DOI: 10.18137/RNU.V9187.26.01.P.170 УДК 004.624

**Иевлев Кирилл Олегович**
аспирант, ассистент кафедры интеллектуального анализа данных, Московский технический университет связи и информатики, Москва.
SPIN-код: 1380-5720, AuthorID: 1226996
Электронный адрес: ievlev.k.o@yandex.ru
**Kirill O. Ievlev**
Postgraduate, Lecturer Assistant at the Department of data mining, Moscow Technical University of Communications and Informatics, Moscow.
SPIN-code: 1380-5720, AuthorID: 1226996
E-mail address: ievlev.k.o@yandex.ru

**Городничев Михаил Геннадьевич**
кандидат технических наук, доцент, декан факультета информационных технологий, заведующий кафедрой математической кибернетики и информационных технологий, Московский технический университет связи и информатики, Москва.
SPIN-код: 4576-9642, AuthorID: 893531
Электронный адрес: m.g.gorodnichev@mtuci.ru
**Mikhail G. Gorodnichev**
Ph.D. of Technical Sciences, Docent, Dean of information technologies faculty, Head of the Department of mathematical cybernetics and information technologies, Technical University of Communications and Informatics, Moscow.
SPIN-code: 4576-9642, AuthorID: 893531
E-mail address: m.g.gorodnichev@mtuci.ru

# СРАВНИТЕЛЬНЫЙ АНАЛИЗ ОТКРЫТЫХ ТАБЛИЧНЫХ ФОРМАТОВ: APACHE ICEBERG, APACHE HUDI, DELTA LAKE, APACHE PAIMON

**Аннотация.** В статье исследуются открытые табличные форматы (Open Table Formats), формирующие технологическую основу хранилищ данных Data Lakehouse. Рассмотрена эволюция подходов к хранению данных – от классических хранилищ (Data Warehouse) через озёра данных (Data Lake) к гибридной модели Data Lakehouse. Проанализированы архитектурные решения и функциональные особенности Apache Iceberg, Apache Hudi, Delta Lake и Apache Paimon. Представлен сравнительный анализ форматов и рекомендации по их применению в зависимости от характера нагрузки.

***Ключевые слова:*** открытые табличные форматы, Apache Iceberg, Apache Hudi, Delta Lake, Apache Paimon, Data Lakehouse, ACID, эволюция схемы.

***Для цитирования:*** *Иевлев К.О., Городничев М.Г.* Сравнительный анализ открытых табличных форматов: Apache Iceberg, Apache Hudi, Delta Lake, Apache Paimon // Вестник Российского нового университета. Серия: Сложные системы: модели, анализ, управление. 2026. № 1. С. 170–188. DOI: 10.18137/RNU.V9187.26.01.P.170

# COMPARATIVE ANALYSIS OF MODERN OPEN TABLE FORMATS: APACHE ICEBERG, APACHE HUDI, DELTA LAKE, APACHE PAIMON

**Abstract.** The article examines open table formats that form the technological foundation of Data Lakehouse architecture. The evolution of data storage approaches from classical Data Warehouses through Data Lakes to the hybrid Data Lakehouse model is reviewed. Architectural solutions and functional features of Apache Iceberg, Apache Hudi, Delta Lake and Apache Paimon are analyzed. A comparative analysis of formats and recommendations for their application depending on workload characteristics are presented.

***Keywords:*** Open Table Formats, Apache Iceberg, Apache Hudi, Delta Lake, Apache Paimon, Data Lakehouse, ACID, schema evolution.

***For citation:*** Ievlev K.O., Gorodnichev M.G. (2026) Comparative analysis of modern open table Formats: Apache Iceberg, Apache Hudi, Delta Lake, Apache Paimon. *Vestnik of Russian New University. Series: Complex Systems: Models, analysis, management.* No. 1. Pp. 170–188. DOI: 10.18137/RNU.V9187.26.01.P.170 (In Russian).

## *Введение*

Рост объёмов и разнообразия данных в цифровом мире привёл к тому, что классические хранилища данных и «чистые» озёра данных по отдельности перестали удовлетворять требованиям современных аналитических и потоковых нагрузок: первым часто не хватает гибкости и экономичности, вторым – согласованности и управляемости. Архитектура Data Lakehouse рассматривается как компромисс, объединяющий преимущества обоих подходов за счёт появления транзакционного табличного слоя поверх объектного/файлового хранилища.

Ключевым механизмом реализации Lakehouse стали *открытые табличные форматы* (Open Table Formats, OTF), обеспечивающие согласованные снапшоты данных, управляемую эволюцию схемы и поддержку операций обновления/удаления.

*Цель* статьи – сравнить Apache Iceberg, Apache Hudi, Delta Lake и Apache Paimon с точки зрения модели метаданных и транзакций, стоимости строковых обновлений и требований к эксплуатации, а также сформулировать практические критерии выбора формата под разные профили нагрузок.

## *Эволюция архитектур хранилищ данных*

Концепция Data Warehouse (DWH) получила распространение в начале 1990-х годов как ответ на ограничения транзакционных OLTP-систем, а именно невозможность консолидации данных из разрозненных источников и отсутствие истории изменений [1]. Классические DWH представляют собой централизованные аналитические системы с поддержкой ACID-транзакций, часто использующие архитектуру массированно-параллельной обработки (MPP) для горизонтального масштабирования [2; 3]. Однако исторически большинство таких решений являлись проприетарными продуктами крупных поставщиков (Teradata, Oracle, IBM) с высокой стоимостью владения и лицензирования. Жёсткая схема данных (schema-on-write) и сложность масштабирования, а также неспособность эффективно обрабатывать полуструктурированные и неструктурированные данные (логи, JSON, медиа, телеметрию IoT), стали критическими огра-

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

ничениями на фоне экспоненциального роста объёмов и разнообразия информации в XXI веке [4; 5].

Ответом на ограничения классических DWH стала концепция Data Lake (озеро данных), сформировавшаяся в начале 2010-х годов на базе распределённых файловых систем (HDFS) и облачных объектных хранилищ (S3, ADLS, GCS) [2; 4]. Озёра данных позволяют сохранять разнородную информацию в исходном виде без предварительной трансформации, реализуя подход «схема при чтении» (schema-on-read) [6; 7]. Это обеспечивает экономичное хранение больших объёмов данных (вплоть до петабайт) и гибкость в выборе инструментов обработки – от SQL-аналитики и полнотекстового поиска до задач машинного обучения. Ключевую роль в эволюции концепции сыграл проект Apache Hadoop, а также появление открытых колоночных форматов (Apache Parquet, ORC). Последние обеспечили высокую эффективность сжатия и чтения данных, что позволило интегрировать файловые хранилища с современными аналитическими фреймворками [2; 7].

Вместе с тем сложность версионирования и управления метаданными в Data Lake породила новые вызовы. Базовые файловые и объектные хранилища не гарантируют атомарность и согласованность при конкурентных операциях чтения/записи, что критически усложняет процессы обновления и удаления данных [2]. При отсутствии строгого контроля качества, каталогизации и управления доступом Data Lake неизбежно деградирует в «болото» данных (data swamp), где затруднительно найти необходимую информацию, отследить её происхождение (data lineage) и гарантировать её достоверность [7].

Первая попытка упростить аналитическую работу с Data Lake была предпринята в рамках проекта Apache Hive. Данный инструмент убрал необходимость написания сложных MapReduce-заданий, предложив SQL-интерфейс, транслирующий декларативные запросы в распределённые вычисления и концепцию метаданных: таблица определялась как набор файлов в директории, а партиции – как поддиректории. Централизованный каталог (Hive Metastore) хранил схему и пути к данным, предоставляя движкам единую точку входа [8].

Однако с ростом объёмов данных в 2010-х (мобильные устройства, IoT, переход на облачные S3-хранилища) и появлением требований к интерактивной аналитике в реальном времени ограничения Hive-подхода стали критичными [2]:

- зависимость от файловой структуры. Прямая привязка «таблица = директория» делала операции изменения данных дорогими (требовалась перезапись файлов), а эволюцию партиционирования – практически невозможной;
- медленное планирование запросов. Для больших таблиц с тысячами партиций и файлов операция листинга директорий (особенно в S3) и построение плана запроса занимали значительное время [9];
- отсутствие полноценных ACID-транзакций. Попытки добавить транзакционность в Hive привели к сложным схемам, требующим постоянного обслуживания и привязанным к конкретным движкам и форматам (ORC) [10];
- отсутствие изоляции версий. Невозможность корректно реализовать Time Travel или атомарный откат изменений [2; 10].

Для преодоления данных ограничений был предложен новый уровень абстракции – открытые табличные форматы (Open Table Formats, OTF) – слой управления данными поверх распределённой файловой системы или объектного хранилища, превращающий набор файлов в транзакционную таблицу с ACID-гарантиями, версионированием и под-

держкой операций на уровне строк [2; 4]. В отличие от Hive, где таблица жёстко привязана к файловой иерархии, OTF вводят явный слой метаданных, отделяющий логическую структуру таблицы от физического расположения данных [9].

В обобщённом виде OTF представляют собой трёхуровневую систему [10]:

1) *файловый слой* – данные в колоночных форматах Parquet/ORC, оптимизированных для аналитических запросов;

2) *слой метаданных* – списки файлов, статистика (min/max значения для колонок), структура партиций, снимки состояния таблиц, индексы и информацию об удалённых записях;

3) *логический слой* (API) – механизмы транзакций, атомарных коммитов, изменения схемы таблиц и контроля доступа.

### *1. Основные механизмы и терминология Open Table Formats*

Для корректного сравнения табличных форматов (Iceberg, Hudi, Delta Lake, Paimon) необходимо определить общие архитектурные примитивы и унифицировать терминологию, поскольку все рассматриваемые решения, несмотря на различия в реализации, используют схожие концептуальные подходы [11; 12].

***1.1. Модель метаданных и транзакционность.*** Фундаментом любого OTF является понятие снапшота (Snapshot) – неизменяемого (immutable) состояния таблицы в конкретный момент времени, включающего полный список активных файлов данных. Переход между снапшотами осуществляется через атомарную операцию коммита (Commit) [13].

Для обеспечения конкурентного доступа используется оптимистическая блокировка (Optimistic Concurrency Control) и принцип MVCC (Multi-Version Concurrency Control): читатели всегда работают с зафиксированным снапшотом, не подвергаясь блокировкам со стороны операций записи. Это позволяет реализовать функции Time Travel (запрос к исторической версии данных) и Rollback (откат к предыдущему состоянию) [14].

Фиксация последовательности изменений реализуется архитектурно по-разному – через иерархию манифестов (файлы со списками партиций и объектов данных, как в Iceberg или Paimon [10; 15]) или через журнал транзакций (Transaction Log) (последовательный список действий, как в Delta Lake или Hudi [16; 17]). Для предотвращения бесконечного роста истории метаданных периодически создаются контрольные точки (Checkpoints), агрегирующие актуальное состояние [15].

***1.2. Управление структурой данных.*** Ключевым преимуществом OTF является разрыв жесткой связи между логической схемой и физическим хранением:

- эволюция схемы (Schema Evolution) – возможность изменять состав колонок без перезаписи данных; продвинутые реализации используют внутренние ID-колонки, что позволяет безопасно переименовывать или менять порядок полей [10];
- скрытое партиционирование (Hidden Partitioning) – концепция, при которой пользователь оперирует бизнес-полями (например, timestamp), а формат автоматически преобразует их в физические партиции (например, hour (timestamp)), это избавляет аналитиков от необходимости знать физическую структуру папок и обеспечивает корректное отсечение партиций (pruning) планировщиком запросов [10];
- эволюция партиционирования (Partition Evolution) – возможность изменения схемы разбиения данных для новых записей без необходимости миграции старых данных [10].

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

***1.3. Стратегии записи и обслуживания.*** Поддержка операций обновления и удаления (UPDATE/DELETE/MERGE) на уровне строк реализуется через две основные стратегии [14]:

• Copy-on-Write (CoW) – при обновлении строки формат переписывает весь файл целиком. Это обеспечивает максимальную скорость чтения, но увеличивает время записи, идеально подходит для сценариев с редкими обновлениями и частым чтением [10; 15];

• Merge-on-Read (MoR) – изменения записываются в отдельные дельта-файлы (логи изменений), при чтении движок «на лету» объединяет базовые файлы с дельтами, это ускоряет запись, но замедляет чтение [10; 15].

Интенсивная работа с таблицей (особенно в режиме MoR) требует регулярных процедур обслуживания:

• compaction (уплотнение) – фоновый процесс слияния мелких файлов и дельт в более крупные базовые файлы [15];

• Vacuum / Expiration – физическое удаление устаревших снапшотов и файлов данных, более не требуемых для Time Travel, для освобождения места и соблюдения политик хранения [10; 17].

## *2. Apache Iceberg*

Наиболее заметным представителем OTF на сегодняшний день является Apache Iceberg – формат, разработанный в Netflix для преодоления ограничений Hive при работе с петабайтными таблицами [10]. В отличие от классического подхода, Iceberg полностью абстрагирует логическую таблицу от физического расположения файлов, становясь универсальным слоем хранения, с которым эффективно работают Spark, Flink, Trino и другие популярные движки распределенных вычислений [11; 13].

***2.1. Архитектура и модель метаданных Apache Iceberg.*** В основе архитектуры Iceberg лежит иерархическое дерево метаданных, которое заменяет собой листинг файловой системы. Состояние таблицы определяется через цепочку объектов [10]:

• Catalog (каталог) – хранит ссылку на текущий файл метаданных (metadatafile);

• Metadata File – содержит схему, настройки партиционирования и историю снапшотов;

• Manifest List– перечень файлов-манифестов для конкретного снапшота;

• Manifest File– детальный список файлов данных со статистикой (min/max значения, количество null).

Именно эта структура отличает Iceberg от конкурентов – планирование запроса происходит путем «спуска» по дереву метаданных. Движок отсекает ненужные ветки (файлы) на основе статистики (scanpruning) еще до обращения к хранилищу S3/HDFS. Это делает время планирования зависимым не от общего размера таблицы, а от объема выборки, что критично для высоконагруженных систем [12].

Транзакционная модель, описанная в разделе 3.1, здесь реализуется через строгую изоляцию снапшотов: любой коммит лишь добавляет новые файлы метаданных и переключает указатель в каталоге, что гарантирует атомарность и отсутствие блокировок при чтении [14]. Схема архитектуры хранения данных в Apache Iceberg представлена на Рисунке 1.

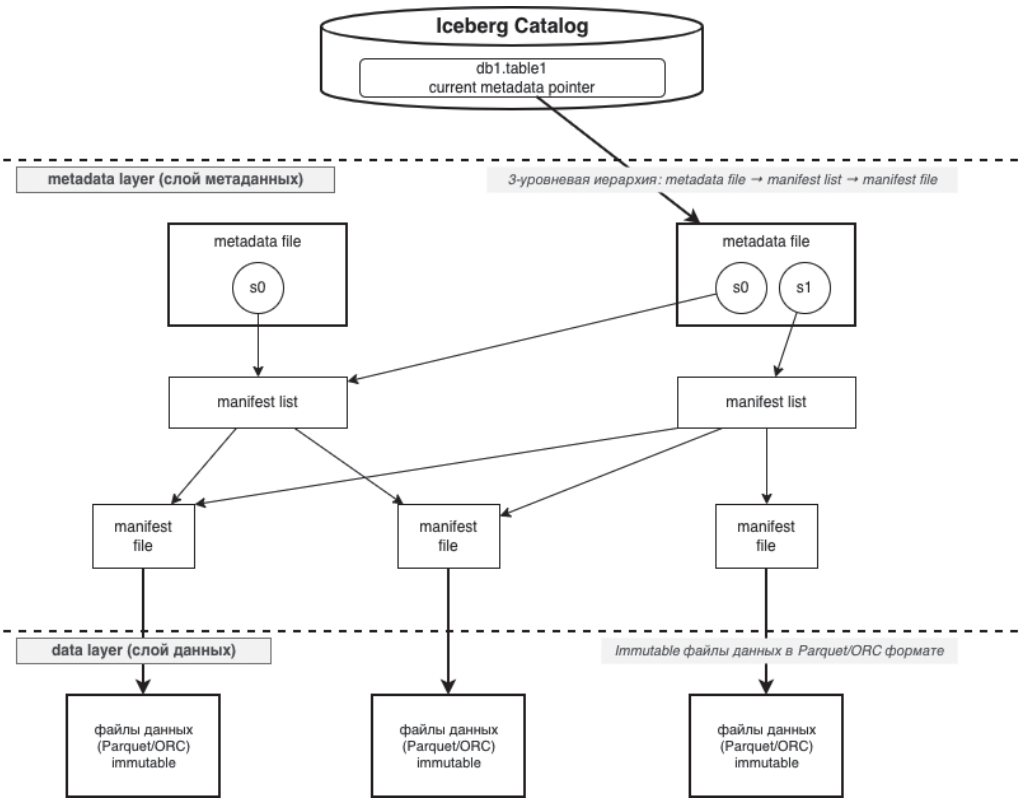


**Рисунок 1.** Схема архитектуры хранения данных в Apache Iceberg
*Источник:* здесь и далее рисунки выполнены авторами

***2.2. Реализация ключевых возможностей Apache Iceberg.*** Iceberg стал эталонной реализацией концепций эволюции схемы и партиционирования.

В части управления структурой формат использует уникальные внутренние идентификаторы (ID) для каждой колонки. Это позволяет безопасно переименовывать поля, менять их тип или порядок, данные при этом физически не перезаписываются, меняется лишь маппинг в метаданных. Аналогично реализовано и скрытое партиционирование: Iceberg хранит не просто пути к директориям, а функции трансформации (например, hour (timestamp)). Это дает возможность менять гранулярность разбиения (Partition Evolution) на лету: старые данные остаются в прежней структуре, новые пишутся по-новому, а движок прозрачно объединяет их в одном запросе [10].

В части обновления данных (DML) Iceberg предоставляет гибкость в выборе стратегии. Для пакетных нагрузок применяется Copy-on-Write, обеспечивающий быстрое чтение. Для потоковых сценариев (Change Data Capture, CDC) и частых обновлений используется Merge-on-Read, реализованный через Positional Delete Files – отдельные специальные файлы, применяемые при чтении и указывающие на удаленные строки в файлах данных. Такой подход считается более эффективным для аналитики, чем классические bloom-фильтры, хотя и требует ресурсов на слияние при чтении данных [14].

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

***2.3. Практическая применимость и ограничения Apache Iceberg.*** Анализ архитектуры [12; 13] позволяет выделить профиль применимости формата.

Сильные стороны Iceberg – это его архитектурная строгость и экосистема, в частности:

• производительность на больших масштабах – благодаря иерархии манифестов формат эффективно справляется с таблицами, содержащими миллионы файлов, – масштаб, на котором архитектура Hive Metastore становилась узким местом [10];

• независимость от движка (Engine Agnostic) – Iceberg предлагает, пожалуй, наиболее полную совместимость с экосистемой (Spark, Flink, Trino, Dremio), не отдавая явного приоритета ни одному из движков [11];

• зрелость экосистемы – активное сообщество и внедрение в продукты гигантов (AWS, Snowflake, Cloudera) гарантируют долгосрочную поддержку [13].

Основные вызовы при эксплуатации*:*

• накладные расходы (Overhead) – для небольших таблиц сложная структура метаданных (множество JSON/Avro файлов) может создавать избыточную нагрузку на хранилище и увеличивать латентность простых операций [12];

• сложность обслуживания – для поддержания производительности (особенно при использовании MoR) критически важны регулярные процедуры компакции (compaction) и очистки старых снапшотов (expiration). Без настроенной оркестрации этих процессов производительность чтения может деградировать [15].

## *3. Apache Hudi*

Apache Hudi (Hadoop Upsert Delta and Incremental) – формат, созданный в Uber для решения задач потокового приема данных (ingestion) и частых обновлений (upsert) с сохранением транзакционной целостности [16]. Если Iceberg изначально фокусировался на проблемах масштабируемости метаданных для больших аналитических таблиц, то архитектура Hudi оптимизирована под near-real-time-аналитику и CDC-конвейеры, где данные не просто дописываются, а постоянно изменяются [18].

***3.1. Архитектура и модель метаданных Apache Hudi.*** Центральным элементом Hudi является Timeline (журнал операций). В отличие от иерархии манифестов Iceberg, Hudi организует метаданные как последовательность событий (commit, delta_commit, compaction, clean) на временной шкале. Это позволяет читателям получать согласованный срез данных, соответствующий конкретной точке фиксации изменений, а также эффективно запрашивать историю обновлений.

В Hudi реализованы обе стратегии записи, описанные в разделе 3.3, однако они имеют свои *особенности реализации* [16]:

• в режиме Copy-on-Write (CoW) данные хранятся в версионированных Parquet-файлах;

• в режиме Merge-on-Read (MoR) используется гибридный подход – базовые файлы (Parquet) дополняются логами изменений в строковом формате Avro, запись производится методом append-only в Avro-логи, что минимизирует задержку, а чтение требует слияния базы и логов.

Отличительной чертой архитектуры Hudi является встроенная подсистема индексации. Индекс в Hudi – это механизм, который сопоставляет первичный ключ записи

(Record Key) с путем к файлу, где эта запись хранится (FilePath). Это позволяет при операции UPSERT мгновенно находить нужный файл для обновления, избегая полного сканирования партиции. Hudi поддерживает различные типы индексов, например, Bloom Index (вероятностная структура для отсечения ненужных файлов) и Bucket Index (детерминированное хеширование ключей по фиксированным бакетам) [16]. Схема архитектуры хранения данных в Apache Hudi представлена на Рисунке 2.

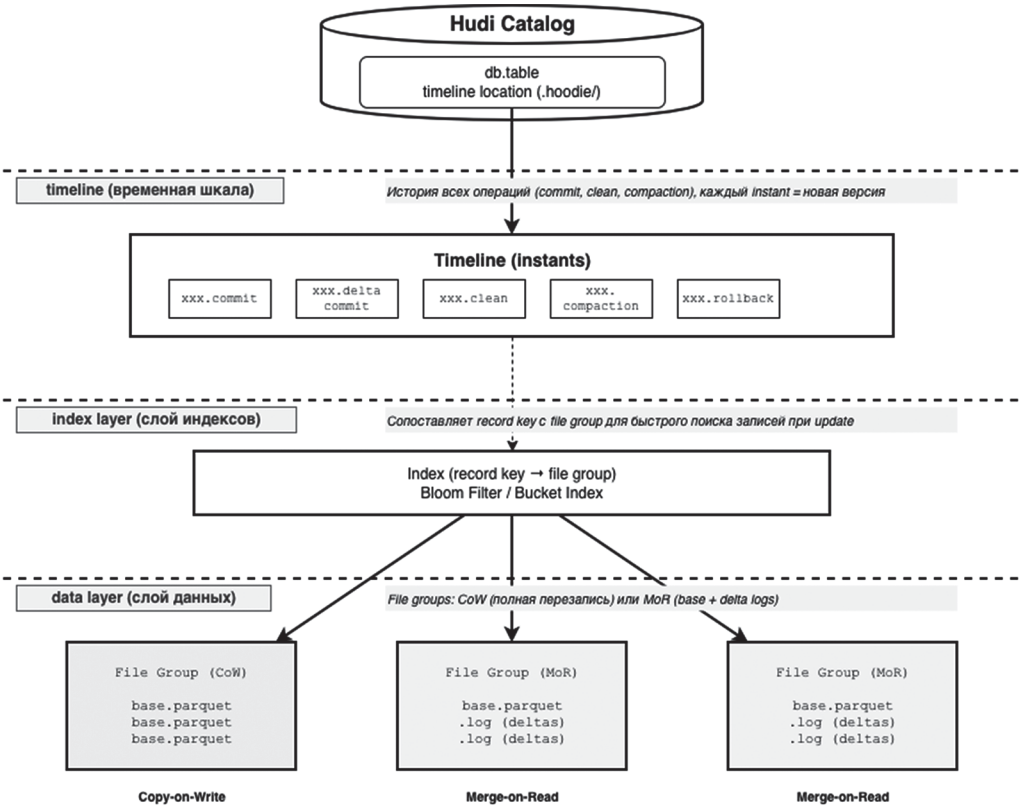


**Рисунок 2.** Схема архитектуры хранения данных в Apache Hudi

***3.2. Реализация ключевых возможностей Apache Hudi.*** Функциональность Hudi ориентирована на сценарии потоковой обработки данных.

*Инкрементальное чтение* (Incremental Queries). Одной из фундаментальных возможностей формата является получение потока записей, измененных с момента конкретного коммита. Это позволяет строить цепочки ETL-задач, где каждый последующий этап обрабатывает только дельту изменений предыдущего, существенно экономя вычислительные ресурсы [12].

*Высокопроизводительные обновления* (Upserts). Благодаря наличию первичных ключей и индексов Hudi предоставляет высокоуровневые примитивы для вставки-или-обновления записей. Это делает формат эффективным инструментом для репликации баз данных (CDC), где необходимо применять поток изменений (binlogs) к целевой таблице с минимальной задержкой.

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

*Встроенные сервисы обслуживания.* В отличие от Iceberg, где оркестрация обслуживания часто делегируется внешним инструментам, Hudi включает в себя сервисы для управления жизненным циклом таблицы. Процессы Compaction (сжатие логов MoR в Parquet), Clustering (перегруппировка данных для ускорения чтения) и Cleaning (удаление устаревших версий файлов) могут выполняться асинхронно, параллельно с операциями записи [16].

***3.3. Практическая применимость и ограничения Apache Hudi.*** Профиль Hudi – это динамически изменяющиеся данные.

*Сильные стороны:*

• эффективность при интенсивной записи – использование строковых Avro-логов в режиме MoR и наличие индексов позволяет Hudi справляться с нагрузками, характерными для CDC-сценариев, с меньшими задержками по сравнению с подходами, требующими полного сканирования файлов [12];

• готовность к потоковой обработке – возможность инкрементального потребления данных превращает таблицу Hudi в аналог топика Kafka, но с возможностью долговременного хранения истории;

• инструментарий ingestion – наличие утилиты Hudi Streamer позволяет настраивать захват данных из Kafka и реляционных СУБД с минимальным написанием кода [16].

*Основные вызовы:*

• сложность настройки – гибкость Hudi оборачивается необходимостью тонкой настройки (выбор типа индекса, стратегии компакции, управление памятью), некорректная конфигурация может привести к деградации производительности;

• операционная сложность – в режиме MoR требуется строгий мониторинг количества лог-файлов, если процесс компакции отстает от темпа записи, чтение может существенно замедлиться из-за необходимости слияния большого количества файлов «на лету»;

• зависимость от Spark – несмотря на расширение поддержки Flink и Presto/Trino, исторически Hudi наиболее глубоко интегрирован со Spark API, и некоторые оптимизации записи могут быть недоступны на других движках [12].

## *4. Delta Lake*

Delta Lake восходит к внутреннему проекту компании Databricks, нацеленному на привнесение транзакционной надежности в классические Data Lake [9]. Формат стал нативным слоем хранения для экосистемы Spark и платформы Databricks, реализуя парадигму Lakehouse, в которой таблицы обладают свойствами как статических наборов данных, так и потоков изменений [17].

В отличие от Iceberg, стремящегося к архитектурной нейтральности, Delta Lake исторически сфокусирован на глубокой интеграции с вычислительным движком (Spark, Photon), что обеспечивает высокую производительность в «родной» среде, но накладывает определенные экосистемные ограничения [12].

***4.1. Архитектура и модель метаданных Delta Lake.*** Ключевым элементом архитектуры является Delta Log – журнал транзакций, расположенный в поддиректории _delta_log рядом с данными. Журнал представляет собой последовательность JSON-файлов, каждый из которых описывает атомарный коммит (добавление/удаление файлов, изменение схемы) [9; 17].

Для ускорения восстановления состояния таблицы периодически создаются Checkpoints в формате Parquet, агрегирующие историю изменений. При чтении движок сначала загружает последний чекпоинт, а затем последовательно применяет JSON-коммиты. Эта модель обеспечивает высокую скорость работы с метаданными [19].

Транзакционная модель реализует принципы оптимистической блокировки, описанные в разделе 3.1. Разрешение конфликтов происходит на уровне логических версий журнала, что позволяет избегать использования файловых блокировок, ненадежных в облачных объектных хранилищах[1] [9]. Схема архитектуры хранения данных в Delta Lake представлена на Рисунке 3.

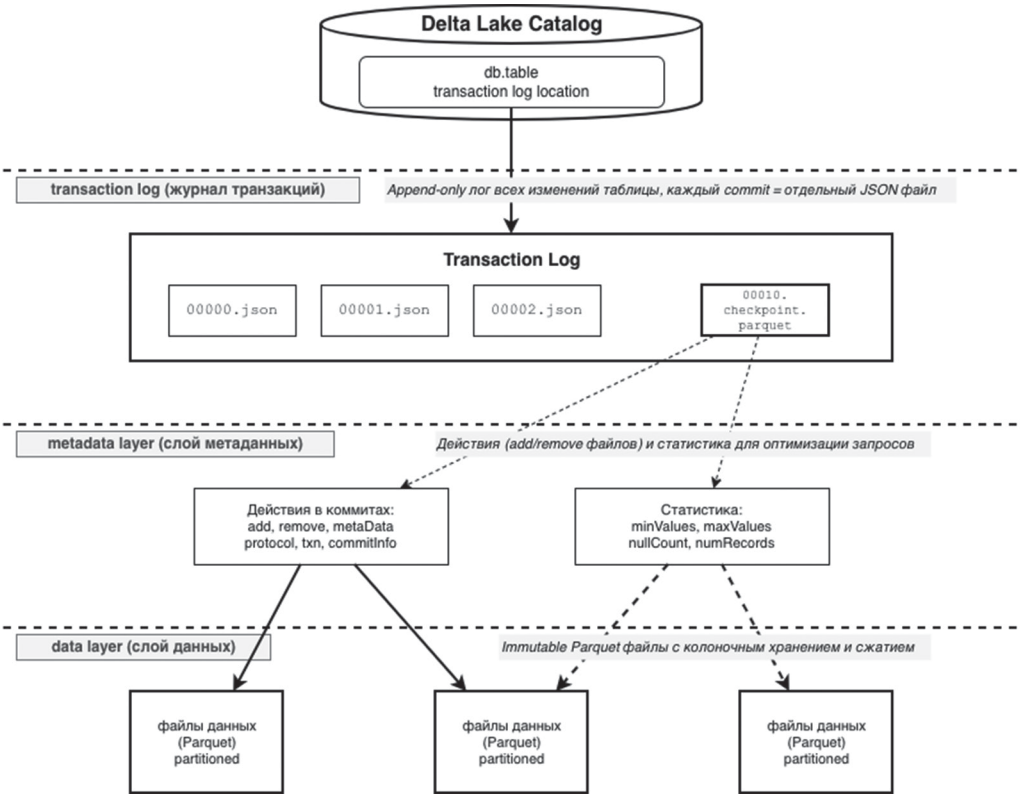


**Рисунок 3.** Схема архитектуры хранения данных в Delta Lake

***4.2. Реализация ключевых возможностей Delta Lake.*** Функциональность Delta Lake направлена на унификацию пакетной и потоковой обработки.

Управление схемой (Schema Enforcement & Evolution). Delta Lake реализует строгий контроль схемы при записи (schema-on-write-поведение). Любая попытка записать данные с несовместимыми типами или лишними колонками блокируется по умолчанию, что

[1] *Polyzos G.* Apache Paimon: The Streaming Lakehouse // Ververica. 2023. October 04. URL: https://www.ververica.com/blog/apache-paimon-the-streaming-lakehouse (дата обращения: 05.12.2025).

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

предотвращает загрязнение данных (data pollution). При этом поддерживается безопасная эволюция схемы через явные команды или настройки mergeSchema [17; 20].

*Оптимизация хранения* (Z-Ordering). Уникальной особенностью Delta Lake является встроенная поддержка Z-Order-кластеризации. Эта техника переупорядочивает данные внутри файлов так, чтобы записи с близкими значениями ключевых колонок находились физически рядом. Это кардинально повышает эффективность пропуска данных (Data Skipping) при фильтрации по нескольким атрибутам одновременно [17].

*Универсальность доступа* (UniForm). Ответом на проблему фрагментации форматов стала технология Universal Format (UniForm). Она позволяет Delta Lake автоматически генерировать метаданные для Iceberg и Hudi поверх тех же файлов данных. Это дает возможность читать таблицу Delta Lake из движков, не имеющих нативной поддержки формата (например, Big Query или Snowflake, ориентированных на Iceberg), без физического дублирования данных [17; 19].

***4.3. Практическая применимость и ограничения Delta Lake.*** Delta Lake является выбором по умолчанию для пользователей платформы Databricks и стека Apache Spark.

*Сильные стороны:*

• простота эксплуатации – единый журнал JSON-файлов проще в отладке и понимании, чем сложные иерархические структуры;

• производительность в Spark – благодаря проприетарным и открытым оптимизациям (Photonengine, Z-Ordering, Delta Cache), формат показывает высочайшую производительность в «родной» среде [12];

• Change Data Feed (CDF) – механизм, позволяющий эффективно читать поток изменений (INSERT, UPDATE, DELETE) между версиями таблицы, упрощая построение инкрементальных ETL-конвейеров [17].

*Основные вызовы:*

• экосистемная зависимость – несмотря на открытость протокола, большинство продвинутых функций (OPTIMIZE, Z-ORDER) максимально эффективно работают именно в связке со Spark, поддержка в других движках (Trino, Flink) часто реализуется с отставанием по функциональности [12];

• управление историей – интенсивные обновления порождают большое количество JSON-файлов в логе. Процедура очистки (VACUUM) является критически важной, без нее стоимость хранения и время листинга метаданных могут неконтролируемо расти [17];

• сложность совместимости – для работы с инструментами, заточенными под Iceberg, требуется явная активация UniForm, что добавляет слой абстракции и потенциальных точек отказа [17].

## *5. Apache Paimon*

Apache Paimon (ранее Flink Table Store) – проект, выросший из экосистемы Apache Flink при активном участии инженеров Alibaba и Ververica[2]. В отличие от универсальных форматов, Paimon изначально создавался не как замена классическим DWH, а как специализированное хранилище для потоковых архитектур (Streaming Lakehouse)[3]. Его ключевая задача – обеспе-

[2] *Polyzos G.* Apache Paimon: The Streaming Lakehouse // Ververica. 2023. October 04. URL: https://www.ververica.com/blog/apache-paimon-the-streaming-lakehouse (дата обращения: 05.12.2025); Apache Paimon Documentation // Apache Software Foundation. URL: https://paimon.apache.org/docs/1.1 (дата обращения: 03.12.2025).

[3] Там же.

чить высокую производительность записи для скоростных потоков данных и поддержку обновлений в реальном времени, что исторически было слабым местом файловых форматов.

***5.1. Архитектура и модель метаданных Apache Paimon.*** Уникальной чертой Paimon является адаптация архитектуры LSM-дерева (Log-Structured Merge-tree), традиционно используемой в NoSQL-базах (RocksDB, Cassandra), к работе поверх распределенных файловых систем и объектных хранилищ (S3/HDFS) [21; 22].

Данные в Paimon организуются в два логических уровня:

1) Write-Optimized (WO) – буфер для новых данных, куда запись происходит с минимальной задержкой;

2) Read-Optimized (RO) – слой, содержащий отсортированные и сжатые данные, прошедшие процедуру слияния.

Такая архитектура решает фундаментальную проблему write amplification в озерах данных: вместо переписывания огромных Parquet-файлов при каждом обновлении (как в CoW), Paimon быстро сбрасывает дельты и выполняет их фоновое слияние (Compaction). Метаданные управляются через систему снапшотов, что обеспечивает ACID-гарантии и возможность чтения согласованных версий[4].

Схема архитектуры хранения данных в Apache Paimon представлена на Рисунке 4.

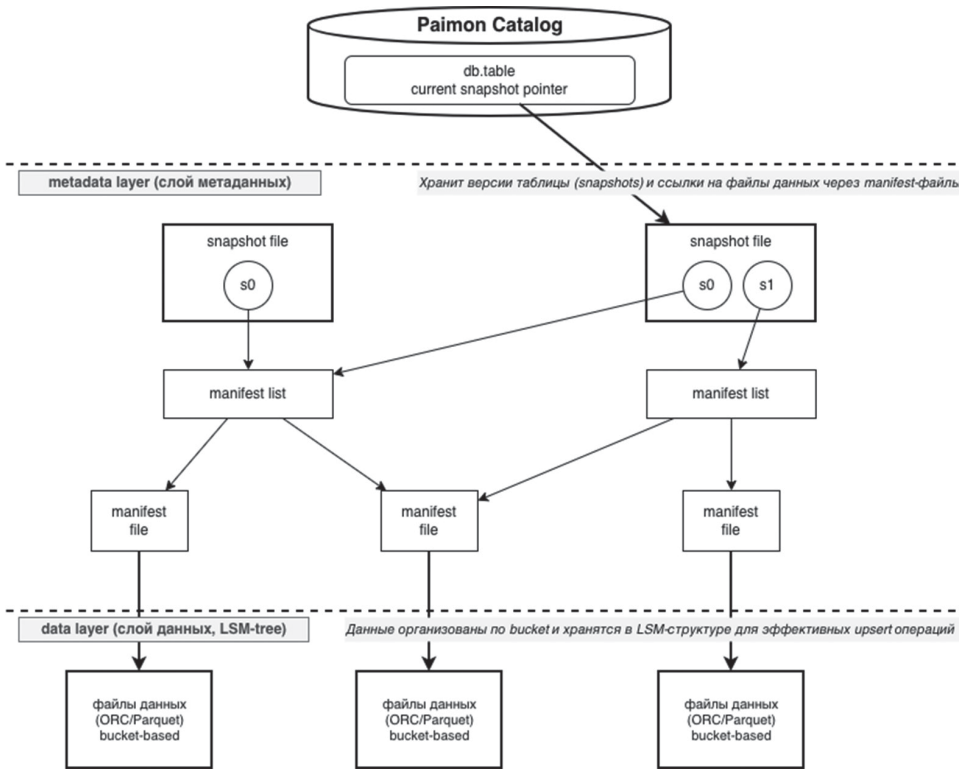


**Рисунок 4.** Схема архитектуры хранения данных в Apache Paimon

[4] Apache Paimon Documentation // Apache Software Foundation. URL: https://paimon.apache.org/docs/1.1 (дата обращения: 03.12.2025).

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

***5.2. Реализация ключевых возможностей Apache Paimon.*** Функциональность Paimon сфокусирована на потребностях потоковой обработки.

*Поддержка Primary Key* и *Upsert*. Paimon предоставляет полноценную семантику первичных ключей. При поступлении записи с существующим ключом формат автоматически обрабатывает коллизию (merge/replace). Это позволяет использовать Paimon как аналог Key-Value хранилища для задач, где допустима небольшая задержка, например, для хранения текущего состояния стриминговых витрин[5] [22].

*Генерация Changelog* (CDC). Благодаря LSM-структуре Paimon способен эффективно генерировать поток изменений (change stream) для систем-потребителей. Читатели могут получать полную картину изменений (-U, +U, +I, -D), что упрощает построение цепочек потоковой обработки без использования внешних брокеров сообщений (как Kafka) [22; 23].

*Нативная интеграция с Flink*. Paimon использует внутренние механизмы Flink для управления ресурсами и чекпоинтами. Это позволяет синхронизировать коммиты в таблицу с чекпоинтами Flink-задачи, обеспечивая гарантии доставки exactly-once[6] [22].

***5.3. Практическая применимость и ограничения Apache Paimon.*** Paimon занимает нишу специализированного хранилища для Real-time Lakehouse.

*Сильные стороны:*

• низкая задержка записи – LSM-архитектура обеспечивает высочайшую пропускную способность для операций upsert, недостижимую для классических форматов;

• унификация Batch и Stream – Paimon является «родным» форматом для Flink SQL, позволяя прозрачно переключаться между пакетным и потоковым режимами работы с одной и той же таблицей[7];

• эффективность обновлений – встроенная поддержка частичного обновления колонок (partial update) делает формат идеальным для сборки «широких витрин» из разных потоков данных[8].

*Основные вызовы:*

• молодость проекта – экосистема Paimon значительно меньше, чем у Iceberg или Delta Lake. Количество интеграций с BI-инструментами и каталогами данных пока ограничено;

• сложность чтения (Read Amplification) – чтение свежих данных требует слияния нескольких слоев файлов «на лету», что может быть медленнее, чем чтение из оптимизированных Parquet-файлов в Iceberg;

• специализация Paimon – это в первую очередь инструмент для Flink, хотя поддержка Spark и Trino существует, она часто отстает по возможностям от эталонной реализации.

## *6. Сравнительный анализ*

***6.1. Табличное сравнение.*** Для выбора табличного формата недостаточно перечислить поддерживаемые функции; важно понимать, как различия в модели метаданных, механизмах ACID-гарантий, обновлениях на уровне строк и компакции отражаются на кон-

---

[5] Там же.

[6] Там же.

[7] *Polyzos G.* Apache Paimon: The Streaming Lakehouse // Ververica. 2023. October 04. URL: https://www.ververica.com/blog/apache-paimon-the-streaming-lakehouse (дата обращения: 05.12.2025).

[8] Там же.

кретных нагрузках – от batch-аналитики до стриминговых сценариев. Форматы развивались в разных экосистемах (Spark, Flink, Presto), поэтому их сильные стороны напрямую связаны с историей развития соответствующих платформ.

В Таблице 1 приведено сопоставление ключевых характеристик Apache Iceberg, Apache Hudi, Delta Lake и Apache Paimon по основным архитектурным и эксплуатационным критериям.

*Таблица 1*

**Сравнительные характеристики Iceberg, Hudi, Delta Lake и Paimon**

| Критерий | Apache Iceberg | Apache Hudi | Delta Lake | Apache Paimon |
|---|---|---|---|---|
| ACID-гарантии | Полные ACID, snapshot isolation, атомарные commits в каталоге | Полные ACID через timeline, гарантии консистентности при incremental pulls | ACID через Delta Log, оптимизированный механизм commit | ACID-транзакции со snapshot-моделью поверх LSM-структур |
| Операции (update/delete/merge) | Merge, Update, Delete, Row-level operations без переписывания файлов | Upsert, Merge-on-Read, Copy-on-Write, Delete | Upsert, Merge, Update, Delete, оптимизация для Spark workloads | Primary key updates, full/partial updates, append-only tables |
| Schema evolution | Свободная эволюция: добавление, удаление, переименование столбцов без переработки данных | Поддержка зависит от таблицы: CoW гибче, MoR ограничен | Гибкая схема, но менее многоступенчатая, чем в Iceberg | Базовая поддержка, эволюция схемы постепенно расширяется |
| Партиционирование | Hidden partitioning, автоматическое ведение partition spec, partition evolution | Классическое партиционирование + metadata indexes | Стандартные partition стратегии, auto-optimize для Databricks | Партиции + bucket-шардинг, оптимизированный под Flink |
| Time travel | Богатые возможности на уровне snapshot и metadata logs | Time travel через timeline, удобен для CDC-пайплайнов | Версионирование через DeltaLog, быстрый rollback | Snapshot- и tag-based time travel, глубина истории по умолчанию краткосрочная |
| Поддержка движков | Spark, Flink, Trino, Presto, Hive, Snowflake, DuckDB | Spark, Flink, Hive, Trino | Spark/Databricks как основная экосистема | Нативная интеграция с Flink, остальные движки в стадии развития |
| Зрелость и сообщество | Крупные корпорации, высокий темп развития | Активное сообщество, большое число индустриальных внедрений | Очень зрелая экосистема Spark | Проект активно растёт, но пока заметно моложе конкурентов |
| Производительность | Отличен в аналитических запросах с большими таблицами | Сильная сторона – ingestion и частые updates | Максимальная производительность в Spark средах | Высокая скорость при real-time ingestion и changelog workloads |

*Источник:* здесь и далее таблицы составлены авторами.

Расширенное сравнение показывает, что форматы развиваются в разных «экосистемных нишах». Iceberg – наиболее зрелый и универсальный вариант для многодвижковых инфраструктур. Hudi сконцентрирован на ingestion и upsert-heavy сценариях. Delta Lake оптимизирован под Spark-архитектуры, где важна скорость развёртывания и стабиль-

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

ность процессов. Paimon формирует свою нишу вокруг Flink и задач real timestreaming, где критичны мгновенное обновление состояния и changelog-модель.

***6.2. Рекомендованные сценарии использования.*** Сильные и слабые стороны каждого формата проявляются только в контексте конкретной архитектуры: объёмов данных, модели обновлений, требований к задержке и используемых движков. Ни один из форматов не является универсальным: Iceberg выбирают не «ради ACID», а из-за гибкости в multi-engine-средах; Delta – там, где инфраструктура уже построена вокруг Spark/Databricks; Hudi – в проектах с интенсивными upsert и CDC-нагрузками; Paimon – в Flink-ориентированных системах, где критичны низкая задержка и changelog-семантика.

Для фиксации типичных сценариев использования в Таблице 2 приведены обобщённые рекомендации по выбору формата с привязкой к практическим сценариям использования.

*Таблица 2*

**Рекомендации по применению форматов в зависимости от сценария**

| Формат | Оптимальные сценарии использования | Когда формат даёт максимальный эффект | Основные ограничения |
|---|---|---|---|
| Apache Iceberg | Универсальные Lakehouse-платформы, кросс-движковая аналитика, долгосрочное хранение | В гетерогенных средах, когда требуется строгий контроль над метаданными, гибкое управление схемой | Накладные расходы на метаданные для малых таблиц. Требует внешней оркестрации обслуживания (compaction/expiration) |
| Apache Hudi | CDC-процессы, Near-Real-Time ingestion, сценарии с интенсивными обновлениями | При обработке потоков upsert/delete с низкой задержкой (минуты) | Высокая сложность настройки (сотни параметров). Риск деградации чтения в MoR, если compaction не успевает за записью |
| Delta Lake | Spark-native проекты, Databricks-экосистема, быстрый старт BI/ML нагрузок | Если инфраструктура построена на Spark и критична простота и скорость | Ограниченная межплатформенность, меньше гибкости, чем Iceberg, Зависимость от VACUUM для контроля размера лога |
| Apache Paimon | Streaming-first архитектуры, Flink-based CDC, динамические витрины (Dynamic Tables) | Когда необходимо объединить потоковую запись и аналитические запросы с секундной задержкой (LSM-tree) | Read amplification (медленное чтение свежих данных без compaction). Меньшая зрелость экосистемы и инструментов за пределами Flink |

Сценарный анализ подтверждает, что каждый формат формирует собственную область эффективности. Iceberg – выбор стратегических, долгоживущих систем. Hudi – инструмент там, где обновления преобладают над чтением. Delta Lake демонстрирует наилучшую интеграцию со Spark, а Paimon – в потоковых Flink-нагрузках. Именно сочетание архитектурной зрелости, операционных требований и движковой совместимости определяет, какой формат становится оптимальным для конкретной компании.

## *Заключение и выводы*

Проведённый анализ показывает, что развитие форматов хранения данных связано с попыткой объединить транзакционную строгость классических Data Warehouse и гибкость Data Lake в единой архитектуре. Iceberg, Hudi, Delta Lake и Paimon выросли из раз-

ных технических традиций и экосистем (Spark, Flink, Databricks), но в каждом случае прослеживается общая тенденция: усиление транзакционной надёжности, снижение стоимости обновлений и расширение совместимости с аналитическими и стриминговыми движками.

Перспективы Lakehouse-архитектуры выглядят устойчивыми: фокус постепенно смещается к real-time и near-real-time-аналитике, что стимулирует внедрение changelog-модели, эффективных индексов и автоматической компакции. При выборе формата организации должны исходить не из абстрактных преимуществ, а из характера собственных данных и требований к задержкам. Iceberg оправдан в многодвижковых и долгоживущих системах – там, где объём обновлений сопоставим или превышает объём чтения, Delta Lake остаётся естественным выбором для Spark/Databricks-инфраструктур; Paimon логично использовать в стриминговых Flink-архитектурах с PK-таблицами и жёсткими требованиями к латентности.

Резюмируя, можно утверждать, что Open Table Formats стали необходимым стандартом для построения современных аналитических платформ. Они решают фундаментальные проблемы согласованности и управляемости, свойственные первому поколению озёр данных, сохраняя при этом их масштабируемость и экономичность. Дальнейшая эволюция этих систем, вероятно, будет направлена на автоматизацию обслуживания и оптимизацию работы со сверхмалыми задержками, окончательно стирая границы между пакетной и потоковой обработкой данных.

**Литература**

1. *Devlin B.A., Murphy P.T.* An architecture for a business and information system // IBM Systems Journal. 1988. Vol. 27. No. 1. P. 60–80. DOI: 10.1147/sj.271.0060

2. *Armbrust M., Ghodsi A., Xin R., Zaharia M.* Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics // 11th Annual Conference on Innovative Data Systems Research (CIDR '21). January 11–15, 2021. URL: https://15721.courses.cs.cmu.edu/spring2023/papers/02-modern/armbrust-cidr21.pdf (дата обращения: 03.12.2025).

3. *Демидов А.А.* Об особенностях организации СУБД в MPP-системе // Программные системы: теория и приложения. 2014. № 4 (22). С. 195–205. URL: http://psta.psiras.ru/read/psta2014_4_195-205.pdf (дата обращения: 03.12.2025). EDN TEYRYB.

4. *Harby A., Zulkernine F.H.* From Data Warehouse to Lakehouse: A Comparative Review // IEEE International Conference on Big Data. Osaka, Japan, 2022. P. 389–395. DOI: 10.1109/BigData55660.2022.10020719

5. *Yessad L., Labiod A.* Comparative study of data warehouses modeling approaches: Inmon, Kimball and Data Vault // 2016 International Conference on System Reliability and Science (ICSRS). Paris, France, 2016. P. 95–99. DOI: 10.1109/ICSRS.2016.7815845

6. *Hai R., Koutras C., Quix C., Jarke M.* Data Lakes: A Survey of Functions and Systems // IEEE Transactions on Knowledge and Data Engineering. 2023. Vol. 35. No. 12. P. 12571–12590. DOI: 10.1109/TKDE.2023.3270101

7. *Azzabi S., Alfughi Z., Ouda A.* Data Lakes: A Survey of Concepts and Architectures // Computers. 2024. Vol. 13. No. 7. Article no. 183. DOI: https://doi.org/10.3390/computers13070183

8. *Thusoo A., Sarma J.S., Jain N., et al.* Hive – A Petabyte Scale Data Warehouse Using Hadoop // 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), Long Beach, CA, USA. 2010. P. 996–1005. DOI: 10.1109/ICDE.2010.5447738

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

9. *Armbrust M., Das T., Sun L., et al.* Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores // Proceedings of the VLDB Endowment. 2020. Vol. 13. No. 12. P. 3411–3424. DOI: https://doi.org/10.14778/3415478.3415560
10. *Shiran T., Hughes J., Merced A.* Apache Iceberg: The Definitive Guide. Data Lakehouse Functionality, Performance and Scalability on the Data Lake. 1st edition. Sebastopol, CA : O'Reilly Media, 2024. ISBN 1098148622.
11. *Muvva S.* Standardizing Open Table Formats for Big Data Analysis: Implications for Machine Learning and AI Applications // Journal of Artificial Intelligence & Cloud Computing. 2023. September. DOI: 10.47363/JAICC/2023(2)E241
12. *Eswararaj D., Nellipudi A.B., Kollati V.* A Comparative Study of Delta Parquet, Iceberg and Hudi for Automotive Data Engineering Use Cases // SSRG International Journal of Computer Science and Engineering. 2025. Vol. 12. No. 7. Pp. 31–42. DOI: https://doi.org/10.14445/23488387/IJCSE-V12I7P104
13. *AbouZaid A., Barclay P. J., Chrysoulas C., Pitropakis N.* Building a Modern Data Platform Based on the Data Lakehouse Architecture and Cloud-Native Ecosystem // Discover Applied Sciences. 2025. Vol. 7. Article no. 166. DOI: https://doi.org/10.1007/s42452-025-06545-w
14. *Okolnychyi A., Sun C., Tanimura K., et al.* Petabyte-Scale Row-Level Operations in Data Lakehouses // Proceedings of the VLDB Endowment. 2024. Vol. 17. No. 12. P. 4159–4172. DOI: https://doi.org/10.14778/3685800.3685834
15. *Gruenheid A., Camacho-Rodríguez J., Curino C., et al.* AutoComp: Automated Data Compaction for Log-Structured Tables in Data Lakes // Companion of the 2025 International Conference on Management of Data (SIGMOD/PODS '25). Berlin, Germany, June 22–27, 2025. P. 404–417. DOI: https://doi.org/10.1145/3722212.3724430
16. *Xu S., Wason P., Saktheeswaran B.S., Bilbro R.* Apache Hudi: The Definitive Guide. Building Robust, Open and High-Performing Data Lakehouses. Santa Rosa, CA : O'Reilly Media, 2025. 290 p. ISBN 109817383X.
17. *Lee D., Wentling T., Haines S., Babu P.* Delta Lake: The Definitive Guide. Modern Data Lakehouse Architectures with Data Lakes. Sebastopol, CA : O'Reilly Media, 2024. 380 p. ISBN 1098151941.
18. *Qu J., Wang J.* Real-time Data Warehousing in the Big Data Environment: A Comprehensive Review of Implementation in the Internet Industry // Applied and Computational Engineering. 2024. Vol. 88. No. 1. P. 110–119. DOI: https://doi.org/10.54254/2755-2721/88/20241643
19. *Katari A.* Delta Lake in Fintech: Enhancing Data Lake Reliability with ACID Transactions // International Research Journal of Modernization in Engineering Technology and Science. 2020. Vol. 2. No. 5. P. 1250–1260. DOI: https://doi.org/10.56726/IRJMETS1372
20. *Muvva S.* The Role of Data Lake and Delta Lake in Big Data, Machine Learning and Artificial Intelligence // International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences. 2022. Vol. 10. No. 6. P. 1–6. DOI: https://doi.org/10.5281/zenodo.14535503
21. *Ievlev K., Surpin V.* Batch Updates and CDC at Scale: A Comparative Study of Iceberg and Paimon // Conference of Open Innovations Association, FRUCT. 2025. No. 38. P. 370–376. URL: https://www.fruct.org/files/publications/volume-38/acm38/Iev.pdf (дата обращения: 03.12.2025). EDN YASDFU.
22. *Mishra S.* A survey of LSM-Tree based Indexes, Data Systems and KV-stores // 2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS). 2024. Vol. 1. P. 1–6. DOI: 10.1109/SCEECS61402.2024.10482249
23. *Sahu R.* Real-time Data Integration: The Evolution of CDC Architecture // Journal of Information Systems Engineering & Management. 2025. Vol. 10. No. 58s. P. 605–615. DOI: https://doi.org/10.52783/jisem.v10i58s.12639

**References**

1. Devlin B.A., Murphy P.T. (1988) An architecture for a business and information system. *IBM Systems Journal*. Vol. 27. No. 1. Pp. 60–80. DOI: 10.1147/sj.271.0060

2. Armbrust M., Ghodsi A., Xin R., Zaharia M. (2021) Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In: *11th Annual Conference on Innovative Data Systems Research (CIDR '21)*. January 11–15, 2021. URL: https://15721.courses.cs.cmu.edu/spring2023/papers/02-modern/armbrust-cidr21.pdf (accessed 03.12.2025).

3. Demidov A.A. (2014) About the features of DBMS architecture in MPP-system. *Program Systems: Theory and Applications*. No. 4 (22). Pp. 195–205. URL: http://psta.psiras.ru/read/psta2014_4_195-205.pdf (accessed 03.12.2025) (In Russian).

4. Harby A., Zulkernine F.H. (2022) From Data Warehouse to Lakehouse: A Comparative Review. In: *IEEE International Conference on Big Data*. Osaka, Japan, 2022. Pp. 389–395. DOI: 10.1109/BigData55660.2022.10020719

5. Yessad L., Labiod A. (2016) Comparative study of data warehouses modeling approaches: Inmon, Kimball and Data Vault. In: *2016 International Conference on System Reliability and Science (ICSRS)*. Paris, France, 2016. P. 95–99. DOI: 10.1109/ICSRS.2016.7815845

6. Hai R., Koutras C., Quix C., Jarke M. (2023) Data Lakes: A Survey of Functions and Systems. In: *IEEE Transactions on Knowledge and Data Engineering*. Vol. 35. No. 12. Pp. 12571–12590. DOI: 10.1109/TKDE.2023.3270101

7. Azzabi S., Alfughi Z., Ouda A. (2024) Data Lakes: A Survey of Concepts and Architectures. *Computers*. Vol. 13. No. 7. Article no. 183. DOI: https://doi.org/10.3390/computers13070183

8. Thusoo A., Sarma J.S., Jain N., et al. (2010) Hive – A Petabyte Scale Data Warehouse Using Hadoop. In: *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. Long Beach, CA, USA. 2010. Pp. 996–1005. DOI: 10.1109/ICDE.2010.5447738

9. Armbrust M., Das T., Sun L., et al. (2020) Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. *Proceedings of the VLDB Endowment*. Vol. 13. No. 12. Pp. 3411–3424. DOI: https://doi.org/10.14778/3415478.3415560

10. Shiran T., Hughes J., Merced A. (2024) *Apache Iceberg: The Definitive Guide. Data Lakehouse Functionality, Performance and Scalability on the Data Lake*. 1st edition. Sebastopol, CA : O'Reilly Media. ISBN 1098148622.

11. Muvva S. (2023) Standardizing Open Table Formats for Big Data Analysis: Implications for Machine Learning and AI Applications. *Journal of Artificial Intelligence & Cloud Computing*. September. DOI: 10.47363/JAICC/2023(2)E241

12. Eswararaj D., Nellipudi A.B., Kollati V. (2025) A Comparative Study of Delta Parquet, Iceberg and Hudi for Automotive Data Engineering Use Cases. *SSRG International Journal of Computer Science and Engineering*. Vol. 12. No. 7. Pp. 31–42. DOI: https://doi.org/10.14445/23488387/IJCSE-V12I7P104

13. AbouZaid A., Barclay P. J., Chrysoulas C., Pitropakis N. (2025) Building a Modern Data Platform Based on the Data Lakehouse Architecture and Cloud-Native Ecosystem. *Discover Applied Sciences*. Vol. 7. Article no. 166. DOI: https://doi.org/10.1007/s42452-025-06545-w

14. Okolnychyi A., Sun C., Tanimura K., et al. (2024) Petabyte-Scale Row-Level Operations in Data Lakehouses. In: *Proceedings of the VLDB Endowment*. Vol. 17. No. 12. Pp. 4159–4172. DOI: https://doi.org/10.14778/3685800.3685834

15. Gruenheid A., Camacho-Rodríguez J., Curino C., et al. (2025) AutoComp: Automated Data Compaction for Log-Structured Tables in Data Lakes. In: *Companion of the 2025 International Conference on Management of Data (SIGMOD/PODS '25)*. Berlin, Germany, June 22–27, 2025. Pp. 404–417. DOI: https://doi.org/10.1145/3722212.3724430

Вестник Российского нового университета
Серия «Сложные системы: модели, анализ и управление». 2026. № 1

16. Xu S., Wason P., Saktheeswaran B.S., Bilbro R. (2025) *Apache Hudi: The Definitive Guide. Building Robust, Open and High-Performing Data Lakehouses*. Santa Rosa, CA : O'Reilly Media. 290 p. ISBN 109817383X.
17. Lee D., Wentling T., Haines S., Babu P. (2024) *Delta Lake: The Definitive Guide. Modern Data Lakehouse Architectures with Data Lakes.* Sebastopol, CA : O'Reilly Media. 380 p. ISBN 1098151941.
18. Qu J., Wang J. (2024) Real-time Data Warehousing in the Big Data Environment: A Comprehensive Review of Implementation in the Internet Industry. *Applied and Computational Engineering.* Vol. 88. No. 1. Pp. 110–119. DOI: https://doi.org/10.54254/2755-2721/88/20241643
19. Katari A. (2020) Delta Lake in Fintech: Enhancing Data Lake Reliability with ACID Transactions. *International Research Journal of Modernization in Engineering Technology and Science*. Vol. 2. No. 5. Pp. 1250–1260. DOI: https://doi.org/10.56726/IRJMETS1372
20. Muvva S. (2022) The Role of Data Lake and Delta Lake in Big Data, Machine Learning and Artificial Intelligence. *International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences.* Vol. 10. No. 6. Pp. 1–6. DOI: https://doi.org/10.5281/zenodo.14535503
21. Ievlev K., Surpin V. (2025) Batch Updates and CDC at Scale: A Comparative Study of Iceberg and Paimon. In: *Conference of Open Innovations Association, FRUCT.* No. 38. Pp. 370–376. URL: https://www.fruct.org/files/publications/volume-38/acm38/Iev.pdf (accessed 03.12.2025).
22. Mishra S. (2024) A survey of LSM-Tree based Indexes, Data Systems and KV-stores. In: *2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*. Vol. 1. Pp. 1–6. DOI: 10.1109/SCEECS61402.2024.10482249
23. Sahu R. (2025) Real-time Data Integration: The Evolution of CDC Architecture. *Journal of Information Systems Engineering & Management.* Vol. 10. No. 58s. Pp. 605–615. DOI: https://doi.org/10.52783/jisem.v10i58s.12639

Поступила в редакцию: 30.12.2025
Поступила после рецензирования: 22.01.2026
Принята к публикации: 09.02.2026

Received: 30.12.2025
Revised: 22.01.2026
Accepted: 09.02.2026