MongoDB — документоориентированная СУБД с открытым исходным кодом, **не требующая описания схемы таблиц**. Класс - **NoSQL**, использует **JSON-подобные документы** и схему базы данных. Написана на языке **C++**.

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

История

Компания ПО **10gen** начала разработку MongoDB в **2007** году как компонент планируемой платформы в качестве сервисного продукта. В **2009** году компания перешла на модель разработки с открытым исходным кодом, а компания предлагает коммерческую поддержку и другие услуги. В **2013** году **10gen изменил** свое название на **MongoDB Inc**.

Термины

Пример. Мы хотим сохранить контакт-лист с телефона. Понятно, что есть данные, которые хорошо кладутся в одну реляционную табличку: Фамилия, Имя и т.д. Но если посмотреть на телефоны или email-адреса, то у одного человека их может быть несколько. Если подобное хранить в хорошем реляционном виде, то нам неплохо бы это хранить в отдельных таблицах, потом это всё собирать JOIN, что менее удобно, чем хранить это всё в одной коллекции, где находятся иерархические документы.

SQL	MongoDB
Таблица	Коллекция
Строка	Документ
Колонка	Поле
SQL	CRUD

В обоих случаях мы говорим о **базах данных**, но то, что мы называем **таблицей** в реляционной базе данных, часто в нереляционной называется **коллекцией**. То, что в MySQL — **колонка**, в MongoDB — **поле**. И так далее.

С точки зрения использования **JOIN**, в MongoDB **нет такого понятия** — это вообще понятие из реляционной структуры. Там мы либо делаем встроенный документ, что близко к концепту денормализации, либо мы просто сохраняем идентификатор документа в каком-то поле, называем это ссылкой и дальше ручками выбираем данные, которые нам нужны.

Что касается доступа: там, где мы к реляционным данным используем язык **SQL**, в MongoDB и многих других NoSQL-базах данных используется такой стандарт, как **CRUD**. Этот стандарт говорит, что есть операции для **создания, чтения, удаления и обновления** документов.

Преимущества

Нет строгой схемы документов. Надо что-то поменять – меняем

Массивы, вложенные документы и т.д.

Язык запросов JSON.

Горизонтальное масштабирование шардингом (отдельные части данных хранятся на разных серверах). Пользователь выбирает ключ шарда, который определяет как данные в коллекции будут распределены. Данные разделятся на диапазоны (в зависимости от ключа шарда) и распределятся по шардам.

Может хранить неструктурную информацию (двоичные данные)

Файловое хранилище GridFS. MongoDB может быть использован в качестве файлового хранилища с балансировкой нагрузки и репликацией данных.

Grid File System поставляется вместе с драйверами MongoDB. MongoDB предлагает разработчикам функции для работы с файлами и их содержимым. GridFS разделяет файл на части и хранит каждую часть как отдельный документ.

Недостатки

MongoDB гарантирует **ACID в том же документе**. Неспособность реализовать свойства ACID означает, что база данных не обеспечивает **долговечность, целостность, согласованность и изоляцию**, необходимые для **транзакций**. Возможно, что в будущих версиях это будет решено.

Отсутствие понятия изоляции

Блокировка БД происходит на уровне документа, т.е. при изменении одного документа, другие документы также можно изменять.

Отсутствие понятия транзакции

Атомарность гарантируется только на уровне документа.

Проблемы с масштабируемостью

Проблемы с производительностью появляются тогда, когда объем данных превышает 100 ТБ

Нет оператора JOIN

Обычно данные могут быть организованы более денормализованным способом, но на разработчиков ложится дополнительная нагрузка по обеспечению непротиворечивости данных.

Максимальный размер документа – 16 МБ

Для чего?

Очень быстрая разработка. Потому что всё можно постоянно менять, не нужно постоянно заботиться о строгом формате документа.

С MongoDB хорошо делать приложения, у которых очень ограниченный цикл жизни. То есть если мы делаем приложение, которое живёт недолго, например, **сайт для запуска фильма** или **олимпиады**. Мы пожили **несколько месяцев** после этого, и это приложение практически не используется. Если приложение живёт дольше, то тут уже **другой вопрос**.

Обработка больших объемов данных

Хранение и регистрация событий

Мобильные приложения

Проекты с гибкими методологиями разработки (Agile, Scrum, etc)

Типы данных

Пример документа

```
{
  firstName: „John“,
  lastName: „Smith“,
```

```
  birthDay: {
    day: 10,
    month: 12,
    year: 1990
  },
  phones: [ „123-456-78-90“, „123-456-78-90“ ]
}
```

CRUD

Что касается доступа: там, где мы к реляционным данным используем язык **SQL**, в MongoDB и многих других NoSQL-базах данных используется такой стандарт, как **CRUD**. Этот стандарт говорит, что есть операции для **создания, чтения, удаления и обновления** документов.

Create

SQL	CRUD
INSERT INTO book („ISBN“, „title“, „author“) VALUES („123456789“, „Full stack JS“, „Colin“);	db.book.insert({ ISBN: „123456789“, title: „Full stack JS“, author: „Colin“ })

Read

SQL	CRUD
SELECT title FROM book WHERE price > 10;	db.book.find({ price: { $gt: 10} }, { _id: 0, title: 1 })

Update

SQL	CRUD
UPDATE book SET price = 19.99 WHERE ISBN = ‘123456789’	db.book.update({ ISBN: “123456789” }, { $set: { price: 19.99 } })

Delete

SQL	CRUD
DELETE FROM book WHERE publisher_id = „SP001“	db.book.remove({ „publisher.id“: „SP001“ })

Count

SQL	CRUD
SELECT COUNT(1) FROM book WHERE publisher_id = „SP001“	db.book.count({ „publisher.id“: „SP001“ })