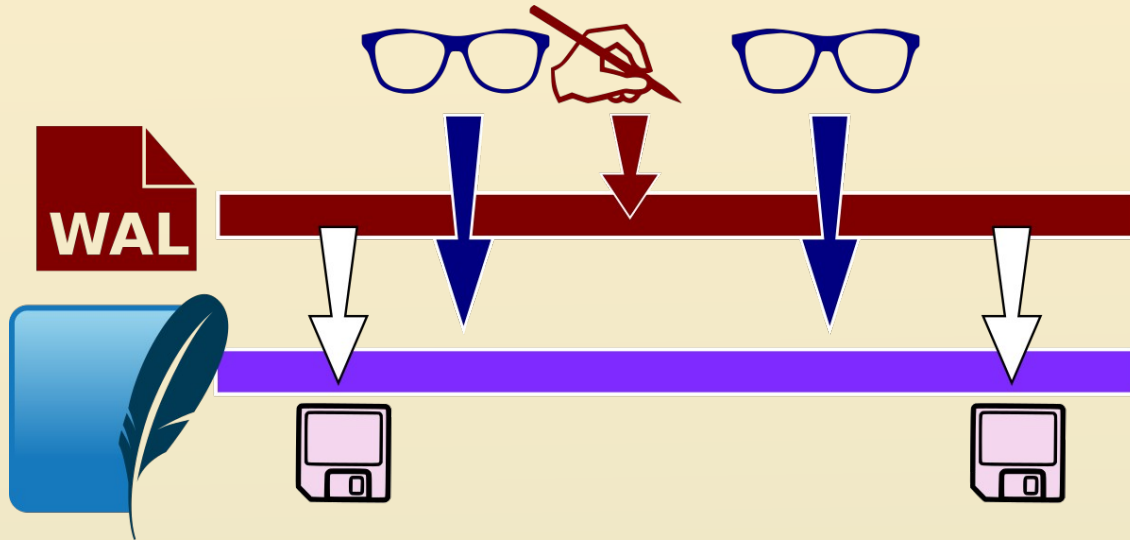
Modifications

COMMIT or ROLLBACK



Journal Mode: **WRITE AHEAD LOGGING**

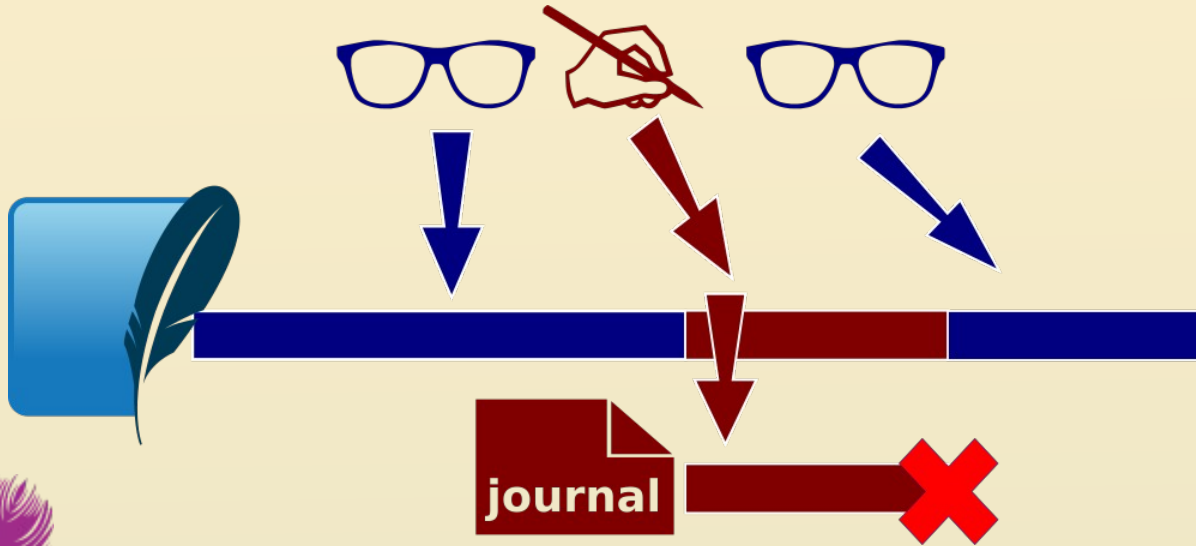
- Writes a logging file. Sync afterwards.



```
PRAGMA  
journal_mode=  
WAL;
```

Journal Mode: **Rollback**

- Rollback files.



```
PRAGMA  
journal_mode=  
DELETE;  
TRUNCATE;  
PERSIST;
```

Journal Mode: **In memory or disabled**

- Maximize performance
- Higher risk of corruption

```
PRAGMA  
journal_mode=  
MEMORY;  
OFF;
```



Isolation levels

- Determines when changes become visible to other connections

Default:

```
con = sqlite3.connect(database,  
check_same_thread=True, isolation_level=None)
```

Isolation levels: **EXCLUSIVE**

- Use only one connection.
- Only one reader and one writer.



Isolation levels: **IMMEDIATE**

- ↳ Request writing
- ↳ Complete pending operations
- ↳ Lock DB and write.
- ↳ Release the DB.

Isolation levels: **DEFERRED**

- ↳ Request permission to write
- ↳ Use read-only mode. Allow readings
- ↳ Finish writing
- ↳ Free the DB for writing



SQLite

Indexing

Indexing: When?

- Fields used for searches.
- Fields that will be used for relationships
- ✗ AVOID over indexing
 - Decreases performance

Partial index

- Index only some rows in a table

```
CREATE UNIQUE INDEX leaders  
ON members(id_team)  
WHERE is_team_leader;
```

Expression index.

- Index by the result of **deterministic functions**.

```
CREATE INDEX t2xy ON t2(x+y);  
SELECT * FROM t2 WHERE y+x=22;
```


FTS FULL TEXT SEARCH

- Split words from text data.
- Generate VIRTUAL TABLES
- Use MATCH operator.

```
CREATE VIRTUAL TABLE email USING fts5(sender, title,  
body);
```

```
SELECT * FROM email WHERE email MATCH 'search';
```



SQLite

Data types

Date & Time

Data types: standard

- Str
- Int
- Float
- None
- Bytes
- **BLOB**: Large binary objects
- JSON



Date and time: SQLite functions.

Convert str to date

- Date, Time,
- Datetime
- julianday
- strftime

Modifiers

Adding or changing dates

- Start of período
- */- x período



Date and time: SQLite functions

```
>>> cur = con.execute("select date('now')")
```

```
>>> cur.fetchall()
```

```
[('2021-07-23',)]
```



Date and time: SQLite functions

The 3rd sunday on October's.

```
date('now', 'start of year',  
    '+9 months', '+14 days', 'weekday 0')
```



Date and time

```
con = sqlite3.connect('db',  
detect_types=sqlite3.PARSE_DECLTYPES)
```

```
con.execute('CREATE TABLE foo (bar, baz timestamp)')
```

```
con.execute('INSERT INTO foo VALUES (?, ?)',  
            (23, datetime.datetime.now()))
```



Date and time

```
>>> cursor = con.execute("select * from foo")
>>> cursor.fetchall()

[(23, datetime.datetime(2021, 7, 22, 22, 24, 17,
561061))]
```






Content

- SQLite and Python: best practices
- Transactions, Isolation and Concurrency
- Indexing
- Data types, date & time



SQLite



**The (un)known
Super Ant**

