**Roadmap for sql-batcher Improvements**

**1. Async Support**

* Add an **AsyncSQLBatcher** sibling class.
* Define async-compatible adapters (e.g., AsyncPostgreSQLAdapter) using asyncpg, aiomysql, or other async DB drivers.
* Support both sync and async execution models for wider adoption in fast ETL and microservice pipelines.

**Why?**  
Modern Python applications (especially FastAPI, aiohttp-based, or asyncio-driven ETL pipelines) demand async I/O for maximum throughput.

**2. Custom Exception Hierarchy**

* Define specific exception classes:
  + SQLBatcherError
  + BatchSizeExceededError
  + InvalidQueryError
  + AdapterConnectionError
* Raise precise exceptions instead of generic ones.

**Why?**  
Makes it easier for users to programmatically handle different failure modes during execution.

**3. Retry and Timeout Handling**

* Implement an optional **retry policy** (e.g., using tenacity library or a lightweight custom retry wrapper).
* Allow user-defined:
  + Maximum retries
  + Backoff strategies
  + Timeout limits for batch execution

**Why?**  
Robustness: batches that temporarily fail (e.g., network hiccups, transient DB overload) could auto-retry safely without manual intervention.

**4. Plugin/Hook System**

* Allow users to inject custom **pre-batch** and **post-batch** hooks:
  + Preprocessing SQL
  + Logging metrics
  + Auditing queries
* Hook architecture could be simple (on\_pre\_batch, on\_post\_batch) or pluggable with entrypoints (like Flask or Airflow plugins).

**Why?**  
Increases extensibility without needing to fork the library for custom workflows.

**5. Context Manager Support**

* Implement \_\_enter\_\_ and \_\_exit\_\_ on adapters and batchers:

python

CopyEdit

with SQLBatcher(adapter=TrinoAdapter(...)) as batcher:

batcher.add("INSERT INTO foo VALUES (1, 'bar')")

* Ensure connections are cleanly closed automatically if opened internally.

**Why?**  
Resource safety and code cleanliness.

**6. Enhanced Metrics and Monitoring**

* Track and expose:
  + Number of statements batched
  + Total bytes sent
  + Average batch size
  + Execution time per batch
* Optional integration with logging or metrics libraries like prometheus\_client.

**Why?**  
Enables users to monitor throughput and debug slowdowns without intrusive instrumentation.

**7. Connection Pooling Helpers (Optional)**

* Offer built-in light helpers for connection pooling setup (e.g., using SQLAlchemy pools).
* Could be offered as optional utilities, not hard dependency.

**Why?**  
Improves scaling performance, especially for high-frequency batch workloads.

**8. Expanded Testing and Benchmarking**

* Add benchmarks:
  + Insert batching vs naive inserts
  + Different databases (Trino, Postgres, Snowflake)
* Increase test coverage to 90%+.
* Add property-based tests with hypothesis for fuzzing unusual batch inputs.

**Why?**  
Gives confidence that optimizations work across DBs and data volumes.

**9. Codebase Refinements**

* Minor refinements:
  + Move shared utilities to a utils/ or common/ module.
  + Clarify public vs internal API surface (e.g., dunder methods, underscore-prefixed helpers).
* Modularize configuration handling (batch size logic, byte counting).

**Why?**  
Future maintainability and easier onboarding for contributors.

**Example Vision Statement for v2.0**

*"SQLBatcher v2.0 is the high-performance, fully extensible SQL batching framework for Python, offering first-class support for sync and async execution, fine-grained error handling, flexible retry policies, dynamic plugin integrations, and out-of-the-box metrics tracking."*

**Quick Summary Table**

| **Feature** | **Priority** | **Notes** |
| --- | --- | --- |
| Async Support | High | Unlocks modern Python use cases |
| Retry/Timeout Mechanism | High | Improves reliability |
| Custom Exception Classes | Medium | Better debugging |
| Plugin/Hook System | Medium | Extensibility |
| Context Manager Support | Medium | Resource management |
| Metrics & Monitoring | Medium | Observability |
| Connection Pooling Helpers | Low | Optional improvement |
| Expanded Testing & Benchmark | Medium | Trust and adoption |