The Journal of Open Source Software

StampDB: A tiny C++ Time Series Database library designed for compatibility with the PyData Ecosystem.

Aadya A. Chinubhai[1]

1 Santa Clara University, Santa Clara, California, United States

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 18 September 2025
Published: unpublished

Abstract

StampDB is a lightweight time series database designed for seamless compatibility with the PyData ecosystem. The system provides a Python-native interface that handles time series data without the architectural complexity of enterprise-grade database systems. StampDB achieves native compatibility with NumPy (Harris et al., 2020) and Python's (*Python Programming Language*, n.d.) datetime module (*Python Datetime Module Documentation*, n.d.), offering researchers and developers a streamlined solution for time series data management in single-node environments.

1. Statement of Need

Time series data processing represents a fundamental requirement across numerous application domains, spanning Internet of Things (IoT) deployments to scientific research initiatives. However, existing database solutions predominantly target enterprise-level architectures, introducing unnecessary complexity for smaller-scale applications. Furthermore, many established systems lack native NumPy (Harris et al., 2020) compatibility, thereby limiting integration with the broader Python (*Python Programming Language*, n.d.) scientific computing ecosystem. This limitation proves particularly significant given that contemporary data analysis workflows predominantly utilize Python-based libraries including Scikit-learn (Pedregosa et al., 2011), NumPy (Harris et al., 2020), and Pandas(team, 2020). The proposed StampDB addresses these limitations through a C++ backend implementation that provides low-level control while maintaining a Python-centric interface design.

2. Introduction

2.1 Project Goals

StampDB was designed to: - Provide a straightforward API for time series data management - Maintain a minimal codebase focused on core functionality - Be natively compatible with NumPy (Harris et al., 2020) and pythons (*Python Programming Language*, n.d.) datetime (*Python Datetime Module Documentation*, n.d.) module.

2.2 Target Use Cases

- Sensor Data
- Scientific and Research Data Acquisition
- Single Node Data Processing
- Private Data Storage

3. System Architecture

3.1 Core Components

At its heart, StampDB is a CSV (Shafranovich, 2005) file with a schema. The schema is stored in a separate file and is used to validate the data as it is written to the CSV (Shafranovich, 2005) file.

StampDB comprises of two things: - A C++ library that provides the core functionality. - A Python wrapper that provides a simple NumPy (Harris et al., 2020) native API for the C++ library.

C++ Core: - CSV Parsing and Reads/Writes using csv2 (Kumar, n.d.). - In-Memory time based indexing. - Atomic Writes via shadow copies. - Append only disk writes. - A converter to convert CSV C++ objects to a NumPy (Harris et al., 2020) structured array.

Python Wrapper: - A simple NumPy (Harris et al., 2020) native API for the C++ library. - A simple query language for NumPy structured arrays.

4. You should not use StampDB if

Access from multiple processes or threads - An HTTP server - Management of relationships between tables - Access control and users - ACID guarantees - High performance as the size of your dataset grows

5. API Reference

5.1 StampDB class

I/O using the StampDB class.

```
class StampDB(path: str, schema: dict) — Opens or creates a CSV-backed database.
Example: db = StampDB("test.csv", schema={"temp": "float", "humidity": "float"})
    - append_point(point: Point) -> None — Append a new record to the database.
    - checkpoint() -> None — Flush in-memory append-only writes to disk.
    - delete_point(time: int | float) -> None — Mark a record deleted (in-memory).
    - compact() -> None — Physically remove deleted records from disk (auto on close).
    - read_range(start: int, end: int) -> np.ndarray — Read data between two time values
    - close() -> None — Cleanly close the database.
```

The Point class.

```
class Point - Represents a single time-stamped record.
Example: p = Point(time=1, data=[22.5, "moderate"])
    time: int | float
    values: dict[str, Any]
```

The Relational Algebra basic classes.

```
class Selection(condition: str, data: np.ndarray) — Filters rows based on a condition st
class Projection(columns: list[str], data: np.ndarray) — Select specific columns; do() -
class Summation(column: str, data: np.ndarray) — Compute sum of a column; do() -> float.
class OrderBy(columns: list[str], data: np.ndarray) — Sort rows by one or more columns;
```

The Relational Algebra join classes.

```
class InnerJoin(left: np.ndarray, right: np.ndarray, left_key: str, right_key: str) — In
class OuterJoin(left: np.ndarray, right: np.ndarray, left_key: str, right_key: str) — Fu
class LeftOuterJoin(left: np.ndarray, right: np.ndarray, left_key: str, right_key: str)
```

6. Runtime Comparison.

Though high performance is not the primary goal of `StampDB`, it performs significantly better than native Python libraries like tinyflux (citrusvanilla, 2025).

Runtime Comparison with tinyflux (citrusvanilla, 2025)

Operation	Speedup
Writes	2×
Queries	50×
Reads	30×

Steps to Reproduce

1. Install `tinyflux` and `StampDB`.
2. Navigate to the directory containing `benchmarks.py`.
3. Run the benchmark:

```
python benchmarks.py
```

7. Conclusion

StampDB provides a minimal, easy-to-use solution for basic time series data storage in Python. Its simplicity makes it suitable for educational purposes and small-scale applications where a full-fledged database would be unnecessary. The implementation focuses on core functionality while maintaining a clean and maintainable codebase.

License

We distribute under the permissive Apache License 2.0.

References

citrusvanilla. (2025). *TinyFlux: The tiny time series database*. https://github.com/citrusvanilla/tinyflux

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). *NumPy: Array computing with python*. https://numpy.org/.

Kumar, P. S. (n.d.). *csv2 c++ library*. https://github.com/p-ranav/csv2.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Python datetime module documentation. (n.d.). https://docs.python.org/3/library/datetime.html.

Python programming language. (n.d.). https://www.python.org/.

Shafranovich, Y. (2005). *Common format and MIME type for comma-separated values (CSV) files*. RFC 4180; IETF. https://doi.org/10.17487/RFC4180

team, T. pandas development. (2020). *Pandas-dev/pandas: pandas* (latest). Zenodo. https://doi.org/10.5281/zenodo.3509134