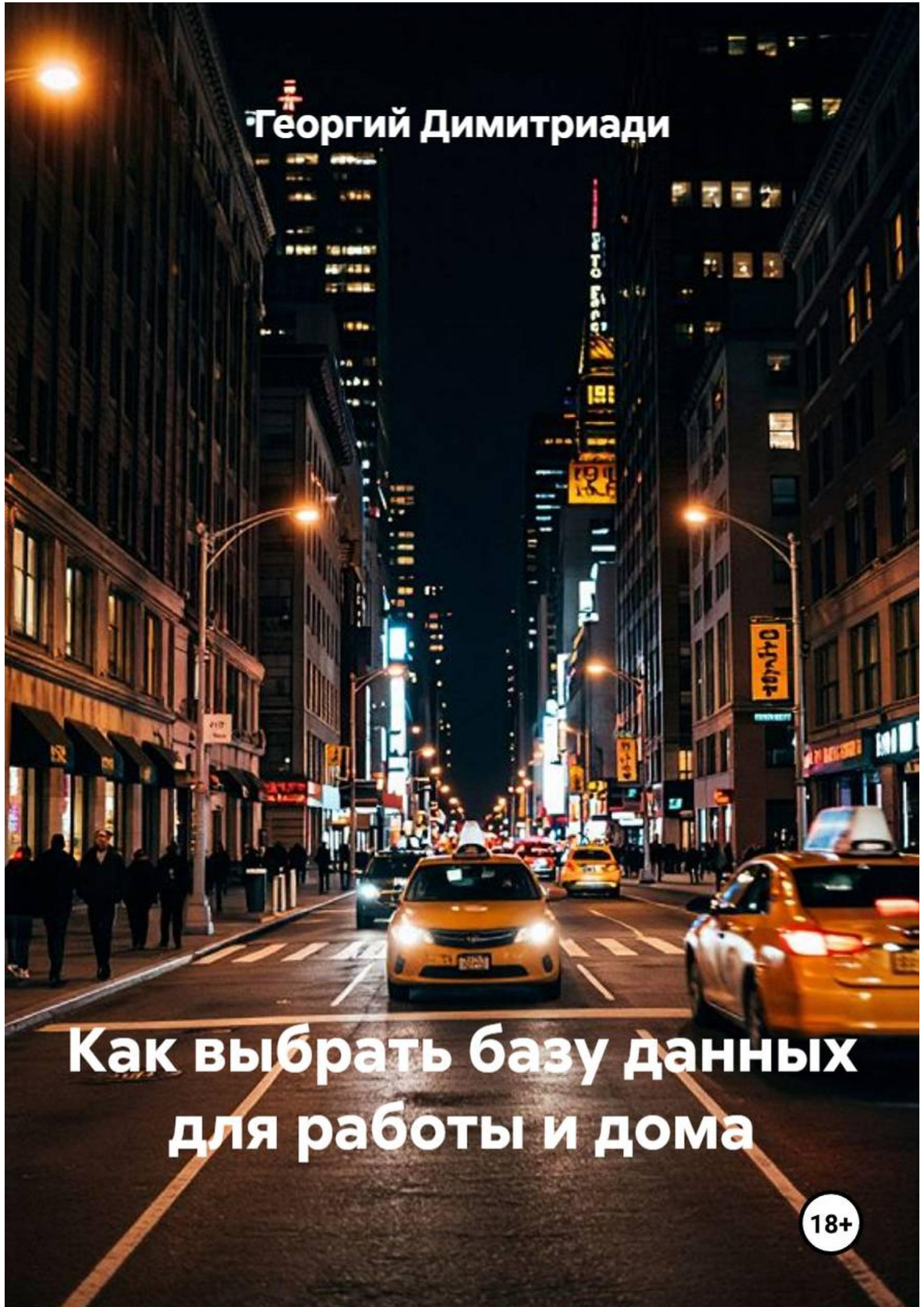




Георгий Димитриади

Как выбрать базу данных
для работы и дома

18+

Георгий Димитриади

**Как выбрать базу данных
для работы и дома**

«Автор»

2026

Димитриади Г.

Как выбрать базу данных для работы и дома / Г. Димитриади —
«Автор», 2026

Практическое руководство для ИТ-архитекторов, инженеров и продвинутых пользователей по выбору технологий хранения данных. Книга систематизирует фундаментальные принципы архитектуры систем управления базами данных (СУБД), классифицирует современные SQL/NoSQL решения (включая векторные БД для искусственного интеллекта) и формализует компромиссы CAP/PACELC. Выбор базы данных для корпоративных и домашних задач изложен в виде алгоритма. Также приведена методика расчета совокупной стоимости владения (ТСО). Заметное место отведено экосистеме PostgreSQL. Издание помогает избежать архитектурных ошибок, переводя выбор СУБД из области интуиции в плоскость инженерного расчета.

© Димитриади Г., 2026

© Автор, 2026

Содержание

Введение	5
Глава 1. Фундаментальные принципы архитектуры хранения данных	7
Глава 2. Технологии хранения данных на логическом уровне	10
Конец ознакомительного фрагмента.	12

Георгий Димитриади

Как выбрать базу данных для работы и дома

Введение

Актуальность проблемы выбора В современной информационной системе данные перестали быть побочным продуктом деятельности, превратившись в ее первостепенный актив. Эффективность бизнес-процессов, скорость принятия решений, качество работы моделей машинного обучения и отказоустойчивость критически важных сервисов напрямую зависят от архитектуры подсистемы хранения. На протяжении десятилетий реляционная модель данных (SQL) являлась безальтернативным стандартом для корпоративных приложений. Однако экспоненциальный рост объемов информации, переход к микросервисным архитектурам и появление новых классов задач, таких как обработка графов, временных рядов и векторных эмбедингов для генеративного искусственного интеллекта, привели к необратимой диверсификации технологического ландшафта.

Для ИТ-специалиста, системного архитектора или DevOps-инженера текущий рынок технологий хранения представляет собой сложную экосистему, насчитывающую сотни решений. Выбор между транзакционной надежностью реляционных СУБД, горизонтальной масштабируемостью NoSQL-платформ, субмиллисекундным доступом key-value хранилищ и аналитической мощностью колоночных баз данных перестал быть тривиальной задачей. Ошибка на этапе проектирования архитектуры данных влечет за собой не только прямые финансовые издержки на миграцию, но и стратегические риски, связанные с невозможностью реализации требуемого функционала или деградацией производительности системы под нагрузкой.

Цель и задачи книги Целью данного издания является формирование системной методологии выбора технологии хранения данных. Книга призвана перевести процесс принятия архитектурных решений из плоскости субъективных предпочтений и маркетинговых обещаний в область инженерного расчета, основанного на анализе формальных требований, фундаментальных принципах компьютерных систем и экономике владения.

Для достижения поставленной цели в тексте решаются следующие задачи:

Систематизация фундаментальных принципов архитектуры хранения данных, включая иерархию памяти, модели конкурентного доступа и уровни изоляции транзакций.

Детальная классификация современных технологий хранения на логическом уровне с акцентом на их внутреннюю структуру, алгоритмы индексации и модели согласованности.

Анализ специфики специализированных баз данных для задач искусственного интеллекта, в частности векторных и time-series СУБД.

Разработка и описание формального алгоритма выбора технологии, применимого как для высоконагруженных корпоративных систем, так и для автономных решений индивидуального пользования.

Исследование компромиссов проектирования (CAP, PACELC) и их влияния на нефункциональные характеристики системы.

Расчет совокупной стоимости владения (Total Cost of Ownership, TCO) с учетом скрытых операционных расходов.

Детальное рассмотрение экосистемы PostgreSQL как наиболее универсальной и зрелой open source платформы, а также алгоритмов обеспечения катастрофоустойчивости.

Целевая аудитория и специфика изложения Данная книга рассчитана на специалистов в области информационных технологий: системных архитекторов, инженеров по эксплуатации (SRE/DevOps), администраторов баз данных (DBA) и ведущих разработчиков программного обеспечения. Предполагается, что читатель обладает базовыми знаниями в области алгоритмизации, понимает принципы работы операционных систем и компьютерных сетей.

Несмотря на техническую глубину материала, изложение строится с соблюдением принципа формальной строгости. Автор избегает метафор, бытовых аналогий и маркетинговой терминологии, свойственной современной ИТ-публицистике. Каждый термин - от «согласованности» до «векторного индекса HNSW» - вводится через его формальное определение, математическую модель или описание конкретного алгоритма. Такой подход обеспечивает однозначность трактовок и позволяет использовать данную книгу в качестве справочного руководства при проектировании реальных систем.

Структура книги Материал выстроен по принципу «от общего к частному». Первая часть (Главы 1–2) закладывает теоретический фундамент. Вторая часть (Главы 3–4) посвящена анализу инструментария и архитектурных компромиссов. Третья часть (Главы 5–8) фокусируется на прикладных аспектах: алгоритме выбора, экономическом обосновании, обеспечении надежности и детальном разборе эталонной СУБД. В заключении синтезируются выводы, формирующие у читателя целостную картину ландшафта технологий хранения данных и критерии их применимости в условиях заданных ограничений.

Глава 1. Фундаментальные принципы архитектуры хранения данных

1.1. Иерархия памяти и физика хранения Эффективность любой системы управления базами данных (СУБД) ограничена физическими характеристиками аппаратного обеспечения. Архитектура хранения данных строится на иерархии памяти, где каждый последующий уровень характеризуется увеличением емкости при одновременном росте задержек доступа (latency) и снижении пропускной способности (throughput).

С точки зрения СУБД, иерархия представляется следующим образом:

Регистры процессора и кэш-память (L1, L2, L3): Доступ осуществляется за 1–40 тактов процессора. СУБД не управляет этим уровнем напрямую, однако эффективность работы планировщика запросов и алгоритмов индексации критически зависит от локальности данных в памяти (cache locality).

Оперативная память (RAM, DRAM): Основной рабочий слой для современных СУБД. Задержки составляют сотни наносекунд. Объемы позволяют размещать индексы, словари и горячие наборы данных целиком в памяти. Ключевая особенность – энергозависимость. При отключении питания данные в DRAM утрачиваются.

Энергонезависимая память (NVM/Optane): Промежуточный слой, сочетающий скорость, близкую к DRAM, и персистентность (сохранность при отключении питания). В современных архитектурах используется для размещения журналов транзакций с целью минимизации задержки подтверждения записи (commit latency).

Твердотельные накопители (SSD/NVMe): Основной слой для персистентного хранения. Задержки доступа составляют десятки микросекунд. Использование NVMe-интерфейсов и протокола PCIe устраняет узкое место традиционных SAS/SATA SSD. Для СУБД это означает изменение профиля нагрузки: время поиска (seek time) стремится к нулю, а основным ограничителем становится параллелизм контроллера и пропускная способность шины.

Жесткие диски (HDD) и магнитные ленты (LTO): Слои холодного хранения. HDD характеризуются высокими механическими задержками (миллисекунды) и используются для архивных копий или хранения логов. Магнитные ленты обеспечивают минимальную стоимость терабайта при максимальных задержках (минуты на установку картриджа), применяясь исключительно для холодного резервирования (Disaster Recovery).

1.2. Модель вычислений фон Неймана и разделение вычислений с хранением Классическая архитектура СУБД базируется на принципах машины фон Неймана, где данные и инструкции хранятся в едином адресном пространстве. Однако в распределенных системах хранения реализуется отход от этой модели в сторону разделения ресурсов (Shared-Nothing Architecture).

В такой парадигме узел СУБД разделяется на два логических компонента:

Compute Layer (Слой вычислений): Выполняет парсинг SQL-запросов, построение плана выполнения, агрегации и соединения (JOIN). Является stateless (без сохранения состояния) и может горизонтально масштабироваться.

Storage Layer (Слой хранения): Отвечает за физическую запись байтов на носитель, управление экстендами, блокировками и журналами.

Разделение позволяет независимо масштабировать вычислительную мощность и объем дискового пространства. Протокол взаимодействия между слоями (например, набор процедур удаленного вызова RPC) определяет накладные расходы системы.

1.3. Принципы организации данных на логическом уровне СУБД абстрагирует физическую запись байтов, представляя пользователю структурированные отношения. Эта абстракция реализуется через три фундаментальных механизма.

1.3.1. Сериализация и десериализация (Marshaling) Преобразование сложных структур данных (строк, массивов, JSON-объектов) в линейный поток байтов для записи на диск.

Фиксированная длина: Целые числа, числа с плавающей запятой. Позволяют осуществлять произвольный доступ ($O(1)$), так как смещение любой записи вычисляется арифметически: $\text{Offset} = \text{Base} + (\text{Index} * \text{RecordSize})$.

Переменная длина (Variable Length): Текстовые строки (VARCHAR), JSON. Требуют использования заголовков с указателями (pointers) и каталогов смещения (offset directories) внутри страницы данных. СУБД PostgreSQL использует для этого формат *TOAST* (The Oversized-Attribute Storage Technique), вынося данные объемом более 2 КБ в отдельные таблицы.

1.3.2. Индексация Вторичная структура данных, целью которой является снижение алгоритмической сложности поиска. Без индекса поиск значения в неупорядоченной таблице требует последовательного сканирования со сложностью $O(n)$.

В-деревья (B-trees): Стандарт де-факто для большинства СУБД. Сбалансированное дерево, где все листья находятся на одной глубине. Обеспечивают поиск, вставку и удаление за $O(\log n)$. Эффективны для точечных запросов и поиска по диапазону (range queries).

Хеш-индексы: Обеспечивают поиск за $O(1)$, но не поддерживают поиск по диапазону и чувствительны к коллизиям.

GiST (Generalized Search Tree) и SP-GiST: Обобщенные древовидные структуры, используемые в PostgreSQL для реализации специализированных индексов (например, R-деревьев для геоанных через расширение PostGIS).

GIN (Generalized Inverted Index): Инвертированный индекс. Применяется для ускорения поиска по элементам внутри составных типов (массивы, JSONB, полнотекстовый поиск).

1.3.3. Управление транзакциями и журналирование Гарантия атомарности (Atomicity) и долговечности (Durability) из набора свойств ACID обеспечивается через протокол журнализации.

Write-Ahead Logging (WAL): Фундаментальный принцип, согласно которому любое изменение данных (вставка, обновление, удаление) должно быть сначала зафиксировано в журнале (WAL-segment) на стабильном носителе, и только после успешного сброса (fsync) журнала на диск СУБД применяет изменения к файлам данных.

Контрольные точки (Checkpoints): Процесс синхронизации грязных страниц (dirty pages) из оперативной памяти с диском. После прохождения контрольной точки СУБД может безопасно удалить старые сегменты WAL, так как все изменения до этой точки уже персистентны в основных файлах данных. Частота контрольных точек определяет баланс между нагрузкой на дисковую подсистему (I/O spikes) и временем восстановления после сбоя (Recovery Time Objective, RTO).

1.4. Уровни изоляции транзакций В многопользовательской среде СУБД должна разрешать конфликты доступа. Стандарт SQL определяет четыре уровня изоляции, реализуемых через механизм многоверсионного контроля (MVCC) или блокировок:

Read Uncommitted (Чтение незафиксированных данных): Допускает «грязное чтение». Транзакция видит изменения, которые еще не подтверждены (COMMIT) другими транзакциями. Практически не используется.

Read Committed (Чтение зафиксированных данных): Каждый запрос внутри транзакции видит только подтвержденные на момент его начала данные. Решает проблему грязного чтения, но допускает неповторяющееся чтение (non-repeatable read) и фантомы.

Repeatable Read (Повторяющееся чтение): Транзакция видит снимок данных (snapshot), зафиксированный на момент ее старта. Гарантирует, что повторное выполнение одного и того же запроса вернет идентичный результат. Решает проблему неповторяющегося чтения, но в классической реализации допускает фантомные строки (PostgreSQL решает это на уровне Serializable).

Serializable (Сериализуемый): Высший уровень изоляции. Результат выполнения параллельных транзакций идентичен результату их последовательного выполнения. Реализуется через предикатные блокировки или отслеживание конфликтов сериализации (Serialization Anomalies). Влечет максимальные накладные расходы и риск взаимоблокировок (deadlocks).

Выбор уровня изоляции является первым архитектурным компромиссом между целостностью данных и производительностью системы, определяющим вероятность возникновения аномалий при конкурентном доступе.

Глава 2. Технологии хранения данных на логическом уровне

Логический уровень хранения данных определяет способ организации, структурирования и доступа к информации, абстрагируясь от физических механизмов записи на магнитные или полупроводниковые носители. Выбор технологии на данном уровне диктует алгоритмическую сложность операций, возможности интеграции и предельную масштабируемость системы.

2.1. Немоделируемые (пассивные) хранилища К данной категории относятся форматы, которые представляют собой последовательность байтов без встроенного механизма индексации, управления транзакциями или обеспечения целостности на уровне формата. Ответственность за интерпретацию данных полностью возлагается на прикладное программное обеспечение.

2.1.1. Плоские текстовые файлы (TXT, CSV, TSV) *Структура:* Последовательность строк, разделенных символом конца строки (LF/CRLF). Поля внутри строки разделяются ограничителем (delimiter), чаще всего запятой (CSV - Comma-Separated Values) или табуляцией (TSV). *Достоинства:* Кроссплатформенность, человекочитаемость, отсутствие необходимости в специализированном серверном ПО. Формат является стандартом де-факто для обмена данными между гетерогенными системами. *Недостатки:* Алгоритмическая сложность поиска составляет $O(n)$, где n - количество строк. Отсутствие типизации данных (все значения интерпретируются как строки) требует парсинга на стороне приложения. Проблемы экранирования спецсимволов и разделителей внутри полей. Низкая производительность при потоковой записи из-за необходимости частого перемещения указателя файла. *Open source инструменты:* Библиотеки парсинга (Python csv, pandas), утилиты командной строки (awk, sed).

2.1.2. Файлы электронных таблиц (XLSX, ODS) *Структура:* ZIP-архив, содержащий набор XML-документов, описывающих листы, стили, формулы и бинарные объекты. Формат ODS (OpenDocument Spreadsheet) является открытым стандартом ISO/IEC 26300. *Достоинства:* Поддержка многолистовой структуры, встроенных формул, метаданных ячеек и визуального форматирования. Позволяет конечным пользователям взаимодействовать с данными без навыков программирования. *Недостатки:* Жесткое ограничение на количество строк (1 048 576 в спецификации XLSX). Блокировка файла на уровне операционной системы при открытии в графическом редакторе, что исключает многопользовательский доступ. Высокие накладные расходы на парсинг XML-структуры при программном доступе. Неэффективность хранения больших массивов чисел. *Open source инструменты:* Библиотеки openpyxl, pandas (для Python), LibreOffice SDK.

2.1.3. Форматы сериализации и обмена данными (JSON, XML, YAML, Avro, Protocol Buffers) *Структура:* Иерархическое представление объектов. JSON и YAML используют синтаксис «ключ-значение» и отступы. XML - теговую структуру. Avro и Protocol Buffers (Protobuf) - бинарное представление со строгой схемой. *Достоинства:* Семантическая нагруженность (данные самодокументируемы). Широкая поддержка во всех языках программирования. JSON и XML не требуют предварительной компиляции схемы. *Недостатки:*

JSON/XML: Избыточность синтаксиса (теги, кавычки), что увеличивает объем хранимых данных на 40–200% по сравнению с бинарными форматами. Отсутствие строгой схемы (в базовом виде) ведет к ошибкам типизации.

Protobuf/Avro: Требуют наличия скомпилированного описания схемы (.proto или .avsc). Не являются человекочитаемыми без применения декодеров. *Open source инструменты:* Библиотеки libxml2, jansson, protobuf-c, avro-cpp.

2.2. Моделируемые (активные) хранилища Данные системы включают в себя программный слой (движок), который предоставляет API для манипуляции данными, обеспечивая их целостность, индексацию и управление конкурентным доступом.

2.2.1. Иерархические базы данных *Модель:* Древовидная структура. Каждая запись имеет одного «родителя» и может иметь множество «потомков». Доступ к данным осуществляется через навигацию по ветвям дерева (от корня к листьям). *Пример:* Файловые системы (NTFS, ext4), ранние версии СУБД IBM IMS. *Назначение:* Хранение данных с жесткой, редко меняющейся вложенностью (например, структура каталогов, спецификации изделий ВОМ). *Достоинства:* Высокая скорость извлечения целых поддеревьев за один проход. *Недостатки:* Неэффективность при реализации связей «многие-ко-многим». Сложность изменения структуры дерева (требует физического перемещения больших блоков данных).

2.2.2. Сетевые базы данных *Модель:* Граф произвольной структуры, расширяющий иерархическую модель. Запись может иметь несколько «родителей». Связи между записями реализуются через указатели (адреса на диске). *Пример:* IDMS (Integrated Data Store), CA Datacom. *Назначение:*

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.