Оглавление

[Введение 3](#_Toc154337385)

[Методы построения мульмодельных БД 13](#_Toc154337386)

[Обзор мультимодельных БД 17](#_Toc154337387)

[MongoDB 17](#_Toc154337388)

[ArangoDB 20](#_Toc154337389)

[OrientDB 22](#_Toc154337390)

[MarkLogic 26](#_Toc154337391)

[Azure Cosmos DB 32](#_Toc154337392)

[Выводы 33](#_Toc154337393)

[Литература 33](#_Toc154337394)

# **Введение**

-) В кратце о ммбд (1-2 ист)

Вот есть реляционные их краткое описание

А вот есть ммд источник 1

И что-то из источника 2

#### Multi-model Database A \*\*multi-model database\*\* is a database management system designed to support multiple data models against a single, integrated backend. This means that it can store, index, and query data in more than one model, such as document, graph, relational, and key-value models. This type of database provides a unified interface for data consistency, security, and access, and it eliminates the need for complex transformations and migrations between different databases [[1]](<https://en.wikipedia.org/wiki/Multi-model_database>).

Многомодельная база данных — это база данных, предназначенная для поддержки нескольких моделей данных в одной системе хранения данных. Это означает, что такая система может хранить, индексировать и запрашивать данные в нескольких моделях. Этот тип базы данных обеспечивает единый интерфейс для обеспечения согласованности, безопасности и доступа к данным, а также устраняет необходимость сложных преобразований и миграций между различными базами данных [1].

-) Какие бывают модели данных и краткое их описание (3-4+ ист)

Right now there are four main categories of NoSQL databases: key-value, column family, document and graph ones [2].

В настоящее время существует четыре основные модели баз данных: ключ-значение, семейство столбцов, документальные и графовые [2].

Возможно не хватает перехода с объяснением про схемы и NoSQL

Описание каждой и источник к ней

This is where Key Value stores come into play. Key value stores allow theapplication developer to store schema-less data. This data is usually consisting of a string which represents the key and the actual data which isconsidered to be the value in the ”key - value” relationship. The data itselfis usually some kind of primitive of the programming language (a string,an integer, an array) or an object that is being marshalled by the programming languages bindings to the key value store. This replaces the need forfixed data model and makes the requirement for properly formatted dataless strict.To make this more clear, here’s a short example (using ruby and the simplebut limited ”pstore” standard library) that should make that concept clearto any programmer.

Базы данных ключ-значения имеют хранилище ключ-значение. Они позволяют разработчику приложения хранить данные без схемы. Эти данные обычно состоят из строки, которая представляет ключ, и фактических данных, которые считаются значением в отношениях «ключ-значение». Сами данные обычно представляют собой своего рода примитив языка программирования (строка, целое число, массив) или объект, который создаётся привязками языков программирования к хранилищу ключ-значение. Это заменяет необходимость в фиксированной модели данных и делает строгими требования к правильному форматированию данных без данных [3].

In summary, a graph db-model is a model in which the data structures for the schemaand/or instances are modeled as a directed, possibly labeled, graph, or generalizationsof the graph data structure, where data manipulation is expressed by graph-orientedoperations and type constructors, and appropriate integrity constraints can be definedover the graph structure..

Графовая модель базы данных — это модель, в которой структуры данных для схемы и/или экземпляров моделируются как направленный, возможно, помеченный граф или обобщение структуры данных графа, где манипулирование данными выражается с помощью графо-ориентированных операций и конструкторов типов, а соответствующие ограничения целостности могут быть определены в структуре графа [4].

The column-family database is another NoSQL databasethat stores data using a column-wise approach, unlikerelational ones, which organize data by rows.Data stored in a column family database is partitionedvertically, which makes partial read more efficient as only asubset of row attributes is loaded.

База данных семейства столбцов - это база данных NoSQL, которая хранит данные с использованием столбцового подхода, в отличие от реляционных, которые упорядочивают данные по строкам. Данные, хранящиеся в базе данных семейства столбцов, выбираются вертикально, что делает частичное чтение более эффективным, поскольку загружается только набор атрибутов строки [5].

Instead of storing data in fixed rows and columns, document databases use flexible documents. A document is a record in a document database. A document typically stores information about one object and any of its related metadata. Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects.

Вместо хранения данных в фиксированных строках и столбцах базы данных документов используют гибкие документы. Документ – это запись в базе данных документов. Документ обычно хранит информацию об одном объекте и любых связанных с ним метаданных. Документы хранят данные в парах поле-значение. Значения могут быть различных типов и структур, включая строки, числа, даты, массивы или объекты [6].

-) Примеры ммбд с описанием какие categories of NoSQL они поддерживают (1-2+ ист + желательно с оф сайта)

Для каждой categories of NoSQL существую собственные ммд:

Перечисление с описанием и источником

Here are some of the most popular NoSQL databases:

1. ArangoDB is a free and open-source database manager that supports key-value, document, and graph database models.

2. The OrientDB engine supports Graph, Document, Key/Value, and Object models, so you can use OrientDB as a replacement for a product in any of these categories.

3. Azure Cosmos DB accommodates operational data models: document, vector, key-value, graph.

Далее представлены некоторые из наиболее популярных мультимодельных баз данных:

1. ArangoDB — это бесплатный менеджер баз данных с открытым исходным кодом, который поддерживает модели баз данных ключ-значение, документ и граф [7].
2. OrientDB поддерживает следующие модели баз данных: граф, документа, ключ-значение [8].
3. Azure Cosmos DB поддерживает следующие модели баз данных: документ, ключ-значение, граф [9].

-) Проблемы ммбд (потенциально в последнюю главу вынести) (1-3 ист)

Ммд имеет проблемы (перефразировать):

Перечисление проблем

—- Multi-model query processing and optimization. Despite ORDBMSs are capable of storing data with various formats (models), they do not provide a cross-modeldata processing language, inter-model compilation or respective multi-model query optimization. In contrast, a multi-model database attempts to embrace this challenge bydeveloping a unified query language to accommodate all the supported data models.As mentioned in the previous sections, there exist proposals of multi-model query languages. For example, AQL provided by ArangoDB enables one to access both graphand document data. However, the existing query languages are immature, and it isstill an open challenge to develop a full-fledged query language for multi-model data.A closely related problem is a proposal of an approach for identification of the optimal query plan for efficient evaluation of a given cross-model query [Lu 2017; Zhanget al. 2018]. Wavelets and histograms enable one to exploit the knowledge of distribution of data and thus optimize query evaluation strategies. However, the currenttechniques (e.g. [Alway and Nica 2016]) are developed for RDBMSs having a fixed relational schema, whereas multi-model DBMSs support both flexible and diverse schema.Thus, new dynamic techniques should be developed capable of adaptation to schemachanges.Currently the single-model DBMSs usually build a separate domain-specific indexfor different domains. Cross-domain queries are then evaluated by (1) separating indexsearches specifically for the individual domain, and (2) integrating the partial resultsto find all solutions. In the multi-model world we can use this approach too. For each ofthe models there exist verified types of indices, such as B-tree and B+-tree for relationaldata, TreePi [Zhang et al. 2007] and gIndex [Yan et al. 2004] for graph data, or XBtree [Bruno et al. 2002] for hierarchical XML data. However, the efficiency of suchapproach is questionable. A natural hypothesis is that a universal index comprisingvarious data models should quite probably be a better solution.In addition, the cloud-based distributed technologies are going forward. Cloud datacan be very diverse, including text, streaming data, unstructured and semi-structureddata. And cloud users and developers may be in high numbers, but not DBMS experts.Therefore, one challenge is to extend the technology of distributed database management and parallel database programming to fulfill the requirement of the scalability,simplicity and flexibility of the cloud-based multi-model data management.—- Multi-model schema design and optimization. A good design of the databaseschema is a critical part influencing many aspects, such as efficiency of query processing, application extensibility etc. There are critical decisions about both the physicaland logical schema of the data. For example, as shown in [Scherzinger et al. 2013] forthe case of key/value stores, a naive schema design will result in 20–35% of databasetransactions failing for a certain workload, whereas this problem can be alleviatedthrough the design of an appropriate schema. A similar paper [Mior 2014] provides acost-based approach to schema optimization in column stores. Contrary to relationaldatabases, NoSQL databases usually use significantly denormalized physical schemawhich requires additional space. Hence, in the world of multi-model systems we encounter contradictory requirements for the distinct models and thus it calls for a newsolution for multi-model schema design to balance and trade-off the diverse requirement of multi-model data.ACM Computing Surveys, Vol. 0, No. 0, Article 0, Publication date: 2019.0:31Even the question of existence of a schema differs significantly – traditional relational databases are based on existence of a pre-defined schema, whereas NoSQLdatabases are based on the assumption of schemalessness. A possible solution mayfind an inspiration, e.g., in the proposal of the NoSQL AbstractModel (NoAM) [Bugiottiet al. 2014], an abstract data model for NoSQL databases that specifies a systemindependent data representation. However, the proposal covers only aggregateoriented NoSQL databases (i.e., key/value, column, and document).A closely related problem of schema inference from a sample set of data instances isanother open issue in the multi-model context. There exists a number of approachesdealing with inference of, e.g., JSON [Baazizi et al. 2017] or XML [Mlynkov ´ a and ´Necasky 2013] schemas. Recently there have appeared approaches inferring a schema ´for NoSQL document stores [Gallinucci et al. 2018a], or in general for aggregateoriented databases [Sevilla Ruiz et al. 2015; Chillon et al. 2017]. There are even meth- ´ods which identify aggregation hierarchies in RDF data [Gallinucci et al. 2018b]. However, in the world of multi-model data we need to infer also references between thedistinct models. In addition, the inference approaches may benefit from informationextracted from related data with distinct models.—- Multi-model evolution. In general, it is a difficult task to efficiently managedata schema evolution and the propagation of the changes to the relevant portions ina database system, such as data instances, queries, indices, or even storage strategies.In some smaller applications a company can rely on a skilled database administrator tomanage the data evolution and to propagate the modification to other impacted partsmanually. But in most cases, it is a complicated and error-prone job.In the context of multi-model databases, this task is more subtle and difficult. Wecan distinguish intra-model and inter-model changes. In the former case we can reuse the existing approaches for single models. In the latter case, however, they cannotbe straightforwardly applied. The state-of-the-art solutions [Polak et al. 2015], usingthe classical Model-Driven Architecture, deal with multiple data models which represent distinct and overlapping views of a common model of the considered reality viawhich a change can be propagated to all affected parts. Then the change propagationcan be solved within particular data models separately. In the case of multi-modeldatabases the distinct models cover separate parts of the reality which are interconnected using references, foreign keys, or similar entities. Hence the evolution management has to be solved across all the supported data models. In addition, the challengeof query rewrite [Curino et al. 2008; Manousis et al. 2013], i.e. propagation of changesto queries, also becomes more complex in case of inter-model changes which requirechanges in data access constructs.—- Multi-model extensibility. The last but not least open problem is the challengeof model extensibility, which can be considered in several scopes. First, we may consider intra-model extensibility which means extending one of the models with newconstructs, e.g., extending the XML model with the support for the query on IDs andIDREF(S). Second, we may consider inter-model extensibility which adds new constructs expressing relations between the models, e.g. the ability to express a CHECKconstraint from the relational model across both relational and XML data. And third,we can provide extra-model extensibility which involves adding a whole new model,together with respective data and query, e.g. adding time series data with the supportof time series analysis.

—- Multi-model query processing and optimization. Despite ORDBMSs are capable of storing data with various formats (models), they do not provide a cross-modeldata processing language, inter-model compilation or respective multi-model query optimization. In contrast, a multi-model database attempts to embrace this challenge bydeveloping a unified query language to accommodate all the supported data models. However, the existing query languages are immature, and it isstill an open challenge to develop a full-fledged query language for multi-model data.A closely related problem is a proposal of an approach for identification of the optimal query plan for efficient evaluation of a given cross-model query.

—- Multi-model schema design and optimization. A good design of the databaseschema is a critical part influencing many aspects, such as efficiency of query processing, application extensibility etc. Contrary to relationaldatabases, NoSQL databases usually use significantly denormalized physical schemawhich requires additional space. Hence, in the world of multi-model systems we encounter contradictory requirements for the distinct models and thus it calls for a newsolution for multi-model schema design to balance and trade-off the diverse requirement of multi-model data. Even the question of existence of a schema differs significantly – traditional relational databases are based on existence of a pre-defined schema, whereas NoSQLdatabases are based on the assumption of schemalessness.

—- Multi-model evolution. In general, it is a difficult task to efficiently managedata schema evolution and the propagation of the changes to the relevant portions ina database system, such as data instances, queries, indices, or even storage strategies.In some smaller applications a company can rely on a skilled database administrator tomanage the data evolution and to propagate the modification to other impacted partsmanually. But in most cases, it is a complicated and error-prone job.In the context of multi-model databases, this task is more subtle and difficult.

—- Multi-model extensibility. The last open problem is the challengeof model extensibility, which can be considered in several scopes. First, we may consider intra-model extensibility which means extending one of the models with newconstructs. Second, we may consider inter-model extensibility which adds new constructs expressing relations between the models. And third,we can provide extra-model extensibility which involves adding a whole new model,together with respective data and query.

Для мультимодельных БД можно выделить следующие проблемы:

* Многомодельная обработка и оптимизация запросов. Несмотря на то, что мультимодельные БД способны хранить данные в различных форматах (моделях), они не обеспечивают язык межмодельной обработки данных, межмодельную компиляцию или соответствующую оптимизацию многомодельных запросов. Напротив, мультимодельная база данных пытается решить эту проблему путем разработки унифицированного языка запросов, который будет охватывать все поддерживаемые модели данных. Однако существующие языки запросов еще незрелы, и разработка полноценного языка запросов для многомодельных данных все еще остается открытой задачей. Тесно связанной проблемой является предложение подхода к определению оптимального плана запроса для эффективной оценки заданный межмодельный запрос.
* Проектирование и оптимизация многомодельной схемы. Хороший дизайн схемы базы данных является важной частью, влияющей на многие аспекты, такие как эффективность обработки запросов, расширяемость приложения и т.д. В отличие от реляционных баз данных, базы данных NoSQL обычно используют значительно денормализованную физическую схему, которая требует дополнительного места. Следовательно, в многомодельных системах мы сталкиваемся с противоречивыми требованиями к различным моделям, и, таким образом, требуется новое решение для проектирования многомодельных схем, чтобы сбалансировать и найти компромисс между разнообразными требованиями к многомодельным данным. Даже вопрос существования схемы существенно различается: традиционные реляционные базы данных основаны на существовании заранее определенной схемы, тогда как базы данных NoSQL основаны на предположении об отсутствии схемы.
* Мультимодельная эволюция. В целом, эффективное управление развитием схемы данных и распространением изменений в соответствующих частях системы базы данных, таких как экземпляры данных, запросы, индексы или даже стратегии хранения, является сложной задачей. В некоторых небольших приложениях компания может положиться на квалифицированный администратор базы данных для управления развитием данных и распространения изменений на другие затронутые части вручную. Но в большинстве случаев это сложная и подверженная ошибкам работа. В контексте многомодельных баз данных эта задача более тонкая и трудная.
* Многомодельная расширяемость. Последней открытой проблемой является проблема расширяемости модели, которую можно рассматривать в нескольких аспектах. Во-первых, мы можем рассмотреть возможность расширения внутри модели, что означает расширение одной из моделей новыми конструкциями. Во-вторых, мы можем рассмотреть межмодельную расширяемость, которая добавляет новые конструкции, выражающие отношения между моделями. В-третьих, мы можем обеспечить дополнительную расширяемость модели, которая включает добавление совершенно новой модели вместе с соответствующими данными и запросом [10].

# **Методы построения мульмодельных БД**

Можно выделить 4 метода построения мультимодельных БД: Polyglot persistence, мультимодельные СУБД на основе реляционной модели, мультимодельные СУБД на основе документной модели, СУБД «без основной модели» [11].

1. Polyglot persistence

Одним из самых известных методов построения БД является Polyglot persistence.

Instead of choosing just one single database management system to store the entire data, so-called polyglot persistence could be a viable option to satisfy allrequirements towards a modern data management infrastructure. Polyglot persistence denotes that one can choose as many databases asneeded so that all requirements are satisfied. Polyglot persistence can in particular be an optimal solution when backward-compatibility with a legacy application must be ensured. The new database system can run alongside the legacy database system; while the legacy application still remains fully functional, novelrequirements can be taken into account by using the new database system.An implementation of a data processing system that connects to several datasources and integrates and merges data from these sources is Apache Drill [2].Apache Drill is inspired by the ideas developed in Google’s Dremel system [6].It should obviously be avoided to push the burden of all of these query handling and database synchronization task to the application level – that is, inthe end to the programmers that maintain the data processing applications. Instead it is usually better to introduce an integration layer. The integration layerthen takes care of processing the queries – decomposing queries in to severalsubqueries, redirecting queries to the appropriate databases and recombiningthe results obtained from the accessed databases; ideally, the integration layershould offer several access methods, and should be able to parse all the differentquery languages of the underlying database systems as well as potentially translate queries into other query languages. Moreover, the integration layer shouldensure cross-database consistency: it must synchronize data in the differentdatabases by propagating additions, modifications or deletions among them.Polyglot persistence however comes with severe disadvantages:– Uniform access: There is no unique query interface or query language, andhence access to the database systems is not unified and requires knowledgeof all needed database access methods.– Consistency: Cross-database consistency is a major challenge because referential integrity must be ensured across databases (for example if a record inone database references a record in another database) and in case data areduplicated (and hence occur in different representation in several databasesat the same time) the duplicates have to be updated or deleted in unison.– Interoperability: The underlying database systems are developed independently. Newer versions of databases may not be interoperable with the integration layer and the administrator has to keep track of frequent updates.– Logical Redundancy: Logical redundancy can only be avoided with a databasedesign that strictly assigns non-intersecting subsets of the data to differentdatabases. This might contradict some access requirements of users.– Security: Access control must be enforced by the integration layer and allconnected databases have to be configured to only allow restricted access.

Этот метод подразумевает выбор столько баз данных, сколько необходимо, чтобы все требования были удовлетворены. Данный метод может быть оптимальным решением, когда необходимо обеспечить обратную совместимость с устаревшим приложением. Новая система баз данных может работать параллельно с устаревшей системой баз данных; хотя устаревшее приложение по-прежнему остается полностью функциональным, новые требования могут быть учтены при использовании новой системы баз данных. Очевидно, следует избегать перекладывания бремени всех этих задач обработки запросов и синхронизации базы данных на уровень приложения, то есть, в конечном итоге, на программистов, которые поддерживают приложения для обработки данных. Вместо этого обычно лучше ввести уровень интеграции. Затем уровень интеграции занимается обработкой запросов: разбивает запросы на несколько подзапросов, перенаправляет запросы в соответствующие базы данных и повторно объединяет результаты, полученные из баз данных, к которым осуществляется доступ. Более того, уровень интеграции должен обеспечивать согласованность между базами данных: он должен синхронизировать данные в различных базах данных путем распространения дополнений, модификаций или удалений между ними. Однако многоязычие сопряжено с серьезными недостатками:

* Отсутствие одинокого способа доступа к данных. Нет уникального интерфейса запроса или языка запросов, и, следовательно, доступ к системам баз данных не унифицирован и требует знания всех необходимых методов доступа к базе данных.
* Согласованность. Согласованность между базами данных является серьезной проблемой, поскольку необходимо обеспечить ссылочную целостность во всех базах данных (например, если запись в одной базе данных ссылается на запись в другой базе данных), а в случае дублирования данных (и, следовательно, в разных представлениях в нескольких базах данных одновременно) дубликаты должны быть обновлены или удалены практически одновременно. Базовые системы баз данных разрабатываются независимо. Более новые версии баз данных могут быть несовместимы с уровнем интеграции, и администратору приходится отслеживать частые обновления.
* Логическая избыточность. Логической избыточности можно избежать только при разработке базы данных, которая строго распределяет непересекающиеся подмножества данных по разным базам данных. Это может противоречить некоторым требованиям пользователей к доступу.
* Безопасность. Уровень интеграции должен обеспечивать контроль доступа, и все подключенные базы данных должны быть настроены так, чтобы разрешать только ограниченный доступ [12].

1. Мультимодельные СУБД на основе реляционной модели

Данный тип включается в себя те СУБД, которые изначально было построены под реляционную модель, но после в не были добавлены Nosql модели.

1. Мультимодельные СУБД на основе документной модели

Данный тип включается в себя те СУБД, которые используют документальную модель как основную, но также имеет поддержку остальных. В качестве основной модели может выбрана любая другая из существующих моделей.

1. СУБД «без основной модели»

Это СУБД, которые не имеют основной модели данных, при этом поддерживает какие-либо другие

# Обзор мультимодельных БД

## MongoDB

-) Основная информация

MongoDB – это не реляционная, а документно-ориентированная система управления базами данных. Документно-ориентированная СУБД заменяет концепцию «строки» более гибкой моделью, «документом». Позволяя использовать вложенные документы и массивы, документно-ориентированный подход дает возможность представлять сложные иерархические отношения с помощью одной записи. Также нет предопределенных схем: ключи и значения документа не имеют фиксированных типов или размеров. Когда нет фиксированной схемы, добавлять или удалять поля по мере необходимости становится проще. MongoDB – СУБД общего назначения, поэтому помимо создания, чтения, обновления и удаления данных она предоставляет большинство тех функций, которые можно ожидать от системы управления базами данных. Специальные типы коллекций и индексов MongoDB поддерживает коллекции данных TTL (time-to-live), срок действия которых должен истечь в определенное время, такие как сеансы и коллекции фиксированного размера, для хранения недавно полученных данных, например, журналов. MongoDB также поддерживает частичные индексы, ограниченные только теми документами, которые соответствуют фильтру критериев, чтобы повысить эффективность и уменьшить необходимый объем дискового пространства [13].

-) Синтаксис языка запросов

Каждый запрос к БД начинается с “db”, после чего идёт символ точки, а затем название коллекции, к которой необходимо обратиться. Так же возможно использоваться collection('name\_ collection ').

Для простого добавления элемента в коллекцию используется метод insert:

db.collection('name\_ collection ').insert( { “elem\_name” : “elem\_value” });

Для добавления множества элементов необходимо через запятую указать нужные:

db.collection('name\_ collection ').insert( { “arr” : [ { a : 1 , b : 1 } , { a : 2 , b : 2 } ] });

Всю информацию из коллекции можно удалить через метод remove (сама коллекция при этом не удаляется):

db.collection('name\_ collection ').remove();

Удалить объект по какому-то параметру необходимо следующим образом:

db.collection('name\_ collection ').remove( { “elem\_name” : “elem\_value” });

Для обновления коллекции используется метод update:

db.collection('name\_ collection ').update( { “elem\_name” : ” elem\_value” });

Но тогда будет произведена замена всего документа, а, чтобы такого не произошло необходимо использовать модификатор “$set”:

db.collection('name\_ collection ').update({ $set: { “elem\_name”: ” new\_elem\_value” }});

Так же с помощью модификатора “$unset”:можно удалить ключ:

db.collection('name\_ collection ').update({ $unset: { “elem\_name”: ” elem\_value” }});

Имеется возможность использования Upsert. При его использовании, если документ по запрашиваемую критерию не найден, то он будет создан, если же найден, то он будет обновлён, как обычно. Чтоб использовать upsert нужно просто в команде update добавить третий параметр равный true:

db.collection('name\_ collection ').update({$set: “elem\_name”: ” new\_elem\_value”}, true);

Для поиска используется Find. Возвращает массив документов в виде коллекции, если документов нет — пустую коллекцию.

db.collection('name\_ collection ').find({ “elem\_name” : ” elem\_value”});

Для добавления условий поиска используется следующие операторы: $lt — меньше, $lte — меньше или равно, $gt — больше, $gte — больше или равно, $ne — не равно.

Так же имеется возможность использовать для поиска регулярные выражения [14].

-) Протоколы взаимодействия

The MongoDB Wire Protocol is a simple socket-based, request-response style protocol. Clients communicate with the database server through a regular TCP/IP socket.

Использует протокол — MongoDB Wire. Это простой протокол в стиле запроса-ответа, основанный на сокетах. Клиенты взаимодействуют с сервером базы данных через обычный сокет TCP/IP [15].

-) Проблемы

<https://habr.com/ru/articles/692736/>

https://www.g2.com/products/mongodb/reviews?utf8=%E2%9C%93&filters%5Bnps\_score%5D=4&order=g2\_default

Одной из больших проблем выделяют отсутствие схемы данных, что затрудняет расширение и валидацию данных. Так же отмечается уменьшение производительности при большом количестве данных. Выделяют сложность поиска из-за языка запросов [16].

## ArangoDB

-) Основная информация

ArangoDB поддерживает три модели данных: графовую, документную и модель «ключ-значение». Работа с базой данных осуществляется при помощи SQL-подобного языка запросов AQL (ArangoDB Query Language). Язык является декларативным и позволяет свободно комбинировать все поддерживаемые модели данных в одном запросе [17].

https://ru.wikipedia.org/wiki/ArangoDB

-) Синтаксис языка запросов

Для ArangoDB разработан свой собственный язык запросов - AQL.

Для добавления значений используется следующая команда:

INSERT {key: value} INTO collection [RETURN NEW]

RETURN NEW позволяет вернуть добавленный объект

Для чтения объектов используется:

RETURN DOCUMENT(\_id) – вернёт объект по \_id

RETURN DOCUMENT(collection\_name, [\_key, ...]) – вернёт список объектов из коллекции по ключам

Для обновления объектов используется:

UPDATE {\_key: "value"} WITH {new\_value: 1234} IN collection

Для удаления используется:

REMOVE {\_key: "value"} IN collection

An example AQL query may look like this:

FOR u IN users

FILTER u.type == "newbie" && u.active == true

RETURN u.name

In this example query, the terms FOR, FILTER, and RETURN initiate the higher-level operation according to their name. These terms are also keywords, meaning that they have a special meaning in the language.

Для получения данных используется RETURN, но с особенностями, дял понимание которых далее приведён пример, где из коллекции users выбираются имена пользователей, которые являются активными:

FOR u IN users

FILTER u.active == true

RETURN u.name

В этом примере запроса термины FOR, FILTER и RETURN инициируют операцию более высокого уровня в соответствии с их переводом [18].

-) Протоколы взаимодействия

ArangoDB exposes its API via HTTP, making the server accessible easily with a variety of clients and tools (e.g. browsers, curl, telnet). The communication can optionally be SSL-encrypted.

ArangoDB предоставляет свой API через HTTP. При необходимости связь может быть зашифрована с помощью SSL [19].

https://docs.arangodb.com/3.11/develop/http-api/general-request-handling/

-) Проблемы

<https://www.g2.com/products/arangodb/reviews?utf8=%E2%9C%93&filters%5Bnps_score%5D=4&order=g2_default#survey-response-105621>

Среди проблем отмечают маленькую популярность, что усложняет решение возникающих вопросов или проблем. Так же иногда отмечают недостаток функционала и, в редких случаях, неполноты документации [20].

## OrientDB

-) Основная информация

OrientDB is an open source NoSQL database management system written in Java. It is a Multi-model database, supporting graph, document, key/value, and object models,[2] but the relationships are managed as in graph databases with direct connections between records. It supports schema-less, schema-full and schema-mixed modes. It has a strong security profiling system based on users and roles and supports querying with Gremlin along with SQL extended for graph traversal. OrientDB uses several indexing mechanisms based on B-tree and Extendible hashing, the last one is known as "hash index", there are plans to implement LSM-tree and Fractal tree index based indexes. Each record has Surrogate key which indicates position of record inside of Array list , links between records are stored either as single value of record's position stored inside of referrer or as B-tree of record positions (so-called record IDs or RIDs) which allows fast traversal (with O(1) complexity) of one-to-many relationships and fast addition/removal of new links. OrientDB is the fifth most popular graph database according to the DB-Engines graph database ranking,[3] as of December 2021.

OrientDB - это система управления базами данных NoSQL с открытым исходным кодом, написанная на Java. Это мультимодельная база данных, поддерживающая графическую, документальную, ключ/значение и объектную модели. Он поддерживает режимы без схемы, с полной схемой и со смешанной схемой. Он имеет систему профилирования безопасности, основанную на пользователях и ролях, и поддерживает запросы с помощью Gremlin наряду с расширенным SQL для обхода графа [21].

https://en.wikipedia.org/wiki/OrientDB

-) Синтаксис языка запросов

В OrientDB таблицы представляются в виде классов, строки - записи в таблице, а свойства класса - поля. В основном OrientDB использует документальную модель, но также можно использовать графовую модель.

Для создания классов используется следующая команда:

CREATE CLASS <class\_name>

Для добавления полей в класс, можно использовать следующую команду:

INSERT INTO <class\_name> (<field\_name>) VALUES (field\_value)

Для удаления записей используется команда DELETE:

DELETE FROM <target-name>

Для изменения значения используется UPDATE:

UPDATE Profile SET nick = 'Luca'

Для выбора результатов используется SELECT [22]:

SELECT name, age FROM Account

Для создания ребра графа используется:

CREATE EDGE FROM 'rid' TO 'rid'

Для создания узла графа используется:

?

-) Протоколы взаимодействия

Использует протокол Binary Protocol.

After the connection has been established, a client can Connect to the server or request the opening of a database Database Open. Currently, only TCP/IP raw sockets are supported. For this operation use socket APIs appropriate to the language you're using. After the Connect and Database Open all the client's requests are sent to the server until the client closes the socket. When the socket is closed, OrientDB Server instance frees resources the used for the connection.

The first operation following the socket-level connection must be one of:

1. Connect to the server to work with the OrientDB Server instance
2. Open a database to open an existing database

In both cases a Session ID is sent back to the client. The server assigns a unique Session-Id to the client. This value must be used for all further operations against the server. You may open a database after connecting to the server, using the same Session-Id

<https://orientdb.com/docs/last/internals/Network-Binary-Protocol.html>

После установления соединения клиент может подключиться к серверу или запросить открытие базы данных Database Open. В настоящее время поддерживаются только необработанные сокеты TCP/IP. После подключения и открытия базы данных все запросы клиента отправляются на сервер до тех пор, пока клиент не закроет сокет. Когда сокет закрыт, экземпляр сервера OrientDB освобождает ресурсы, используемые для подключения.

Первая операция, следующая за подключением на уровне сокета, должна быть одной из:

1. Подключение к серверу для работы с экземпляром сервера OrientDB

2. Открытие существующей базы данных

В обоих случаях клиенту отправляется обратно идентификатор сеанса (Session ID). Сервер присваивает клиенту уникальный идентификатор сеанса. Это значение должно использоваться для всех дальнейших операций с сервером [23].

-) Проблемы

https://www.g2.com/products/orientdb/reviews?utf8=%E2%9C%93&filters%5Bnps\_score%5D=5&order=g2\_default

Среди проблем выделяют следующие: плохая документация, плохая обратная совместимость. Так же отмечают огромное количество дефектов, что разработчики обещают исправить в новых версиях, так как сделают основной целью сделать продукт более стабильным [24].

## MarkLogic

-) Основная информация

MarkLogic Server is a document-oriented database developed by MarkLogic. It is a NoSQL multi-model database that evolved from an XML database to natively store JSON documents and RDF triples, the data model for semantics. MarkLogic is designed to be a data hub for operational and analytical data.

<https://en.wikipedia.org/wiki/MarkLogic_Server>

MarkLogic - это документально-ориентированная база данных, разработанная MarkLogic. Это многомодельная база данных NoSQL, которая развилась из базы данных XML для хранения документов JSON и троек RDF [25].

-) Синтаксис языка запросов

Фактически любая БД в MarkLogic — это виртуальная файловая система, с каталогами, разграничением доступа, временными метками и т.д. Каждый файл — документ XML, проиндексированный сервером; поиск осуществляется внутри любого XML с учётом его разметки. В качестве языка для запросов используется XQuery.

Добавление тега в xml

Within the modify() function, Insert allows us to add a new XML tag to a specified XML path location. This XQuery command needs two parameters a new XML tag with the value that will be inserted; and the XML Path where the new XML tag will be published:

Для добавления XML-тег в указанное местоположение XML-пути используется функция modify. Для этой команды требуются два параметра: новый XML-тег со значением, которое будет вставлено; и XML-путь, по которому будет опубликован новый XML-тег:

xml.modify(insert "new\_tag" into "xml\_path(path)")

Заменить тег

Within the modify() function, the Replace parameter allows us to modify the value of a specific XML-tag in the XML data type column or variable. XQuery needs an XML path and a new value with which to update an existing value in the XML tag:

В функции modify() параметр Replace позволяет нам изменять значение определенного XML-тега в столбце или переменной типа данных XML. Нужен путь к XML и новое значение, с помощью которого можно обновить существующее значение в теге XML:

xml.modify(replace value of "xml\_path" with "new\_value")

Удалить тег

Within the modify() function, the Delete parameter allows us to delete XML-tags in an XML column or XML variable. A query needs the XML path to delete the target XML tags. See the example:

В функции modify() параметр Delete позволяет удалять XML-теги в столбце XML или переменной XML. Для удаления целевых XML-тегов запросу требуется XML-путь.

xml\_data.modify('delete (/user/age)')

Поиск

fn:doc

Retrieves a document using a URI supplied as an xs:string, and returns the corresponding document node.

If $uri is the empty sequence, the result is an empty sequence.

If $uri is a relative URI reference, it is resolved relative to the value of the static base URI property from the static context. The resulting absolute URI is promoted to an xs:string.

fn:collection

Returns a sequence of items identified by a collection URI; or a default collection if no URI is supplied.

If $arg is not specified, the function returns the sequence of items in the default collection in the dynamic context.

If the value of $arg is a relative xs:anyURI, it is resolved against the value of the base-URI property from the static context.

If $arg is the empty sequence, the function behaves as if it had been called without an argument.

Для получения значений можно использовать fn:doc, которая извлекает документ, используя URI, указанный в виде xs:string, и возвращает соответствующий узел документа:

fn:doc($uri)

Если $uri - пустая последовательность, результатом будет пустая последовательность.

Если $uri является относительной ссылкой на URI, она определяется относительно значения статического базового свойства URI из статического контекста. Результирующий абсолютный URI преобразуется в xs:string.

Или можно использовать fn:collection, которая возвращает последовательность элементов, идентифицируемых URI коллекции; или коллекцию по умолчанию, если URI не указан.

fn:collection($arg)

Если $arg не указан, функция возвращает последовательность элементов в коллекции по умолчанию в динамическом контексте.

Если значение $arg является относительным xs:anyURI, оно сопоставляется со значением свойства base-URI из статического контекста.

Если $arg - пустая последовательность, функция ведет себя так, как если бы она была вызвана без аргумента [26].

-) Протоколы взаимодействия

MarkLogic uses a proprietary protocol for internal communication between nodes in a cluster. This article is describes the XML Data Query Protocol (XDQP) and related configuration settings.

XDQP is Application protocol that runs at the TCP Layer, using Port 7999 and 7998. Port 7999 is used for Heartbeat check as well as E-node/D-node data communication. Port 7998 is mainly used for communication between clusters for Replication. This article mainly will focus on port 7999 XDQP communication.

XDQP Connections are created when MarkLogic Server starts up and, barring any exception condition, remains persistent until the Server is shutdown. During steady state cluster operations, no XDQP connections are being opened or closed.

During startup, For each node in cluster, MarkLogic Server will create 3 connections to other Node's on their port 7999. Symmetrically other node will also create 3 connections to first Node's 7999 port. Therefore, each node in the cluster will have 6x(N-1) connections created for Cluster with N number nodes. Multiple connection channels (3) for XDQP port 7999 are intended for Performance reasons and number of channels are NOT configurable.

MarkLogic Server sends heartbeat message to each node in the cluster every second (NOT configurable). The heartbeat message synchronizes all servers to the same clock (transaction timestamp), keeps a consistent state of the 'quorum', propagates configuration changes, and may carry Query Data. If the heartbeat messages are interrupted for an extended period of time, a host may become disconnected from the cluster (Host Timeout).

Heartbeat messages are cycled across all 3 connection channels; Since MarkLogic sends heartbeat messages to each specific node every second, we would expect to see traffic at least once on each connection channel every 3 seconds.

Heartbeats from other hosts are dropped if clock skew is detected beyond acceptable limit, causing hosts to disconnect.

New connections are also rejected if clock skew beyond acceptable limit are detected (host timeout). Other Host is declared down if the problem persists with new connect.

<https://help.marklogic.com/knowledgebase/article/View/xml-data-query-protocol-xdqp>

MarkLogic использует протокол запроса данных XML (XDQP)для внутренней связи между узлами в кластере.

XDQP - это прикладной протокол, который работает на уровне TCP, используя порты 7999 и 7998.

Соединения XDQP создаются при запуске MarkLogic Server и, за исключением каких-либо исключительных условий, остаются постоянными до завершения работы сервера. Во время работы кластера в стационарном режиме соединения XDQP не открываются и не закрываются.

Во время запуска для каждого узла в кластере сервер MarkLogic создаст 3 подключения к другим узлам на их порту 7999. Симметрично другой узел также создаст 3 подключения к порту 7999 первого узла.

Сервер MarkLogic отправляет сообщение heartbeat на каждый узел кластера каждую секунду. Сообщение heartbeat синхронизирует все серверы с одинаковыми часами, поддерживает согласованное состояние "quorum", распространяет изменения конфигурации и может содержать данные запроса.

Сообщения о “сердцебиении” передаются циклически по всем 3 каналам подключения. Если сообщения о “сердцебиении” прерываются на длительный период времени, узел может отключиться от кластера (тайм-аут узла).

Новые подключения отклоняются, если обнаруживается превышение допустимого предела синхронизации (тайм-аут хоста). Другой хост объявляется недоступным, если проблема сохраняется при новом подключении [27].

-) Проблемы

https://www.g2.com/products/marklogic/reviews?utf8=%E2%9C%93&filters%5Bnps\_score%5D=5&order=g2\_default

Среди проблем выделяют очень высокую стоимость лицензирования и необходимость в большом объем пространства, необходимого для хранения данных. Так же отмечают большой порог входа для людей, которые привыкли к другим БД [28].

## Azure Cosmos DB

-) Основная информация

Azure Cosmos DB is a globally distributed, multi-model database service offered by Microsoft. It is designed to provide high availability, scalability, and low-latency access to data for mission-critical applications. Unlike traditional relational databases, Cosmos DB is a NoSQL database, which means it can handle unstructured and semi-structured, in addition to structured, data types.

<https://en.wikipedia.org/wiki/Cosmos_DB>

Azure Cosmos DB - это глобально распределенная служба многомодельных баз данных, предлагаемая Microsoft. Она предназначена для обеспечения высокой доступности, масштабируемости и доступа к данным с низкой задержкой для критически важных приложений [29].

-) Синтаксис языка запросов

Azure Cosmos DB поддерживает несколько api, которые используются разные языки запросов.

Для Azure Cosmos DB for NoSQL используется язык SQL.

Для Azure Cosmos DB for PostgreSQL так же используется SQL.

Для Azure Cosmos DB for MongoDB используется тот же язык, что и в MongoDB. Подробный его синтаксис был описан ранее [30].

-) Протоколы взаимодействия (тсп айпи и т.д.)

Default value is Https.

This setting is not used when ConnectionMode is set to Gateway. Gateway mode only supports HTTPS. For more information, see Connection policy: Use the TCP protocol.

<https://learn.microsoft.com/en-us/dotnet/api/microsoft.azure.documents.client.connectionpolicy.connectionprotocol?view=azure-dotnet>

https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/sdk-connection-modes

Про режимы подключения и их назначения <https://learn.microsoft.com/en-us/dotnet/api/microsoft.azure.documents.client.connectionmode?view=azure-dotnet>

Протокол подключения можно выбрать TCP или HTTPS [31].

-) Проблемы (а как найти?)

<https://www.g2.com/products/azure-cosmos-db/reviews>

Среди проблем отмечается высокая цена использования из-за чего сложнее планировать проект. Так же возникают неудобства из-за ограничений аккаунта, который необходим для использования [32].

# Выводы

Было произведено изучение мультимодельных данных, их типы построения. Так же был произведён анализ некоторых мультимодельных баз данных, который включал в себя изучение синтаксиса, определение протокола, через который работает СУБД, и выявлены проблемы, возникающие при работе с каждой из рассмотренных СУБД.

# Литература

1. [Электронный ресурс] – 2023 г. – Режим доступа: https://en.wikipedia.org/wiki/Multi-model\_database, свободный.
2. The Multi-model Databases – A Review / Ewa Płuciennik & Kamil Zgorzałek; — Springer, Cham: Communications in Computer and Information Science, vol. 716, 2017. — 141–152 с.
3. Key-Value stores:a practical overview / Marc Seeger; — Stuttgart, Germany, 2009.
4. Survey of graph database models / Renzo Angles, Claudio Gutierrez; — ACM Computing Surveys, vol. 40, 2008. — 1–39 с.
5. An Approach for Implementing Online Analytical Processing Systems under ColumnFamily Databases / Abdelhak Khalil and Mustapha Belaissaoui; — IAENG International Journal of Applied Mathematics, vol. 53, 2023. — 31–39 c.
6. [Электронный ресурс] – 2023 г. – Режим доступа: <https://www.mongodb.com/document-databases>, свободный.
7. [Электронный ресурс] – 2023 г. – Режим доступа: <https://docs.arangodb.com/3.11/concepts/data-models/>, свободный.
8. [Электронный ресурс] – 2023 г. – Режим доступа: <https://orientdb.org/docs/3.0.x/datamodeling/Tutorial-Document-and-graph-model.html>, свободный.
9. [Электронный ресурс] – 2023 г. – Режим доступа: <https://learn.microsoft.com/en-us/azure/cosmos-db/introduction>, свободный.
10. Multi-model Databases : A New Journey to Handle the Variety of Data / Lu , J & Holubová; — ACM Computing Surveys , vol. 52 , no. 3 , 2019. — 55 с.
11. [Электронный ресурс] – 2023 г. – Режим доступа: https://habr.com/ru/articles/462493/
12. Polyglot database architectures = polyglot challenges / Lena Wiese; — Gottingen, Germany: CEUR Workshop Proceedings, vol. 1458, 2015. — 422-426 с.
13. MongoDB: The Definitive Guide / Shannon Bradshaw, Eoin Brazil and Kristina Chodorow. — Boston: O'Reilly Media, Inc., 2019. — 511 с.
14. [Электронный ресурс] – 2023 г. – Режим доступа: <https://www.mongodb.com/docs/manual/tutorial/query-documents/>, свободный.
15. [Электронный ресурс] – 2023 г. – Режим доступа: <https://www.mongodb.com/docs/manual/reference/mongodb-wire-protocol/>, свободный.
16. [Электронный ресурс] – 2023 г. – Режим доступа: <https://www.g2.com/products/mongodb/reviews>, свободный.
17. [Электронный ресурс] – 2023 г. – Режим доступа: <https://ru.wikipedia.org/wiki/ArangoDB>, свободный.
18. [Электронный ресурс] – 2023 г. – Режим доступа: <https://docs.arangodb.com/3.11/aql/fundamentals/syntax/>, свободный.
19. [Электронный ресурс] – 2023 г. – Режим доступа: <https://docs.arangodb.com/3.11/develop/http-api/general-request-handling/>, свободный.
20. [Электронный ресурс] – 2023 г. – Режим доступа: <https://www.g2.com/products/arangodb/reviews>, свободный.
21. [Электронный ресурс] – 2020 г. – Режим доступа: <https://en.wikipedia.org/wiki/OrientDB>, свободный.
22. [Электронный ресурс] – 2023 г. – Режим доступа: <https://orientdb.org/docs/3.2.x/>, свободный.
23. [Электронный ресурс] – 2023 г. – Режим доступа: <https://orientdb.com/docs/last/internals/Network-Binary-Protocol.html>, свободный.
24. [Электронный ресурс] – 2023 г. – Режим доступа: <https://www.g2.com/products/orientdb/reviews>, свободный.
25. [Электронный ресурс] – 2023 г. – Режим доступа: <https://en.wikipedia.org/wiki/MarkLogic_Server>, свободный.
26. [Электронный ресурс] – 2023 г. – Режим доступа: <https://www.w3.org/TR/xpath-functions-31/>, свободный.
27. [Электронный ресурс] – 2023 г. – Режим доступа: <https://help.marklogic.com/knowledgebase/article/View/xml-data-query-protocol-xdqp>, свободный.
28. [Электронный ресурс] – 2023 г. – Режим доступа: <https://www.g2.com/products/marklogic/reviews>, свободный.
29. [Электронный ресурс] – 2023 г. – Режим доступа: <https://en.wikipedia.org/wiki/Cosmos_DB>, свободный.
30. [Электронный ресурс] – 2023 г. – Режим доступа: <https://learn.microsoft.com/en-us/azure/cosmos-db/choose-api>, свободный.
31. [Электронный ресурс] – 2023 г. – Режим доступа: <https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/sdk-connection-modes>, свободный.
32. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.g2.com/products/azure-cosmos-db/reviews, свободный.