Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004.651:004.738.5

СРАВНИТЕЛЬНЫЙ АНАЛИЗ POSTGRESQL, MYSQL И SQLITE ПРИ РАЗРАБОТКЕ НЕБОЛЬШИХ ВЕБ-ПРИЛОЖЕНИЙ

Корчагин Павел Алексеевич,
Студент, Поволжский Государственный Университет Телекоммуникаций и Информатики, г. Самара, Российская Федерация
E-mail: pasha.korchagin.03@mail.ru

Ахметшина Элеонора Газинуровна,
К.т.н., доцент каф. ИРС
Поволжский Государственный Университет Телекоммуникаций и Информатики, г. Самара, Российская Федерация
E-mail: e.ahmetshina@psuti.ru

Аннотация

Актуальность исследования обусловлена необходимостью обоснованного выбора реляционной СУБД при разработке небольшого веб-приложения с REST API, когда учебная литература даёт фрагментарное сравнение на прикладном уровне. Разработчик учебного или прототипного сервиса нередко начинает с SQLite и лишь на этапе сдачи проекта задумывается о переходе на серверную СУБД; при этом отсутствуют количественные данные о том, насколько изменится время отклика API при такой замене на типовых CRUD-операциях. Цель работы — выполнить сравнительный анализ PostgreSQL, MySQL и SQLite на едином прототипе и измерить время отклика API для типовых CRUD-операций. Методика включает разработку приложения на FastAPI и SQLAlchemy, подключение трёх СУБД через общий ORM-слой, предзаполнение базы 200 записями и выполнение 100 последовательных HTTP-запросов по пяти сценариям (list, get, create, update, count). Результаты показали, что SQLite обеспечивает наименьшие задержки в четырёх из пяти сценариев (get_by_id — 2,11 мс, count — 2,01 мс); исключение — create_item, где быстрее PostgreSQL (9,71 мс). PostgreSQL и MySQL демонстрируют сопоставимую производительность на чтении списка и агрегации с запасом по масштабируемости. Выводы: SQLite рекомендуется для локальной разработки и учебных проектов; PostgreSQL или MySQL — при переходе к эксплуатации с несколькими клиентами и ростом объёма данных.

Ключевые слова: PostgreSQL; MySQL; SQLite; веб-приложение; CRUD; производительность; сравнительный анализ; СУБД.

A COMPARATIVE ANALYSIS OF POSTGRESQL, MYSQL, AND SQLITE IN SMALL WEB APPLICATION DEVELOPMENT

Korchagin Pavel Alekseevich,
Student, Volga Region State University of Telecommunications and Informatics, Samara, Russian Federation
Email: pasha.korchagin.03@mail.ru

Akhmetshina Eleonora Gazinurovna,
PhD, Associate Professor, Department of Information Systems
Volga Region State University of Telecommunications and Informatics, Samara, Russian Federation
Email: e.ahmetshina@psuti.ru

ABSTRACT

The relevance of this study stems from the need to make an informed choice of a relational DBMS when developing a small web application with a REST API, when educational literature provides a fragmented comparison at the application level. Developers of educational or prototype services often start with SQLite and only consider switching to a server-side DBMS at the project delivery stage. However, there is a lack of quantitative data on how such a change will affect the API response time for typical CRUD operations. The objective of this study is to perform a comparative analysis of PostgreSQL, MySQL, and SQLite on a single prototype and measure the API response time for typical CRUD operations. The methodology includes developing an application using FastAPI and SQLAlchemy, connecting three DBMSs via a common ORM layer, pre-populating the database with 200 records, and executing 100 sequential HTTP requests using five scenarios (list, get, create, update, count). The results showed that SQLite provides the lowest latency in four out of five scenarios (get_by_id - 2.11 ms, count - 2.01 ms); the exception is create_item, where PostgreSQL is faster (9.71 ms). PostgreSQL and MySQL demonstrate comparable performance for list reads and aggregations, with some scalability to spare. Conclusions: SQLite is recommended for local development and educational projects; PostgreSQL or MySQL are recommended for transitioning to multi-tenant production and data growth.

Keywords: PostgreSQL; MySQL; SQLite; web application; CRUD; performance; benchmarking; DBMS.

Введение.

Выбор системы управления базами данных (СУБД) — одно из ключевых проектных решений при создании веб-приложения. От него зависят скорость разработки, стоимость сопровождения, устойчивость к нагрузке и возможность дальнейшего масштабирования. Для небольших проектов разработчики часто рассматривают встраиваемую SQLite как простой стартовый вариант, тогда как PostgreSQL и MySQL традиционно применяются в промышленной эксплуатации [2; 4].

Современные веб-сервисы на Python всё чаще строятся по схеме «REST API + ORM + реляционная СУБД»: сервер приложений обрабатывает HTTP-запросы, ORM формирует SQL-запросы, а СУБД обеспечивает персистентность. При этом итоговое время отклика складывается из задержек нескольких уровней — разбора запроса во фреймворке, работы пула соединений, выполнения SQL, сериализации результата в JSON и передачи по сети.

Сравнение СУБД только по синтетическим тестам не отражает поведение типового учебного или малого коммерческого приложения [6].

При этом формальное сравнение производительности на идентичном прикладном коде в учебной литературе представлено фрагментарно: чаще анализируются отдельные СУБД или синтетические бенчмарки без привязки к типовому веб-контуру [2; 5]. На практике же разработчик сталкивается с необходимостью выбрать хранилище именно для REST-сервиса с ORM-слоем, где значимы не только «сырые» показатели СУБД, но и накладные расходы фреймворка, сериализации и сетевого взаимодействия [6].

Объект исследования — REST API небольшого веб-приложения на стеке FastAPI/SQLAlchemy. Предмет исследования — влияние выбора реляционной СУБД (PostgreSQL, MySQL, SQLite) на время отклика REST API и 95-й перцентиль задержки (p95) при типовых CRUD-операциях. Научная новизна заключается в эмпирическом сопоставлении трёх распространённых СУБД на одном прикладном стеке Python/FastAPI с фиксированным набором CRUD-сценариев и измерением не только среднего времени отклика, но и p95, отражающего «хвост» распределения задержек.

Практическая значимость работы состоит в том, что полученные результаты и рекомендации могут быть использованы при выборе СУБД в рамках учебной практики, дипломных проектов и прототипов веб-сервисов, где ограничены ресурсы на развёртывание инфраструктуры, но требуется обоснованное решение о хранилище данных.

Обзор подходов к выбору СУБД.

PostgreSQL относится к объектно-реляционным СУБД с развитой поддержкой стандарта SQL, расширяемостью и механизмами транзакционной целостности [4; 7]. Система поддерживает сложные типы данных, оконные функции, полнотекстовый поиск и внешние ключи, что делает её предпочтительной при проектировании приложений с жёсткими требованиями к согласованности данных. PostgreSQL использует многоверсионное управление параллелизмом (MVCC), что снижает блокировки при параллельном чтении, но накладывает накладные расходы на vacuum и управление версиями строк [4].

MySQL широко используется в веб-разработке благодаря зрелой экосистеме, наличию готовых хостинг-решений и интеграции с LAMP/LEMP-стеком [3]. В учебных и малых коммерческих проектах MySQL часто выбирают из-за простоты первичной настройки и обширной документации. Движок InnoDB обеспечивает транзакционность и внешние ключи; при этом поведение планировщика и конфигурация буферного пула существенно влияют на время отклика при смешанной нагрузке [8].

SQLite представляет собой встраиваемую СУБД без отдельного серверного процесса: вся база хранится в одном файле, что упрощает развёртывание, но ограничивает модель конкурентного доступа при интенсивной записи [9]. Библиотека SQLite встраивается в процесс приложения; обращения к данным выполняются через системные вызовы без сетевого протокола, что сокращает задержку на малых объёмах, но создаёт риск блокировки файла при конкурентной записи из нескольких процессов.

В обзорных работах подчёркивается, что выбор СУБД должен учитывать не только абсолютную производительность, но и требования к масштабированию, администрированию и отказоустойчивости [2; 6]. Для небольших приложений с умеренной нагрузкой критичны простота развёртывания и предсказуемость времени отклика при типовых запросах [5]. Сопоставительные исследования на прикладном уровне позволяют снизить риск преждевременной оптимизации и обоснованно выбрать хранилище на ранних этапах жизненного цикла проекта.

Постановка задачи.

Цель работы — выполнить сравнительный анализ PostgreSQL, MySQL и SQLite в условиях небольшого веб-приложения с одинаковой бизнес-логикой и измерить время отклика API для повторяющихся операций.

Для достижения поставленной цели сформулированы следующие задачи исследования:

Проанализировать особенности PostgreSQL, MySQL и SQLite в контексте разработки небольших веб-приложений с REST API.

Разработать единый прототип с REST API и подключить три СУБД через общий слой ORM (SQLAlchemy).

Сформировать тестовый набор данных и сценарии измерений, охватывающие операции чтения и записи.

Сравнить среднее время отклика и 95-й перцентиль (p95) для каждой СУБД и каждого сценария.

Сформулировать рекомендации по выбору СУБД для учебных, прототипных и малых эксплуатационных проектов.

Гипотеза исследования: на малых объёмах данных SQLite обеспечит наименьшее время отклика API за счёт отсутствия сетевого взаимодействия с сервером СУБД; PostgreSQL и MySQL покажут сопоставимые результаты на чтении, но превосходство одной из серверных СУБД проявится на операциях записи в зависимости от драйвера и конфигурации.

Архитектура прототипа.

Прототип построен по трёхзвенной схеме «клиент — сервер приложений — хранилище данных». Серверная часть реализована на Python с использованием фреймворка FastAPI; доступ к данным — через SQLAlchemy 2.0 с декларативными моделями и сессиями. REST API предоставляет эндпоинты для типовых CRUD-операций над сущностью «запись» (список, чтение по id, создание, обновление, подсчёт количества).

Слой SQLAlchemy абстрагирует различия СУБД: один и тот же код моделей и запросов работает с SQLite, PostgreSQL и MySQL при смене строки подключения DATABASE_URL. Для PostgreSQL используется драйвер psycopg 3, для MySQL — pymysql, для SQLite — встроенный драйвер через URI sqlite:/// [11]. Такой подход соответствует практике разработки, когда СУБД выбирается на этапе конфигурации, а не переписывания кода [1; 10].

Кэширование в прототипе отключено (CACHE_ENABLED=0), чтобы измеряемое время отклика отражало именно работу выбранной СУБД, а не промежуточного кэш-слоя. Каждый HTTP-запрос проходит полный цикл: маршрутизация FastAPI → получение сессии БД → SQL через ORM → сериализация ответа в JSON.

Методика эксперимента.

Прототип — небольшое веб-приложение на FastAPI и SQLAlchemy [1]. Реализованы эндпоинты GET /api/items (список), GET /api/items/{id} (чтение по идентификатору), POST /api/items (создание), PUT /api/items/{id} (обновление) и GET /api/stats/count (агрегирующий запрос COUNT). Приложение не использует кэширование, чтобы изолировать влияние именно СУБД на время отклика. Один и тот же исходный код подключается к трём хранилищам через переменную окружения DATABASE_URL, что обеспечивает сопоставимость результатов [10; 11].

Параметры экспериментального стенда приведены в таблице 1. PostgreSQL и MySQL развёрнуты в контейнерах Docker на локальной машине; SQLite используется как файловая база в каталоге приложения. Перед серией измерений для каждой СУБД база предзаполняется 200 тестовыми записями; в рамках одного сценария выполняется 100 последовательных HTTP-запросов к API. Сценарии запускаются в фиксированном порядке

(list_items → get_by_id → create_item → update_item → count_items), поэтому сценарий create_item увеличивает число записей в базе.

Таблица 1. Параметры экспериментального стенда

Параметр	Значение
ОС	Linux
Python	3.14
FastAPI	0.115
SQLAlchemy	2.0
PostgreSQL	16 (Docker)
MySQL	8.4 (Docker)
SQLite	файл data/items.sqlite3
Объём данных	200 записей перед тестом
Число итераций	100 на сценарий

Использовались пять сценариев: list_items (полный список), get_by_id (чтение одной записи), create_item (вставка), update_item (обновление), count_items (агрегация). Для каждого сценария фиксировались среднее время отклика (мс), p95, минимум и максимум. Автоматизация прогона выполнена скриптом с использованием httpx; результаты сохранялись в формате JSON для последующего построения таблиц и графиков.

Методика измерения включает полный цикл обработки запроса на стороне клиента: от отправки HTTP-запроса до получения полного тела ответа. В метрику входят задержки localhost, обработки в FastAPI, работы ORM и СУБД, сериализации JSON. Для каждой СУБД сервер приложений запускался на отдельном порту; между прогонами по разным СУБД процесс uvicorn перезапускался, чтобы исключить влияние предыдущего состояния пула соединений.

Результаты эксперимента.

Результаты измерений сведены в таблице 2. На рисунке 1 представлено сравнение среднего времени отклика по сценариям для трёх СУБД. Эксперимент показал, что разброс значений между СУБД наиболее заметен для операций create_item и update_item, тогда для list_items различия умеренные (4,85-6,63 мс).

Таблица 2. Сравнение времени отклика API при разных СУБД

СУБД	Сценарий	Среднее, мс	p95, мс
sqlite	list_items	4.85	5.03
	get_by_id	2.11	2.38
	create_item	13.09	16.57
	update_item	4.33	8.58
	count_items	2.01	2.75
postgresql	list_items	6.48	9.50
	get_by_id	4.83	5.45
	create_item	9.71	14.61

СУБД	Сценарий	Среднее, мс	p95, мс
	update_item	7.58	9.40
	count_items	5.15	5.73
mysql mysql	list_items	6.63	9.17
	get_by_id	3.29	5.28
	create_item	14.44	21.79
	update_item	11.42	11.81
	count_items	5.08	7.43

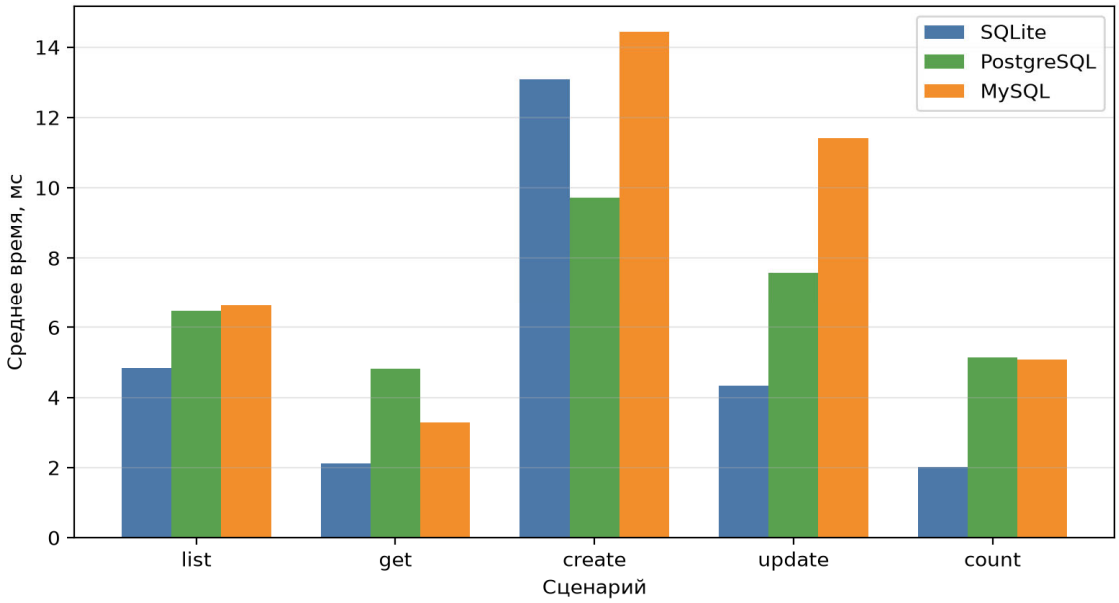

Рисунок 1. Среднее время отклика API по сценариям тестирования.

Анализ операций чтения. При малом объёме данных SQLite демонстрирует минимальные задержки за счёт отсутствия сетевого взаимодействия с сервером СУБД и низких накладных расходов [9]. Наиболее быстрыми среди сценариев SQLite оказались get_by_id и count_items — 2,11 и 2,01 мс соответственно (таблица 2). Операция list_items при SQLite (4,85 мс) также быстрее серверных СУБД (6,48 и 6,63 мс), поскольку выборка 200 записей и сериализация JSON выполняются без сетевого round-trip к отдельному процессу БД.

PostgreSQL и MySQL показывают сопоставимые результаты на чтении списка и агрегации (разброс менее 2 мс); при чтении по идентификатору MySQL (3,29 мс) опережает PostgreSQL (4,83 мс). Показатель p95 для list_items у PostgreSQL (9,50 мс) и MySQL (9,17 мс) существенно выше среднего, что указывает на эпизодические «выбросы» задержки — вероятно, из-за прогрева буферного пула или сборки мусора в контейнере [4; 8].

Анализ операций записи. Картина различается по сценариям. При создании записи (create_item) наименьшую задержку показала PostgreSQL — 9,71 мс; SQLite (13,09 мс) и MySQL (14,44 мс) отстают. При обновлении (update_item) быстрее всех оказалась SQLite (4,33 мс), PostgreSQL заняла промежуточное положение (7,58 мс), MySQL продемонстрировала наибольшую задержку (11,42 мс). Наибольший разброс p95 наблюдается у MySQL на create_item (21,79 мс), что может быть связано с fsync и журналированием InnoDB в конфигурации Docker по умолчанию [8].

Сводная оценка. Таблица 3 фиксирует лидирующую СУБД по среднему времени отклика в каждом сценарии. SQLite лидирует в четырёх из пяти случаев; PostgreSQL — только в create_item.

Таблица 3. Лидирующая СУБД по сценариям (минимальное среднее время отклика).

Сценарий	SQLite, мс	PostgreSQL, мс	MySQL, мс	Лидер
list_items	4,85	6,48	6,63	SQLite
get_by_id	2,11	4,83	3,29	SQLite
create_item	13,09	9,71	14,44	PostgreSQL
update_item	4,33	7,58	11,42	SQLite
count_items	2,01	5,15	5,08	SQLite

Для операций чтения списка накладные расходы ORM и сериализации JSON сопоставимы с затратами на выполнение SQL-запроса. При росте объёма таблицы и числа одновременных клиентов преимущество встраиваемой СУБД может снижаться из-за блокировок файла и отсутствия полноценного параллелизма записи [2; 12].

Обсуждение и рекомендации.

Для учебных и прототипных проектов SQLite оправдана на этапе разработки: минимальная настройка, воспроизводимость эксперимента, достаточная скорость на малых выборках [5; 9]. Студенту или начинающему разработчику не требуется поднимать отдельный сервер БД, что ускоряет старт проекта и упрощает сдачу лабораторных работ на одной машине. Файл базы можно включить в репозиторий или передавать преподавателю для проверки.

Для развёртывания небольшого веб-сервиса с перспективой роста данных и несколькими одновременными пользователями целесообразны PostgreSQL или MySQL. PostgreSQL предпочтительна при необходимости строгой целостности, сложных запросов, расширений и соответствия стандарту SQL [4; 7]; MySQL — при ориентации на типовый LAMP-стек и готовую хостинг-инфраструктуру [3; 8]. Выбор между серверными СУБД в условиях данного эксперимента не даёт кратного выигрыша в скорости на чтении, но определяет долгосрочную гибкость архитектуры.

Сравнение по критериям выбора для малых веб-приложений: SQLite выигрывает по простоте развёртывания и задержке чтения на малых данных; PostgreSQL — по целостности, расширяемости и производительности вставки в данном эксперименте; MySQL — по доступности на shared-хостинге, но показывает наибольшие задержки записи в тестовой конфигурации. Альтернативные подходы без смены СУБД включают оптимизацию индексов, connection pooling и переход на асинхронный режим SQLAlchemy; на малых объёмах выигрыш от смены СУБД сопоставим с тюнингом приложения, а при росте числа клиентов серверные СУБД становятся необходимы не столько ради скорости одного запроса, сколько ради модели доступа [12].

Ограничения и направления дальнейших исследований. Тестирование выполнено на одной машине без имитации высокой конкуренции записи и без репликации; сценарии выполнялись последовательно, а create_item увеличивал объём таблицы в ходе прогона. Не исследовано влияние размера выборки (1000, 10000 записей), параллельных клиентов и асинхронного режима SQLAlchemy. Результаты отражают класс нагрузки «небольшое приложение с последовательными запросами», а не промышленный кластер. Дальнейшая работа может включить нагрузочное тестирование с параллельными запросами, сравнение с пулом PgBouncer и оценку стоимости миграции с SQLite на серверную СУБД [12].

Заключение.

В работе выполнен сравнительный анализ PostgreSQL, MySQL и SQLite на едином веб-прототипе с REST API на стеке FastAPI/SQLAlchemy. Реализован воспроизводимый стенд с переключением СУБД через DATABASE_URL и автоматизированным прогоном пяти CRUD-сценариев. Проведено измерение среднего времени отклика и p95 для 100 последовательных запросов на каждый сценарий.

Экспериментально подтверждено, что при объёме около 200 записей SQLite обеспечивает наименьшее среднее время отклика в четырёх из пяти сценариев; исключение — create_item, где быстрее PostgreSQL (таблица 2, таблица 3, рисунок 1). Серверные СУБД демонстрируют сопоставимую производительность на операциях чтения списка и агрегации, но создают запас по масштабируемости, конкурентному доступу и администрированию.

Рекомендуется использовать SQLite для локальной разработки, прототипирования и учебных проектов с одним пользователем. PostgreSQL целесообразна при необходимости строгой целостности и лучшей производительности вставки на серверном стеке; MySQL — при развёртывании на типовом web-хостинге. Полученные результаты могут использоваться при выборе СУБД в рамках учебной практики и при проектировании малых веб-сервисов.

Список литературы:

1. Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул ; под ред. Л. Г. Гагариной. — Москва: ИД «Форум»: ИНФРА-М, 2018. — 400 с. — ISBN 978-5-8199-0812-9.
2. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт ; пер. с англ. — 8-е изд. — Москва : Вильямс, 2006. — 1328 с. — ISBN 978-5-8459-0788-8.
3. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг ; пер. с англ. — 3-е изд. — Москва : Вильямс, 2017. — 1440 с. — ISBN 978-5-8459-2020-1.
4. Рогов, Е. В. PostgreSQL 16 изнутри / Е. В. Рогов. — Москва: ДМК Пресс, 2024. — 664 с. — ISBN 978-5-93700-305-8.
5. Хомоненко, А. Д. Базы данных: учебник для высших учебных заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев ; под ред. А. Д. Хомоненко. — 6-е изд., доп. — Санкт-Петербург: КОРОНА-Век, 2009. — 736 с. — ISBN 978-5-7931-0527-9.
6. Database System Concepts [Электронный ресурс] / A. Silberschatz, H. F. Korth, S. Sudarshan. — 7th ed. — Режим доступа: https://www.db-book.com/ (дата обращения: 04.07.2026).
7. Документация PostgreSQL 16 [Электронный ресурс] / PostgreSQL Global Development Group. — Режим доступа: https://www.postgresql.org/docs/16/ (дата обращения: 04.07.2026).
8. Документация MySQL 8.4 [Электронный ресурс] / Oracle Corporation. — Режим доступа: https://dev.mysql.com/doc/refman/8.4/en/ (дата обращения: 04.07.2026).
9. Документация SQLite [Электронный ресурс] / SQLite Development Team. — Режим доступа: https://www.sqlite.org/docs.html (дата обращения: 04.07.2026).
10. Документация FastAPI [Электронный ресурс]. — Режим доступа: https://fastapi.tiangolo.com/ (дата обращения: 04.07.2026).

11. Документация SQLAlchemy 2.0 [Электронный ресурс]. — Режим доступа: https://docs.sqlalchemy.org/en/20/ (дата обращения: 04.07.2026).
12. Клеппман, М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / М. Клеппман; пер. с англ. — Санкт-Петербург: Питер, 2018. — 640 с. — ISBN 978-5-4461-0512-0.

References:

1. Gagarina, L. G. Software Development Technology: a textbook / L. G. Gagarina, E. V. Kokoreva, B. D. Sidorova-Visnadul; edited by L. G. Gagarina. - Moscow: Publishing House "Forum": INFRA-M, 2018. - 400 p. - ISBN 978-5-8199-0812-9.
2. Date, K. J. Introduction to Database Systems / K. J. Date; translated from English. - 8th ed. - Moscow: Williams, 2006. - 1328 p. - ISBN 978-5-8459-0788-8.
3. Connolly, T. Databases. Design, Implementation, and Maintenance. Theory and Practice / T. Connolly, K. Begg; translated from English. — 3rd ed. — Moscow: Williams, 2017. — 1440 p. — ISBN 978-5-8459-2020-1.
4. Rogov, E. V. PostgreSQL 16 from the Inside / E. V. Rogov. — Moscow: DMK Press, 2024. — 664 p. — ISBN 978-5-93700-305-8.
5. Khomenko, A. D. Databases: a textbook for higher educational institutions / A. D. Khomenko, V. M. Tsygankov, M. G. Maltsev; edited by A. D. Khomenko. — 6th ed., supplemented. — St. Petersburg: KORONA-Vek, 2009. — 736 p. — ISBN 978-5-7931-0527-9.
6. Database System Concepts [Electronic resource] / A. Silberschatz, H. F. Korth, S. Sudarshan. — 7th ed. — Access mode: https://www.db-book.com/ (accessed date: 04.07.2026).
7. PostgreSQL 16 Documentation [Electronic resource] / PostgreSQL Global Development Group. — Access mode: https://www.postgresql.org/docs/16/ (accessed date: 04.07.2026).
8. MySQL 8.4 Documentation [Electronic resource] / Oracle Corporation. — Access mode: https://dev.mysql.com/doc/refman/8.4/en/ (accessed date: 04.07.2026).
9. SQLite Documentation [Electronic resource] / SQLite Development Team. — Access mode: https://www.sqlite.org/docs.html (accessed: 04.07.2026).
10. FastAPI Documentation [Electronic resource]. — Access mode: https://fastapi.tiangolo.com/ (accessed: 04.07.2026).
11. SQLAlchemy 2.0 Documentation [Electronic resource]. — Access mode: https://docs.sqlalchemy.org/en/20/ (accessed: 04.07.2026).
12. Kleppman, M. Highly Loaded Applications. Programming, Scaling, Support / M. Kleppman; trans. from English. — St. Petersburg: Piter, 2018. — 640 p. — ISBN 978-5-4461-0512-0.