



MongoDB является одной из самых популярных нереляционных, документо-ориентированных баз данных. Рассмотрим основные понятия MongoDB:

1. База данных

- Контейнер для коллекций (аналог базе данных с таблицами в SQL-БД): База данных в MongoDB служит контейнером для коллекций. Она может содержать множество коллекций, не связанных друг с другом.

2. Коллекция

- Аналог таблицы: Коллекция в MongoDB аналогична таблице в реляционных базах данных. Она представляет собой группу документов.
- Динамическая схема (т.е. не статичная структура таблицы, как в SQL-БД): В отличие от реляционных баз данных, коллекции в MongoDB не требуют заранее определенной схемы. Это означает, что документы в одной коллекции могут иметь разные поля.

3. Документ

- Основная единица данных (аналог строки в SQL-БД): В MongoDB, основной единицей хранения данных является документ.
- Формат BSON: Документы хранятся в формате BSON (Binary JSON), который расширяет стандартный JSON добавлением дополнительных типов данных, таких как `date` и `binary` .

4. Поле

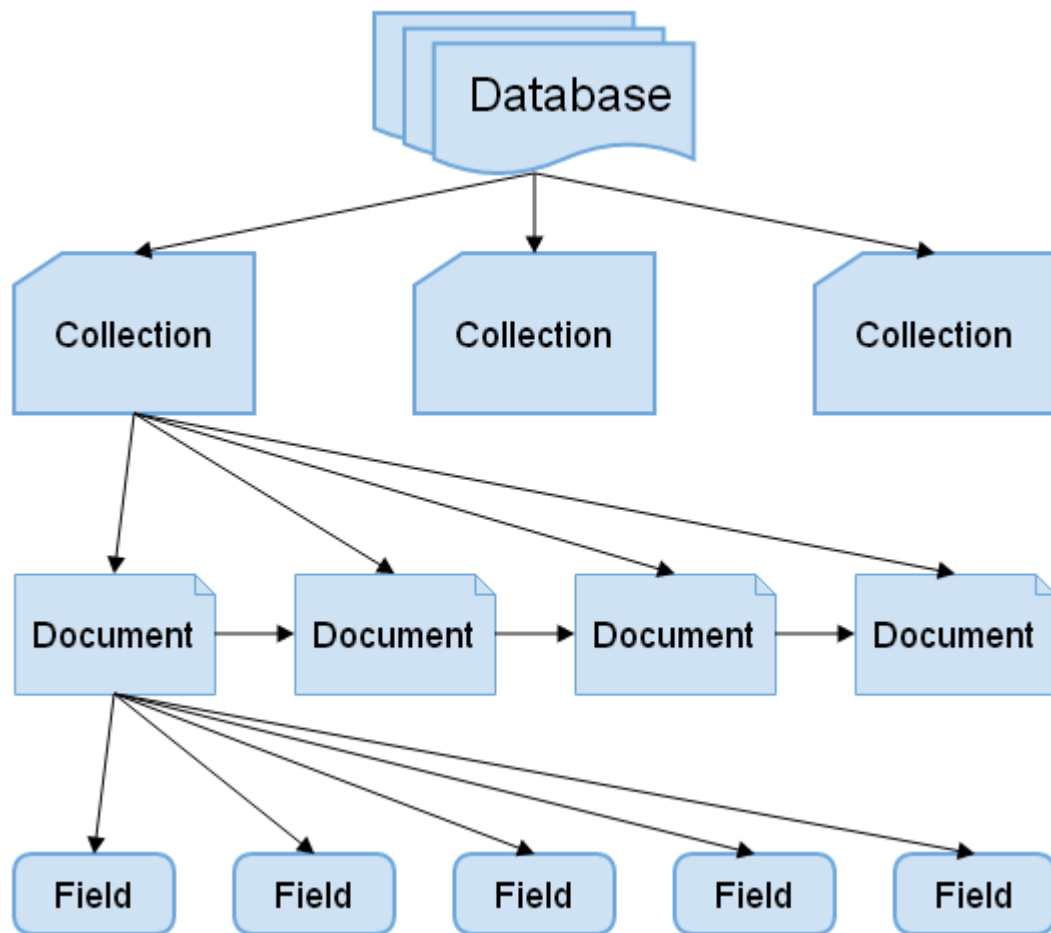
- Элемент данных в документе: Поле в документе MongoDB аналогично столбцу в реляционной таблице. Оно представляет собой пару ключ-значение.

5. ObjectId

- Уникальный идентификатор (аналог PK ключа в SQL-БД): Каждый документ в MongoDB автоматически получает поле `_id` , которое уникально идентифицирует документ в коллекции. Если при создании документа `_id` не указывается, MongoDB генерирует его автоматически в форме ObjectId.

7. Вложенность и массивы

- Вложенные документы: Документы могут содержать вложенные документы, позволяя создавать иерархические структуры данных.
- Массивы: Поля могут включать в себя массивы значений или даже массивы вложенных документов.



Как же нам спроектировать структуру данных для MongoDB?

Для NoSQL решений процесс проектирования структур данных выглядит похожим до момента реализации физической модели.

То есть, представим, что вы (по ранее изученному в нашем курсе, процессу проектирования SQL баз данных) уже подготовили логическую модель, до момента нормализации данных.

Допустим, нам нужно сделать так, чтобы сущности Клиент, Бронь, Номер, были в NoSQL решениях для MongoDB.

Тогда вам остаётся выполнить:

Шаг 1: Анализ сценариев использования и доступа к данным

Первый шаг — глубокое погружение в бизнес-логику и требования к данным. Необходимо понять, какие данные важны, как они будут использоваться, и какие запросы будут наиболее частыми. Это поможет определить, какие данные следует группировать вместе, а какие держать отдельно. В контексте нашего примера, важно определить, как часто клиенты обращаются к своим бронированиям и какие операции чаще всего выполняются с номерами отеля.

Шаг 2: Определение структуры документов и связей между документами (при сложных структурах можно разделить на два шага)

На этом шаге вы преобразуете вашу логическую модель в набор документов NoSQL. Здесь ключевым является решение о том, какие данные следует включать непосредственно в документ, а какие оставить для ссылок или вложенных структур. Например, каждый документ клиента может содержать базовую информацию о клиенте и массив идентификаторов или вложенных документов бронирований.

Также вам нужно определить, как будут управляться отношения между различными сущностями в вашей модели. В NoSQL обычно используются ссылки между документами или вложенные документы. Например, в документе бронирования может быть ссылка на документ номера отеля, что позволит легко получить информацию о номере при запросе бронирования.

Шаг 3: Денормализация и оптимизация для частых запросов

Денормализация — это процесс, при котором вы дублируете некоторую информацию в разных местах для улучшения производительности запросов. Например, если запросы на проверку доступности номера часты, может иметь смысл хранить информацию о статусе доступности номера непосредственно в документе бронирования.

Шаг 4: Гибкость и адаптация схемы

Один из ключевых моментов при работе с NoSQL — это гибкость схемы данных. Вы должны оставить пространство для будущих изменений и расширений, позволяя документам в коллекции иметь различные поля и структуры. Это не только обеспечивает гибкость при изменении бизнес-требований, но и позволяет легко интегрировать новые типы данных.