DOI: 10.15514/ISPRAS-2026-38(4)-25

Сравнительный анализ производительности библиотек объектно-реляционного отображения для Python

А.Е. Чернобровенко, ORCID: 0009-0003-0745-884X <sashachernobrovenko@gmail.com>
Д.А. Лылов, ORCID: 0009-0007-3015-953X <denislylov2004@mail.ru>
М.В. Панченко, ORCID: 0000-0001-9032-2203 <panchenko.maks@gmail.com>

Белгородский государственный технологический университет им. В.Г. Шухова,
Россия, 308012, Белгород, ул. Костюкова, д. 46.

Аннотация. Объектно-реляционное отображение является распространённым способом работы с реляционными базами данных в приложениях на языке программирования Python: данные описываются через модели, а запросы выполняются на уровне объектов. Цена такой абстракции – накладные расходы, способные заметно влиять на время выполнения типовых операций чтения и записи, особенно в условиях высокого числа транзакций к базе данных. В работе предложена воспроизводимая методика тестирования производительности библиотек объектно-реляционного отображения и представлены результаты сопоставления библиотек Django, Peewee, Pony, SQLAlchemy и SQLModel в 14 сценариях создания, чтения, обновления и удаления данных. Для каждого сценария фиксируются среднее значение, медиана и 99-й перцентиль времени выполнения операции. Результаты сведены в таблицы и сопровождаются краткими комментариями, ориентированными на практический выбор инструмента под характерные сценарии.

Ключевые слова: язык программирования Python; объектно-реляционное отображение; производительность; SQL.

Для цитирования: Чернобровенко А.Е., Лылов Д.А., Панченко М.В. Сравнительный анализ производительности библиотек объектно-реляционного отображения для Python. Труды ИСП РАН, 2026, том 38, вып. 4, часть 2, стр. 177–192. DOI: 10.15514/ISPRAS–2026–38(4)–25.

Comparative performance analysis of object-relational mapping libraries for Python

A.E. Chernobrovenko, ORCID: 0009-0003-0745-884X <sashachernobrovenko@gmail.com>
D.A. Lylov, ORCID: 0009-0007-3015-953X <denislylov2004@mail.ru>
M.V. Panchenko, ORCID: 0000-0001-9032-2203 <panchenko.maks@gmail.com>

Belgorod State Technological University named after V.G. Shukhov,
46, Kostyukova st., Belgorod, 308012, Russia.

Abstract. Object–relational mapping is a widely used approach for working with relational databases in Python applications: data are represented as models, and queries are executed at the object level. The cost of this abstraction is additional overhead, which can significantly affect the execution time of typical read and write operations, especially when handling a large number of database transactions. This paper proposes a reproducible methodology for performance benchmarking of object–relational mapping libraries and presents results for Django, Peewee, Pony, SQLAlchemy, and SQLModel across 14 scenarios covering create, read, update, and delete operations. For each scenario, the mean, median, and 99th percentile of operation latency are reported. The results are summarized in tables and accompanied by brief comments to support practical tool selection for representative usage scenarios.

Keywords: Python; object-relational mapping; performance; SQL.

For citation: Chernobrovenko A.E., Lylov D.A., Panchenko M.V. Comparative performance analysis of object-relational mapping libraries for Python. Trudy ISP RAN/Proc. ISP RAS, 2026, vol. 38, issue 4, part 2, pp. 177-192 (in Russian). DOI: 10.15514/ISPRAS-2026-38(4)-25.

1. Введение

Выбор объектно-реляционных отображений (ORM) в проектах на Python нередко определяется интеграцией с прикладной инфраструктурой, наличием миграций, средствами типизации и уровнем поддержки инструмента. Однако при росте нагрузки производительность слоя доступа к данным становится критической: увеличивается число запросов, появляются массовые операции и усложняются выборки.

Сравнение ORM по результатам тестирования требует аккуратной интерпретации. Измеряемое время включает не только выполнение SQL на стороне системы управления базами данных (СУБД), но и накладные расходы ORM на работу с объектной моделью: ведение контекста/сессии, материализацию результата в объекты, синхронизацию изменений, обработку связей и другие операции, которые зависят от внутренней архитектуры библиотеки.

В настоящей работе предлагается воспроизводимая методика оценки производительности ORM-библиотек для Python. Эксперимент выполнен на основе стандартизированного набора из 14 тестовых сценариев операций создания, чтения, обновления и удаления данных (CRUD) в синхронной реализации, позволяющих выявлять особенности реализации транзакций, пакетной обработки и жизненного цикла объектов в разных ORM. Для эксперимента выбраны пять популярных свободно распространяемых ORM-библиотек: Django [1], Peewee [2], Pony [3], SQLAlchemy [4] и SQLModel [5].

Существующие подходы к измерению производительности средств доступа к данным, работающих с реляционными СУБД, можно условно разделить на две группы. К первой относятся стандартизированные бенчмарки семейства TPC, в том числе TPC-C и TPC-E для транзакционных нагрузок, а также TPC-H и TPC-DS для аналитических запросов. Их достоинством является формализованный протокол испытаний, воспроизводимые схемы данных и унифицированные метрики. Вместе с тем такие бенчмарки ориентированы прежде всего на сопоставление СУБД и аппаратно-программных конфигураций и потому не позволяют анализировать непосредственно накладные расходы ORM-слоя, связанные с

генерацией SQL, ведением контекста, материализацией результатов в объекты и синхронизацией изменений [6].

Ко второй группе относятся универсальные платформы и прикладные бенчмарки, такие как YCSB, OLTP-Bench и TechEmpower Framework Benchmarks. В них используются либо типовые нагрузки CRUD и сканирования, либо сценарии web-приложений с обращением к базе данных через ORM. Эти подходы важны с точки зрения контроля нагрузки и анализа процентилей задержек, однако они либо ориентированы на сравнение широкого класса систем хранения данных, либо измеряют производительность приложения end-to-end и потому не выделяют влияние ORM-библиотеки как отдельного объекта исследования [7-9].

В работах, посвящённых непосредственно ORM, обычно рассматриваются отдельные экосистемы и конкретные наборы сценариев: стратегии объектно-реляционного отображения, типовые ошибки доступа к данным, а также сравнительный анализ ORM в .NET- и TypeScript-стеке [10-13]. Проведённый обзор показывает, что для современных Python ORM отсутствует общепринятый воспроизводимый набор тестов, позволяющий в едином окружении сопоставить актуальные версии Django, Peewee, Pony, SQLAlchemy и SQLModel по типовым операциям CRUD.

Новизна работы состоит в формировании воспроизводимого набора микросценариев, ориентированных именно на накладные расходы ORM-слоя в актуальном стеке разработки на основе Python. Актуальность исследования определяется тем, что версии ORM-библиотек, драйверов и поддерживаемых ими режимов работы регулярно обновляются, из-за чего результаты ранее выполненных исследований быстро теряют прикладную ценность.

2. Особенности сравниваемых ORM-библиотек

Во всех рассматриваемых ORM взаимодействие с PostgreSQL осуществляется через драйвер, реализующий спецификацию Python DB-API [14]. В идеальном случае для повышения сопоставимости измерений следует фиксировать единую версию драйвера для всех библиотек, однако целью эксперимента является сравнение ORM в конфигурациях, релевантных их практическому использованию, то есть на актуальных и официально поддерживаемых версиях драйверов. В связи с отсутствием поддержки драйвера psycopg3 [15] в Pony используется psycopg2 [16] (как наиболее современная поддерживаемая связка для Pony). Это различие следует учитывать при интерпретации результатов как фактор, потенциально влияющий на абсолютные значения времени выполнения.

Отдельно следует отметить, что SQLModel представляет собой надстройку над SQLAlchemy: библиотека использует SQLAlchemy как «ядро» для построения и выполнения SQL-запросов (декларативное описание сущностей, механизм сессий и конструирование запросов), дополняя его удобными средствами типизации и интеграцией с Pydantic [17]. Это означает, что часть поведения SQLModel наследуется от SQLAlchemy, тогда как различия проявляются в накладных расходах на уровень абстракции, типизацию и преобразование данных [18].

Для Pony важно учитывать особенность внутренней реализации: библиотека кэширует результат трансляции генераторных выражений и лямбда-выражений в SQL, поэтому компиляция каждого структурно идентичного запроса выполняется только один раз. При повторном использовании запроса применяется результат из глобального кэша процесса. За счёт этого снижаются накладные расходы на построение SQL на стороне приложения при большом числе однотипных операций.

Существенные различия между ORM также обусловлены применяемой моделью работы с единицей данных и управлением транзакциями. Pony, SQLAlchemy и SQLModel реализуют подход «Unit of Work» [19]: изменения отслеживаются в таблице соответствия объектов (identity map) – структуре, сопоставляющей первичные ключи объектам и обеспечивающей уникальность экземпляров в рамках сессии. Фактическая синхронизация с БД выполняется на этапе flush/commit, что позволяет агрегировать операции и влияет на структуру

генерируемого SQL. В отличие от этого, Django и Peewee, близкие к паттерну «Active Record» [20], как правило, инициируют выполнение SQL-операций непосредственно при вызове соответствующих методов. Объединение нескольких запросов в одну транзакцию требует явного задания границ транзакции со стороны разработчика. Эти архитектурные особенности напрямую влияют на количество запросов [21], частоту и границы транзакций [22], а также на накладные расходы по созданию и сопровождению ORM-объектов.

3. Методика проведения сравнительного тестирования

3.1 Особенности проведения измерений

Для тестов используется предварительно подготовленная база данных [23], схема которой представлена на рис. 1. Отдельно стоит отметить, что таблица Ticket содержит внешний ключ по полю book_ref на Bookings без каскадирования при удалении, а также составной уникальный ключ (book_ref, passenger_id, outbound).

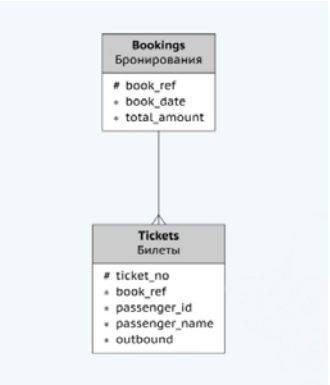

Рис. 1. Схема исходной базы данных.
Fig. 1. Original database schema.

В результаты экспериментальных измерений скорости работы ORM заметную долю времени могут вносить не собственно операции CRUD, а сопутствующие «шумы» среды выполнения [24]: задержки операций ввода-вывода (I/O) подсистемы, создание соединений, первичная инициализация контекста, первичная инициализация драйвера и заполнение пулов. Чтобы измерения отражали именно работу ORM поверх уже готовой инфраструктуры доступа к БД, используются заранее подготовленные пулы с установленными соединениями.

Перед основной серией измерений выполняется прогрев: серия пробных CRUD-запросов, позволяющая заполнить пул рабочими соединениями и стабилизировать состояние БД и драйвера. Непосредственно перед замером выполняется подготовка контекста/сессии, чтобы исключить из измеряемого интервала накладные расходы на получение соединения из пула и первичную инициализацию. Для последующей интерпретации результатов также анализируются SQL-запросы, сгенерированные ORM.

Для каждой ORM выполняется 100 полных циклов измерений. Полный цикл включает последовательное выполнение всех 14 тестовых сценариев в фиксированном порядке.

Число повторений сценария в пределах одного цикла зависит от его типа. Одноэлементные операции создания, обновления и удаления выполняются 2500 раз за цикл. Результатом сценария для данного цикла является медиана полученных значений времени выполнения. Сценарии выборки выполняются 75 раз за цикл. Результатом сценария для данного цикла также является медиана полученных значений времени выполнения. Массовые транзакции и

пакетные операции выполняются однократно за цикл, поскольку одна такая операция соответствует обработке 2500 объектов. Результатом такого сценария является время его выполнения.

Итоговые статистические характеристики рассчитываются после завершения всех циклов на основе значений, полученных для каждого сценария в отдельных циклах.

Таким образом, результаты отражают стоимость генерации и выполнения запросов, а также накладные расходы ORM-слоя (включая жизненный цикл ORM-объектов) в условиях, близких к типичной серверной конфигурации: долгоживущие процессы и заранее прогретые соединения.

3.2 Набор тестов

Перечень тестовых сценариев включает типовые операции CRUD (одиночные, пакетные и транзакционные), а также выборки различной сложности – от точечных запросов до фильтрации с пагинацией и запросов со связями:

1. Одиночное создание объекта;
2. Создание 2500 объектов в транзакции;
3. Пакетное создание 2500 объектов;
4. Выборка всех записей;
5. Выборка первой записи;
6. Выборка по первичному ключу;
7. Выборка с ограничением и включением атрибутов связанной записи;
8. Выборка с фильтром, пагинацией со смещением и сортировкой;
9. Одиночное обновление объекта;
10. Обновление 2500 объектов в транзакции;
11. Пакетное обновление 2500 объектов;
12. Одиночное удаление объекта;
13. Удаление 2500 объектов в транзакции;
14. Пакетное удаление 2500 объектов.

В большинстве тестов измеряется не только время генерации и выполнения SQL-запроса, но и накладные расходы ORM, связанные с работой с объектами моделей: материализация результата в объекты, ведение контекста/сессии и жизненный цикл объектов при создании, обновлении и удалении. Отдельно выделен сценарий, в котором оценивается стоимость запроса со связями между сущностями без материализации ORM-объектов, поскольку запрос формирует проекцию набора полей, и результат возвращается в виде кортежей значений, а не экземпляров моделей.

3.3 Метрики

Для количественной оценки производительности рассматриваются три статистики времени выполнения операции [25]:

- avg – среднее время выполнения;
- p50 – медиана: значение, относительно которого половина измерений быстрее, половина медленнее;
- p99 – 99-й перцентиль: в 99% измерений время выполнения операции не превышает это значение.

3.4 Конфигурация

Измерения выполняются на виртуальном сервере (VPS) с 4 vCPU (3,3 ГГц) и 8 ГБ оперативной памяти. Эксперименты проводятся в среде Python 3.12. В конфигурации используются следующие версии ORM-библиотек: Django 5.2.8, Peewee 3.19.0, Pony 0.7.19, SQLAlchemy 2.0.44 и SQLModel 0.0.31. В качестве СУБД выбрана PostgreSQL 17 – распространённая система управления базами данных с открытым исходным кодом, поддерживаемая всеми рассматриваемыми ORM-библиотеками и широко используемая в прикладной разработке.

4. Результаты

Результаты измерений по каждому из 14 тестовых сценариев представлены в виде таблиц. Для каждого сценария приведены агрегированные показатели времени выполнения операций: среднее значение (avg), медиана (p50) и 99-й перцентиль (p99) [25]. При интерпретации различий учитывались накладные расходы ORM-слоя и структура SQL-запросов, формируемая соответствующими библиотеками. Комментарии содержат пояснения особенностей выполнения сценариев и ключевых закономерностей, влияющих на практический выбор инструмента.

4.1 Тест 1. Одиночное создание объекта

В сценарии измеряется суммарное время операции создания и сохранения одного объекта: формирование данных на стороне приложения, обработку их ORM и драйвером, а также выполнение одной операции INSERT на стороне СУБД. Результаты приведены в табл. 1.

Табл. 1. Одиночное создание объекта – время выполнения, мс.
Table. 1. Single object creation – execution time, ms.

ORM	avg	p50	p99
Peewee	0,850	0,829	0,996
Pony	0,925	0,762	1,529
Django	0,890	0,835	1,180
SQLAlchemy	1,898	1,818	2,263
SQLModel	1,960	1,990	2,165

Наименьшее суммарное время выполнения операции демонстрируют Peewee, Django и Pony. SQLAlchemy и SQLModel в данном сценарии оказываются медленнее. Причина различий – более высокая стоимость накладных расходов на операцию в этих библиотеках (ведение сессии/контекста и служебных структур), которая особенно заметна в простых точечных сценариях [4, 5]. Хотя Pony использует схожую с SQLAlchemy модель работы, она оказывается быстрее за счёт кэширования трансляции запроса в SQL: после первой компиляции повторные вызовы не включают эту стоимость [3].

4.2 Тест 2. Создание 2500 объектов в транзакции

В тесте подсчитывается суммарное время выполнения операции создания 2500 объектов и их сохранения в базе данных в рамках одной крупной транзакции, что позволяет оценить эффективность выполнения операции создания при использовании транзакций.

Результаты приведены в табл. 2.

SQLAlchemy и SQLModel демонстрируют лучшие результаты, поскольку в рамках одной крупной транзакции выполняют группировку вставок: вместо последовательности отдельных INSERT формируется пакетная вставка, что подтверждается анализом трассировки SQL-запросов, полученной из встроенных средств SQL-логирования ORM [4-5].

Следом идёт Pony: при сопоставимом с Peewee и Django SQL-запросе ускорение объясняется тем же эффектом кэширования, который отмечен в тесте 1.

Django и Peewee показывают более высокие значения, что соответствует выполнению цикла отдельных INSERT внутри транзакционной обёртки [1-2].

Табл. 2. Создание 2500 объектов в транзакции – время выполнения, с.
Table 2. Creation of 2500 objects in a transaction – execution time, s.

ORM	avg	p50	p99
Peewee	1,006	0,995	1,201
Pony	0,446	0,428	0,655
Django	0,971	0,908	1,258
SQLAlchemy	0,167	0,165	0,189
SQLModel	0,239	0,229	0,326

4.3 Тест 3. Пакетное создание 2500 объектов

Данный сценарий показывает суммарное время создания 2500 объектов и их сохранения в БД одной операцией (одним INSERT-запросом с множеством значений). Результаты приведены в табл. 3.

Табл. 3. Пакетное создание 2500 объектов – время выполнения, с.
Table 3. Bulk creation of 2500 objects – execution time, s.

ORM	avg	p50	p99
Peewee	0,108	0,104	0,137
Pony	–	–	–
Django	0,112	0,108	0,161
SQLAlchemy	0,100	0,097	0,141
SQLModel	0,151	0,142	0,202

Лучшее время демонстрирует SQLAlchemy. Далее следуют Peewee и Django с близкими значениями. SQLModel несколько медленнее из-за накладных расходов на валидацию данных [5, 18].

Pony в данном тесте не представлен, поскольку библиотека не поддерживает пакетное создание [3].

4.4 Тест 4. Выборка всех записей

В сценарии измеряется время операции чтения всех строк таблицы и материализации результата в ORM-объекты. Результаты приведены в табл. 4.

Лучшее время демонстрирует Peewee, поскольку тратит меньше ресурсов на преобразование строк результата в ORM-объекты и сопутствующие служебные операции [2]. Сопоставимые значения показывают Django и SQLAlchemy. SQLModel демонстрирует несколько более высокое время выполнения. Pony заметно медленнее остальных ORM из-за более тяжёлого механизма построения объектов и работы с контекстом [3].

Табл. 4. Выборка всех записей – время выполнения, с.
Table 4. Retrieval of all records – execution time, s.

ORM	avg	p50	p99
Peewee	0,054	0,052	0,063
Pony	0,150	0,148	0,170
Django	0,084	0,080	0,109
SQLAlchemy	0,080	0,080	0,089
SQLModel	0,098	0,097	0,106

4.5 Тест 5. Выборка первой записи

В тесте измеряется время выборки первой записи (упорядочивание по первичному ключу и ограничение результата одной строкой). Результаты приведены в табл. 5.

Табл. 5. Выборка первой записи – время выполнения, мс.
Table 5. Retrieval of the first record – execution time, ms.

ORM	avg	p50	p99
Peewee	0,521	0,501	0,634
Pony	0,376	0,338	0,553
Django	0,655	0,636	0,778
SQLAlchemy	1,006	0,990	1,305
SQLModel	0,990	0,957	1,771

Pony демонстрирует лучший результат благодаря кэшированию, описанному в тесте 1. Показатели Peewee и Django оказываются близкими. SQLModel и SQLAlchemy медленнее по причине, описанной в тесте 1: более высокие накладные расходы ORM-слоя при одиночных операциях.

4.6 Тест 6. Выборка по первичному ключу

В тесте выполняется замер времени точечного чтения одной записи по первичному ключу: выполнение SELECT-запроса и материализацию результата в ORM-объект. Результаты приведены в табл. 6.

Лучший результат демонстрирует Pony за счёт кэширования построения запросов, описанного в тесте 1. Peewee и Django демонстрируют схожие значения: они уступают Pony, но заметно опережают SQLAlchemy и SQLModel, которые показывают более высокое время выполнения по причине, описанной в тесте 1 (накладные расходы сессии/контекста и служебных структур, заметные на лёгких одиночных операциях).

Табл. 6. Выборка по первичному ключу – время выполнения, мс.
Table 6. Retrieval by primary key – execution time, ms.

ORM	avg	p50	p99
Peewee	0,518	0,508	0,628
Pony	0,298	0,261	0,520
Django	0,688	0,654	0,883
SQLAlchemy	1,041	1,078	1,312
SQLModel	0,926	0,838	1,159

4.7 Тест 7. Выборка с ограничением и включением атрибутов связанной записи

В сценарии измеряется время выборки с ограничением по числу строк: выполняется SELECT-запрос с LIMIT 250, при этом атрибуты связанной записи выбираются в том же SQL-запросе с использованием INNER JOIN. Возвращаемые данные представлены кортежами, а не экземплярами ORM-объектов. Результаты приведены в табл. 7.

Табл. 7. Выборка с ограничением и включением атрибутов связанной записи – время выполнения, мс.
Table 7. Retrieval with limit including attributes of related record – execution time, ms.

ORM	avg	p50	p99
Peewee	0,534	0,532	0,669
Pony	0,515	0,448	1,073
Django	1,180	1,165	1,577
SQLAlchemy	1,563	1,520	2,212
SQLModel	1,402	1,379	1,837

Лучшие результаты демонстрируют Pony и Peewee. Для Pony это согласуется с эффектом кэширования построения запросов, описанным в тесте 1.

Далее следует Django: несмотря на сходный с Peewee SQL-запрос, время выполнения оказывается выше, что указывает на заметные накладные расходы ORM на стороне приложения (формирование запроса и обработка результата).

SQLModel и SQLAlchemy демонстрируют наибольшие значения времени выполнения по той же причине, что и в тесте 1: накладные расходы ORM-слоя и управление сессией/контекстом заметнее при относительно лёгких одиночных операциях.

4.8 Тест 8. Выборка с фильтром, пагинацией со смещением и сортировкой

В данном тесте измеряется время выполнения выборки с фильтрацией и пагинацией: выполняется SELECT-запрос с условием отбора (WHERE), сортировкой результата (ORDER BY по одному из полей) и пагинацией со смещением (LIMIT/OFFSET). Результаты приведены в табл. 8.

Лучшее время демонстрирует Peewee. Далее следуют SQLAlchemy и SQLModel: на более сложных выражениях относительная доля фиксированных накладных расходов (ведение

сессии/контекста и служебных структур) становится менее заметной, поэтому их результаты более конкурентоспособны, чем в тестах с более простыми одиночными операциями.

Табл. 8. Выборка с фильтром, пагинацией со смещением и сортировкой – время выполнения, мс.
Table 8. Filtered retrieval with offset pagination and sorting – execution time, ms.

ORM	avg	p50	p99
Peewee	6,209	5,985	7,466
Pony	10,206	9,678	15,733
Django	8,151	8,258	8,987
SQLAlchemy	6,685	6,739	7,211
SQLModel	7,107	7,221	7,745

Для Django этот сценарий оказывается более тяжёлым, что можно объяснить повышенной стоимостью обработки результата и служебной логики ORM при усложнении запроса.

Pony в данном тесте показывает худший результат: при возврате 250 записей заметную долю времени занимают материализация в ORM-объекты и постобработка данных, тогда как выигрыш от кэширования построения запросов становится менее заметным. В тесте 7 этот эффект не проявлялся, поскольку результат возвращался в виде кортежей (без создания объектов модели).

4.9 Тест 9. Одиночное обновление объекта

Сценарий позволяет оценить суммарное время операции обновления и сохранения одного объекта: формирование новых значений на стороне приложения, их обработку ORM и драйвером, а также выполнение операции UPDATE на стороне СУБД. Результаты приведены в табл. 9.

Табл. 9. Одиночное обновление объекта – время выполнения, мс.
Table 9. Single object update – execution time, ms.

ORM	avg	p50	p99
Peewee	0,834	0,850	0,950
Pony	1,064	1,069	1,460
Django	1,210	1,237	1,421
SQLAlchemy	1,821	1,831	2,045
SQLModel	1,824	1,826	2,086

Лучший результат демонстрирует Peewee, что согласуется с общей тенденцией тестов одиночных CRUD-операций: библиотека характеризуется меньшей стоимостью создания и сопровождения ORM-объектов, а также меньшими служебными накладными расходами.

Далее следуют Pony и Django с сопоставимыми значениями. Pony достигает такого результата благодаря кэшированию построения запросов, описанному в тесте 1.

SQLAlchemy и SQLModel снова оказываются самыми медленными по причине, описанной в тесте 1: более высокие накладные расходы ORM-слоя (контекст/сессия и служебные структуры).

4.10 Тест 10. Обновление 2500 объектов в транзакции

В тесте выполняется замер суммарного времени операции обновления 2500 объектов и их сохранения в базе данных в рамках одной крупной транзакции, что позволяет оценить эффективность выполнения операции обновления при использовании транзакций. Результаты приведены в табл. 10.

Табл. 10. Обновление 2500 объектов в транзакции – время выполнения, с.
Table 10. Update of 2500 objects in a transaction – execution time, s.

ORM	avg	p50	p99
Peewee	0,974	0,936	1,290
Pony	0,784	0,674	1,899
Django	1,624	1,596	1,990
SQLAlchemy	0,211	0,207	0,268
SQLModel	0,209	0,203	0,242

SQLAlchemy и SQLModel демонстрируют лучшие результаты благодаря группировке обновлений, аналогичной группировке вставок из теста 2. Показатели Pony идут следом благодаря эффекту кэширования, описанному в тесте 1. Django и Peewee показывают худший результат из-за реализации полного цикла генерации запросов и их выполнения внутри транзакционной обёртки.

4.11 Тест 11. Пакетное обновление 2500 объектов

В сценарии измеряется суммарное время обновления 2500 объектов одной операцией (одним UPDATE-запросом с массивом первичных ключей тех строк, которые нужно обновить). Результаты приведены в табл. 11.

Табл. 11. Пакетное обновление 2500 объектов – время выполнения, с.
Table 11. Bulk update of 2500 objects – execution time, s.

ORM	avg	p50	p99
Peewee	0,051	0,047	0,070
Pony	–	–	–
Django	0,085	0,084	0,119
SQLAlchemy	0,065	0,066	0,088
SQLModel	0,070	0,068	0,102

Наименьшее суммарное время выполнения операции показывает Peewee. SQLAlchemy и SQLModel демонстрируют близкие значения. Django в этом сценарии уступает, что говорит о более высоких накладных расходах на построение пакетного запроса и передачу параметров на уровне ORM.

Pony в данном тесте не представлен, поскольку библиотека не поддерживает пакетное обновление [3].

4.12 Тест 12. Одиночное удаление объекта

В тесте измеряется суммарное время операции удаления объекта: обработку операции на уровне ORM и выполнение SQL-запроса на стороне СУБД. В зависимости от реализации и настроек конкретной ORM операция может включать не только DELETE, но и дополнительные служебные запросы (например, предварительные SELECT). Результаты приведены в табл. 12.

Табл. 12. Одиночное удаление объекта – время выполнения, мс.
Table 12. Single object deletion – execution time, ms.

ORM	avg	p50	p99
Peewee	0,841	0,815	1,055
Pony	1,472	1,489	1,853
Django	1,252	1,201	1,566
SQLAlchemy	1,786	1,749	2,222
SQLModel	1,757	1,727	2,172

Лучший результат демонстрирует Peewee, что согласуется с тенденцией одиночных тестов из-за более лёгкого ORM-слоя и меньших накладных расходов.

Django и Pony имеют сопоставимые результаты, однако для Pony наблюдается дополнительная нагрузка: библиотека продолжает строить дерево удаления даже при отсутствии каскадного удаления в модели. Это приводит к дополнительным SELECT-запросам без последующих действий над полученными объектами [3].

SQLAlchemy и SQLModel демонстрируют худший результат по причине более высоких накладных расходов ORM-слоя, описанной в тесте 1.

4.13 Тест 13. Удаление 2500 объектов в транзакции

Сценарий позволяет оценить суммарное время операции удаления 2500 объектов в рамках одной крупной транзакции: обработку операции на уровне ORM и выполнение SQL-запросов на стороне СУБД. В зависимости от реализации конкретной ORM удаление может выполняться как одним DELETE-запросом, так и циклом отдельных DELETE-запросов, а также включать дополнительные служебные запросы (например, предварительные SELECT-запросы). Результаты приведены в табл. 13.

Табл. 13. Удаление 2500 объектов в транзакции – время выполнения, с.
Table 13. Deletion of 2500 objects in a transaction – execution time, s.

ORM	avg	p50	p99
Peewee	1,002	0,895	2,081
Pony	0,665	0,651	0,951
Django	1,661	1,549	2,705
SQLAlchemy	0,212	0,212	0,243
SQLModel	0,218	0,213	0,267

SQLAlchemy и SQLModel демонстрируют лучшие результаты за счёт группировки удалений в один запрос в рамках одной крупной транзакции, аналогично группировке вставок в тесте 2 и группировке обновлений в тесте 10.

Далее следует Pony. Анализ трассировки SQL-запросов, полученной из встроенных средств SQL-логирования ORM, показывает, что Pony (как и в тесте 12) строит дерево удаления и выполняет один общий SELECT-запрос для предварительной выборки, после чего реализует цикл отдельных DELETE-запросов. Несмотря на дополнительный запрос выборки, ускорение относительно Peewee и Django объясняется эффектом кэширования построения запросов, отмеченным в тесте 1.

Django и Peewee демонстрируют худшие результаты из-за выполнения полного цикла генерации запросов и их выполнения внутри транзакционной обёртки.

4.14 Тест 14. Пакетное удаление 2500 объектов

В тесте выполняется замер суммарного времени удаления 2500 объектов одной операцией (одним DELETE-запросом с множеством первичных ключей). Результаты приведены в табл 14.

Табл. 14. Пакетное удаление 2500 объектов – время выполнения, с.
Table 14. Bulk deletion of 2500 objects – execution time, s.

ORM	avg	p50	p99
Peewee	0,160	0,151	0,238
Pony	0,163	0,162	0,195
Django	0,169	0,167	0,196
SQLAlchemy	0,123	0,122	0,141
SQLModel	0,119	0,118	0,136

Лучшее время демонстрируют SQLAlchemy и SQLModel, которые эффективно выполняют пакетные запросы. В данном сценарии накладные расходы SQLModel на валидацию не дают заметного эффекта.

Peewee, Pony и Django имеют сопоставимые результаты. При пакетном удалении Pony, в отличие от тестов 12 и 13, фактически не строит дерево удаления и также генерирует один SQL-запрос.

5. Заключение

Результаты тестирования продемонстрировали, что производительность исследуемых библиотек (Django, Peewee, Pony, SQLAlchemy и SQLModel) существенно зависит от характера выполняемых операций и сценариев нагрузки. В CRUD-операциях с одним объектом наименьшие накладные расходы и, как следствие, минимальное время выполнения продемонстрировал Peewee. Django и Pony в подобных сценариях показывают сопоставимые результаты: Django – благодаря оптимизированным базовым операциям, Pony – за счёт механизма кэширования компиляции запросов. В то же время SQLAlchemy и основанный на нём SQLModel с «тяжёлым» слоем абстракции оказываются медленнее на отдельных лёгких операциях из-за затрат на ведение сессии и других служебных структур.

При групповых операциях ситуация меняется. В сценариях вставки, обновления или удаления большого числа объектов внутри одной транзакции SQLAlchemy и SQLModel значительно опережают другие ORM благодаря агрегации операций: их архитектура

позволяет сгенерировать укрупнённые SQL-запросы, тогда как Peewee и Django по умолчанию выполняют серию отдельных запросов. Pony в массовых операциях также выигрывает от единоразовой компиляции запроса: повторные однотипные действия выполняются быстрее, однако отсутствие специальных методов пакетного создания и обновления в Pony не позволяет ему полноценно конкурировать в тестах, требующих одной общей операции над множеством записей. Отдельно следует отметить, что при использовании механизмов пакетной вставки, обновления или удаления разница между ORM сглаживается: все библиотеки затрачивают сопоставимое время на обработку, причём SQLAlchemy и SQLModel незначительно опережают Peewee и Django, демонстрируя преимущество в оптимизации передачи данных через драйвер.

Сравнение производительности операций чтения показало, что Peewee обеспечивает самую быструю материализацию результатов за счёт минимальных накладных расходов на каждый объект. Django и SQLAlchemy в простых запросах чтения несколько уступают за счёт дополнительных слоёв обработки результатов, однако с усложнением запросов (фильтрация, сортировка, соединение таблиц) разница между ними и Peewee сокращается. В сложных запросах (например, тест с фильтрацией, сортировкой и пагинацией) SQLAlchemy и SQLModel сумели приблизиться по скорости к Peewee, поскольку фиксированные затраты их механизма сессии составляют меньшую долю от общего времени выполнения более тяжёлого запроса. Pony ORM продемонстрировал противоречивые результаты на чтение: в ряде простых случаев Pony был одним из лидеров благодаря упомянутому кэшированию SQL-представления запроса, однако при выборках, возвращающих множество объектов или требующих обширной постобработки, Pony заметно уступал из-за затрат на создание объектов и ведение контекста.

Анализ 99-го перцентиля времени выполнения указывает на различия в устойчивости производительности, однако эти различия не стоит интерпретировать как безусловное преимущество одной ORM во всех сценариях. Для Peewee во многих тестах разрыв между медианой и p99 оставался умеренным, что согласуется с относительно небольшими накладными расходами библиотеки. Вместе с тем отдельные сценарии демонстрируют заметные отклонения: например, в тесте 14 значение p99 для Peewee примерно в полтора раза превышает медиану. Следовательно, полученные данные характеризуют не полное отсутствие пиковых задержек, а общую тенденцию, выявленную по совокупности сценариев. У SQLAlchemy и SQLModel также наблюдалась достаточно стабильная производительность, особенно в сценариях с массовыми операциями – благодаря единообразию выполнения серии однотипных действий разброс результатов тестирования оказался невелик. Pony в некоторых тестах продемонстрировал более существенный разрыв между медианой и p99. В целом, для практического использования это означает, что при выборе ORM следует учитывать не только среднюю скорость выполнения операции, но и предсказуемость времени отклика.

Разработанная методика и полученные результаты дают репрезентативную оценку производительности ORM в условиях, приближённых к промышленной эксплуатации для ключевых шаблонов операций. При обобщении выводов необходимо учитывать специфику данных, интенсивность и характер нагрузки, а также конфигурацию окружения. Совокупность эмпирических данных представляет практический ориентир для архитекторов высоконагруженных систем и одновременно формирует воспроизводимую эталонную базу для последующих исследований, охватывающих сравнение различных поддерживаемых СУБД, поведение ORM при высокой конкуренции и масштабировании, а также выявление и систематизацию типовых ошибок производительности.

Список литературы / References

[1]. Django Software Foundation. Django documentation (v5.2). Available at: https://docs.djangoproject.com/en/5.2/, accessed: 28.11.2025.

[2]. Peewee. Peewee documentation. Available at: https://docs.peewee-orm.com/en/latest/, accessed: 20.01.2026.
[3]. PonyORM. Pony ORM documentation. Available at: https://docs.ponyorm.org/, accessed: 28.11.2025.
[4]. SQLAlchemy. SQLAlchemy documentation (v2.0). Available at: https://docs.sqlalchemy.org/en/20/, accessed: 30.11.2025.
[5]. tiangolo. SQLModel documentation. Available at: https://sqlmodel.tiangolo.com/, accessed: 21.01.2026.
[6]. Transaction Processing Performance Council. TPC Benchmarks Overview. Available at: https://www.tpc.org/information/benchmarks5.asp, accessed: 11.06.2026.
[7]. Cooper B.F., Silberstein A., Tam E., Ramakrishnan R., Sears R. Benchmarking Cloud Serving Systems with YCSB. In Proc. of the 1st ACM Symposium on Cloud Computing (SoCC 2010), 2010, pp. 143–154. DOI: 10.1145/1807128.1807152. Available at: https://github.com/brianfrankcooper/YCSB, accessed: 11.06.2026.
[8]. Curino C., Difallah D.E., Pavlo A., Cudré-Mauroux P. Benchmarking OLTP/Web Databases in the Cloud: the OLTP-Bench Framework. In Proc. of the 4th International Workshop on Cloud Data Management (CloudDB 2012), 2012, pp. 17–20. DOI: 10.1145/2390021.2390025. Available at: https://db.cs.cmu.edu/projects/benchbase/, accessed: 12.06.2026.
[9]. TechEmpower. TechEmpower Framework Benchmarks. Available at: https://www.techempower.com/benchmarks/, accessed: 13.06.2026.
[10]. Lorenz M., Rudolph J.-P., Hesse G., Uflacker M., Plattner H. Object-Relational Mapping Revisited – A Quantitative Study on the Impact of Database Technology on O/R Mapping Strategies. In Proc. of the 50th Hawaii International Conference on System Sciences (HICSS 2017), 2017, pp. 4877–4886. DOI: 10.24251/HICSS.2017.592. Available at: https://aisel.aisnet.org/hicss-50/os/enterprise_architecture/9/, accessed: 14.06.2026.
[11]. Chen T.-H., Shang W., Jiang Z.M., Hassan A.E., Nasser M., Flora P. Detecting Performance Anti-patterns for Applications Developed using Object-Relational Mapping. In Proc. of the 36th International Conference on Software Engineering (ICSE 2014), 2014, pp. 1001–1012. DOI: 10.1145/2568225.2568259. Available at: https://sailresearch.github.io/sail-website/data/pdfs/ICSE2014_DetectingPerformanceAnti-patternsForApplicationsDevelopedUsingObject-RelationalMapping.pdf, accessed: 14.06.2026.
[12]. Wiatrowski T. Comparative Analysis of ORM Systems for the .NET Platform. Journal of Computer Sciences Institute, 2024, vol. 31, pp. 97–102. DOI: 10.35784/jcsi.6012. Available at: https://ph.pollub.pl/index.php/jcsi/article/view/6012, accessed: 15.06.2026.
[13]. Attala J., Khemapatpapan C. Comparing the Performance of ORMs. Journal of Computer and Creative Technology, 2025, vol. 3, no. 2, pp. 201–213. DOI: 10.14456/jcct.2025.16. Available at: https://so13.tci-thaijo.org/index.php/jcct/article/view/2330, accessed: 16.06.2026.
[14]. Python Software Foundation. PEP 249 – Python Database API Specification v2.0. Available at: https://peps.python.org/pep-0249/, accessed: 26.11.2025.
[15]. psycopg. psycopg 3 documentation. Available at: https://www.psycopg.org/psycopg3/docs/, accessed: 27.11.2025.
[16]. psycopg. psycopg 2.9 documentation. Available at: https://www.psycopg.org/docs/, accessed: 27.11.2025.
[17]. Pydantic. Pydantic documentation (v2.12). Available at: https://docs.pydantic.dev/2.12/, accessed: 21.01.2026.
[18]. Bednarz B., Miłosz M. Benchmarking the performance of Python web frameworks. Journal of Computer Sciences Institute, 2025, vol. 36, pp. 336–341. DOI: 10.35784/jcsi.7738. Available at: https://ph.pollub.pl/index.php/jcsi/article/view/7738, accessed: 22.01.2026.
[19]. Fowler M. Unit of Work. Available at: https://martinfowler.com/eaaCatalog/unitOfWork.html, accessed: 30.11.2025.
[20]. Fowler M. Active Record. Available at: https://martinfowler.com/eaaCatalog/activeRecord.html, accessed: 30.11.2025.
[21]. Xu Z., Zhu J., Yang L., Zuo C. Mining the Relationship between Object-Relational Mapping Performance Anti-patterns and Code Clones. In Proc. of the 35th International Conference on Software Engineering and Knowledge Engineering (SEKE 2023), 2023, pp. 136–141. DOI: 10.18293/SEKE2023-161. Available at: https://ksiresearch.org/seke/seke23paper/paper161.pdf, accessed: 01.12.2025.
[22]. Tang C., Wang Z., Zhang X., Yu Q., Zang B., Guan H., Chen H. Ad Hoc Transactions in Web Applications: The Good, the Bad, and the Ugly. In Proc. of the 2022 International Conference on Management of Data (SIGMOD 2022), 2022, pp. 4–18. DOI: 10.1145/3514221.3526120. Available at: https://ipads.se.sjtu.edu.cn/_media/publications/concerto-sigmod22.pdf, accessed: 02.12.2025.
[23]. PostgresPro. Demo database. Available at: https://postgrespro.ru/education/demodb, accessed: 29.11.2025.
[24]. Reichelt D.G., Jung R., van Hoorn A. Overhead Measurement Noise in Different Runtime Environments. Softwaretechnik-Trends, 2024, vol. 44, no. 4, pp. 12–14. Available at: https://arxiv.org/abs/2411.05491, accessed: 28.01.2026.
[25]. Dean J., Barroso L.A. The Tail at Scale. Communications of the ACM, 2013, vol. 56, no. 2, pp. 74–80. DOI: 10.1145/2408776.2408794. Available at: https://dl.acm.org/doi/10.1145/2408776.2408794, accessed: 27.01.2026.

Информация об авторах / Information about authors

Александр Евгеньевич ЧЕРНОБРОВЕНКО – студент Белгородского государственного технологического университета им. В.Г. Шухова. Сфера научных интересов: машинное обучение, цифровые технологии, проектирование и разработка баз данных, алгоритмы и структуры данных, параллельное программирование.

Alexander Evgenyevich CHERNOBROVENKO – Student of the Belgorod State Technological University named after V.G. Shukhov. Research interests: machine learning, digital technologies, database design and development, algorithms and data structures, parallel programming.

Денис Андреевич ЛЫЛОВ – студент Белгородского государственного технологического университета им. В.Г. Шухова. Сфера научных интересов: машинное обучение, проектирование и разработка баз данных, информационно-коммуникационные технологии.

Denis Andreevich LYLOV – Student of the Belgorod State Technological University named after V.G. Shukhov. Research interests: machine learning, database design and development, information and communication technologies.

Максим Владимирович ПАНЧЕНКО – старший преподаватель кафедры программного обеспечения вычислительной техники и автоматизированных систем Белгородского государственного технологического университета им. В.Г. Шухова. Сфера научных интересов: технологии проектирования и разработки баз данных, методы интеллектуального анализа данных, нейро-нечеткие системы.

Maksim Vladimirovich PANCHENKO – Senior Lecturer at the Department of Software for Computer Engineering and Automated Systems, Belgorod State Technological University named after V.G. Shukhov. Research interests: technologies for database design and development, methods of intelligent data analysis, neuro-fuzzy systems.