

SQLAlchemy ORM transactions

SQLodin's Python 0.3 client supports real Session and Core transactions. Statements are provisional until commit; flush does not publish writes. Exceptions, rollback, connection close and pool reset discard uncommitted work.

```
from sqlalchemy.orm import Session
from sqlodin.sqlalchemy import create_engine

engine = create_engine(nodes, cluster="orders", tls=tls)
with Session(engine) as session, session.begin():
    order = Order(customer="Ada")
    session.add(order)
    session.flush()          # generated integer primary key is now available
    session.add(Line(order_id=order.id, product="Book"))
# Both objects commit together through Paxos, or neither does.
```

The default isolation level is `SERIALIZABLE`. `Session.begin_nested()` supports savepoint rollback/release, including recovering from a constraint violation inside a nested block. Generated integer primary keys, relationships, autoflush, expire-on-commit reload and SQLAlchemy's matched-row counts are supported. Declare mapped tables explicitly; general schema reflection, explicit DML RETURNING and two-phase/XA transactions remain outside this implementation.

Execution and commit

The service first obtains a fresh quorum read barrier and returns an application revision. A transaction retains that revision and its bounded write statements in the Python connection. Each flush/read sends the staged body to the contacted voter. A short-lived SQLite connection evaluates the body privately, returns changes, generated row IDs or query results, then rolls back before the service processes another request. No database copy, permanent workspace, SQLite transaction or writer lock survives a network round trip. Speculative work has a VM instruction budget.

Previews never constitute a durable acknowledgement. The complete body and read revision are submitted once through Paxos at commit. Every voter checks the revision in log order before executing SQL. The outcome, session identity, new revision and application watermark commit atomically. The existing durable function policy is also enforced at commit; defaults or expressions outside that policy may therefore be rejected at commit even if a provisional evaluation succeeded.

Every newly applied application request, including DDL, advances the revision. Read barriers, rejected requests and deduplicated retries do not. A transaction that observes a different revision aborts with `SQLSTATE 40001`. This is currently a database-wide conflict check: unrelated concurrent writes can also force a retry. It prevents lost updates, write skew and phantom-dependent writes conservatively.

The serializability argument is conditional on SQLodin's existing deterministic SQL contract: every preview uses the same application revision and replays the same preceding staged statements; a successful ordered commit proves no other application write intervened. Read-only transactions can serialize at their initial barrier. This is not a proof of arbitrary SQL determinism, nor a claim that all workloads benefit from optimistic execution.

Retrying a serialization failure

Retry the whole transaction in a new Session after rollback, including its reads and application decisions. Do not retry only its last UPDATE. The adapter deliberately does not rerun user code automatically.

```

from sqlalchemy.exc import OperationalError
from sqlodcin.dbapi import SerializationError

for attempt in range(3):
    try:
        with Session(engine) as session, session.begin():
            order = session.get(Order, order_id)
            order.status = "paid"
        break
    except OperationalError as exc:
        if not isinstance(exc.orig, SerializationError) or attempt == 2:
            raise

```

An uncertain commit is different: inspect `OperationalError.orig.pending`, save its serialized identity, then resolve the exact request through the native client using the same authenticated SAN. Its digest includes the read revision. A successful commit remains successful when retried after other writes; it is not re-executed. Rollback/reset is refused while that commit is unresolved. Discard the affected SQLAlchemy connection after saving the pending identity. A fresh transaction is not a substitute for resolving an unknown commit.

Bounds, cost and compatibility

A transaction retains the existing bounds of eight write statements, 4096 SQL bytes, sixteen bound write parameters and 384 total vector components. Queries have their own bounded parameters/results. Releasing a savepoint retains its writes; rolling back a savepoint removes its staged suffix. Large flushes fail before being committed.

A flush/read replays preceding staged writes, so repeated flushes cost more CPU and SQL execution than a native one-shot buffered batch. Database-wide validation can cause high abort rates under sustained concurrent writers. Use short transactions; finer read/write dependency tracking and incremental private workspaces require separate performance measurements and correctness work. The database still uses the same pinned static dependencies and whole paxos-odin library.

Explicit `autocommit=True` remains available for independent per-statement commits. In that mode SQLAlchemy rollback does not undo earlier successful statements, as in SQLAlchemy's documented AUTOCOMMIT contract. The default transactional mode follows SQLAlchemy's Session lifecycle.

The current compatibility boundary is **store format 5 / SQL policy 9 / peer wire 3**. Incompatible stores and peers are rejected. The reviewed format-4 policies 6/7/8 have an explicit offline migration; there is no automatic or rolling upgrade. Preserve source stores and follow the migration and recovery procedure. The transaction revision is explicitly encoded in the request identity. The qualified scope and evidence are in the release record.

Verification

The reproducible service suites are `tools/check_orm_transactions.py` and `tools/check_network_hosts.py --orm`. They exercise disk-backed voters, including read-own-writes, generated keys, nested rollback, predicate write-skew prevention, voter loss, uncertain commit recovery, minority refusal and all-voter restart. Reports retain binary/source hashes under `benchmarks/results/`. SOD 0004 records the design decisions; the release record binds the qualified reports.