ЛЕКЦИЯ 9

Графовые и документоориентированные БД

План занятия

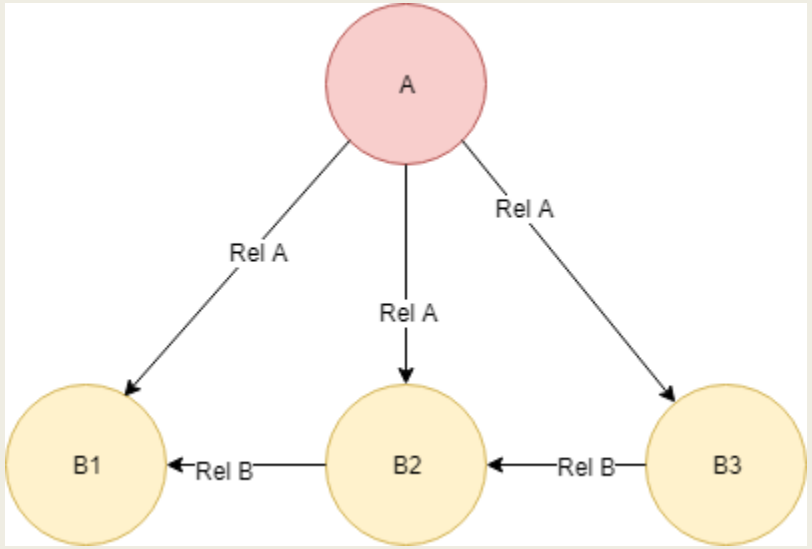
- Графовые СУБД
- Пример: Neo4j
- Документоориентированные СУБД
- Пример: MongoDB

Графовые СУБД

- сейчас большая часть данных существует в форме отношений между объектами, и чаще всего отношения между данными являются более ценными, чем сами данные
- реляционные СУБД хранят высокоструктурированные данные, они чаще используются для хранения структурированных данных и не хранят связи между данными

Графовые БД

- используются для моделирования данных в виде графа
- узлы графа - сущности, отношения - ассоциация этих узлов



Когда использовать

- Нынешнее применение графовых БД:
  - *социальных сети*
  - *системах рекомендаций (с этим товаром часто покупают...)*
  - *обработка пользовательских данных, корреляция данных из разных источников (информационный след в сети)*
- если в приложении планируется много сущностей и связей многие-ко-многим, то это один из признаков, что предпочтительней выбрать графовую СУБД
- приложение с множеством связей, и обход этих связей занимает много времени и ресурсов—стоит присмотреться в сторону графов, т.к. обход связей в них практически ничего не стоит
- их следует использовать для решения задач, которые решаются только графами

Neo4j

- графовая СУБД с открытым исходным кодом, реализована на Java
- приложение может взаимодействовать с БД по одному из протоколов:
  - *HTTP / HTTPS*
  - *BOLT (Собственный протокол Neo Technology) - работает через соединение TCP или WebSocket*

Neo4j состоит из:

- Вершины (узел)
- Свойства
- Отношения
- Метки (label)

Узел (вершина) и его свойства

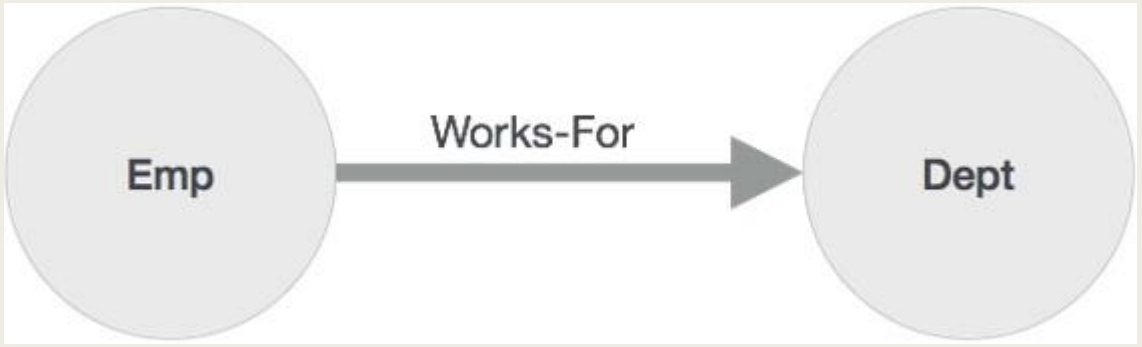
- фундаментальная единица Графа
- содержит свойства с парами ключ-значение

Node Name = «Employee»

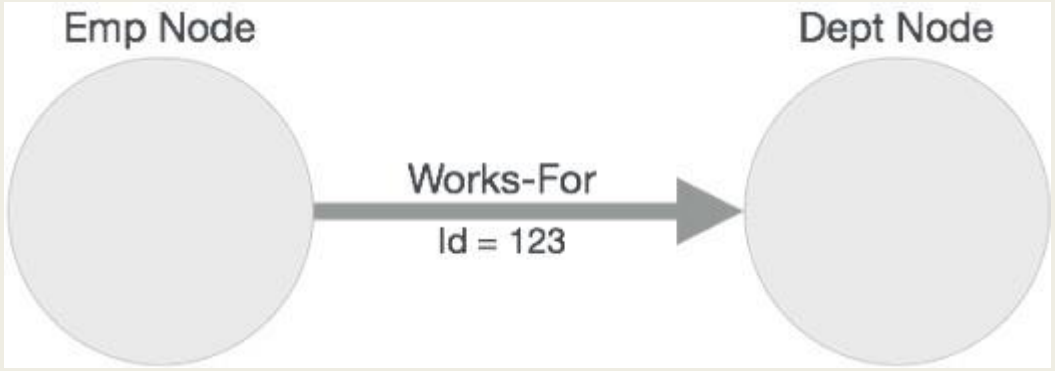
Свойство — это пара ключ-значение для описания узлов и отношений графа

Отношение

- соединяет два узла



- Emp и Dept — два разных узла. «WORKS_FOR» — это отношение между узлами Emp и Dept
- отношения также могут содержать свойства в виде пар ключ-значение



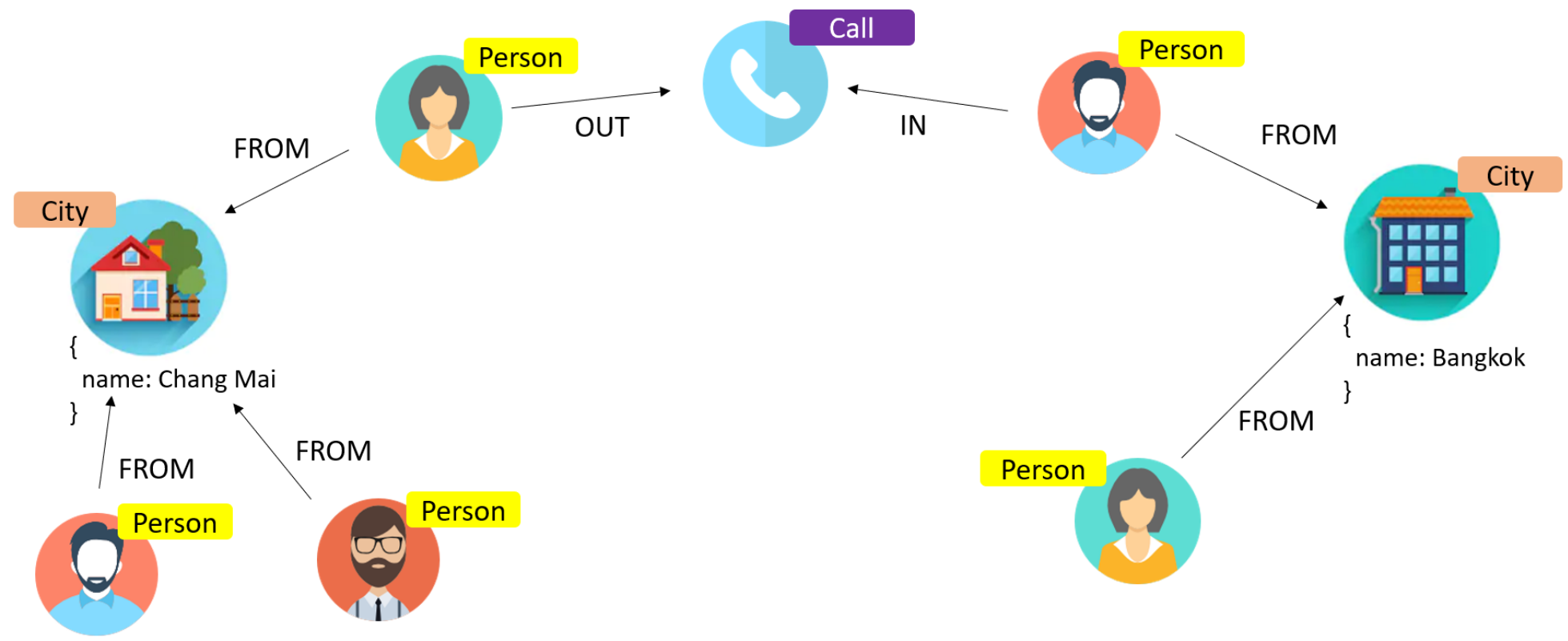
Метки

- Метка связывает общее имя с набором узлов или отношений. Узел или отношение могут содержать одну или несколько меток.
- в предыдущей диаграмме есть два узла
- Левый боковой узел имеет метку: «Emp», а правый боковой узел имеет метку: «Dept».
- Связь между этими двумя узлами также имеет метку: «WORKS_FOR».
- Neo4j хранит данные в свойствах узлов или отношений

Описание узла

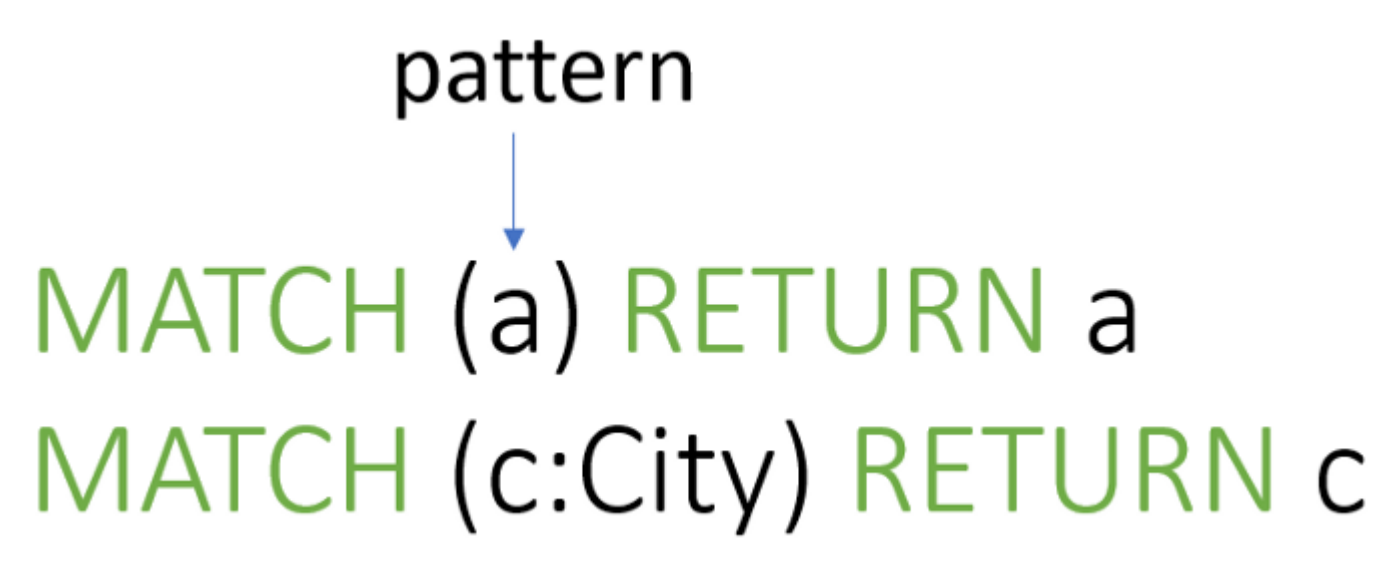

```
{
  "identity": 10,
  "labels": [
    "Movie"
  ],
  "properties": {
"tagline": "Everything that has a beginning has an end",
"title": "The Matrix Revolutions",
"released": 2003
  }
}
```

Пример графа



Язык Cypher

SQL	Cypher
SELECT FROM	MATCH pattern RETURN it
SELECT FROM JOIN	MATCH complex pattern RETURN it



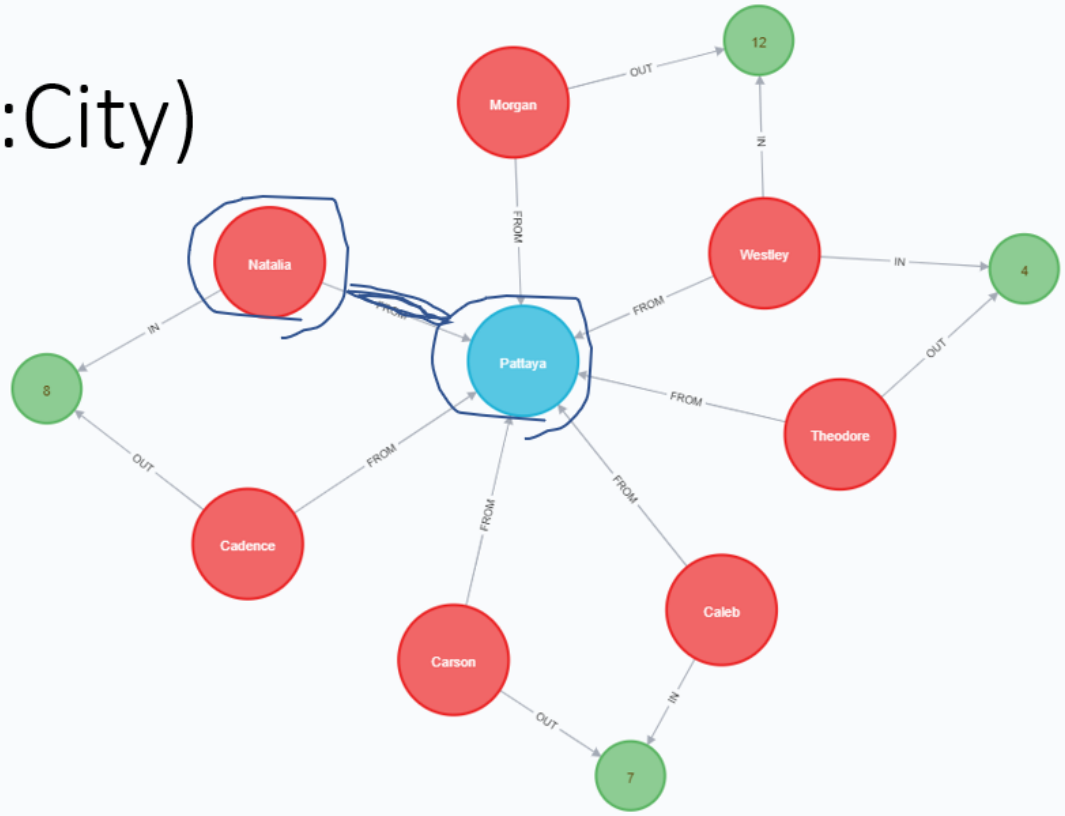
Паттерны и отношения

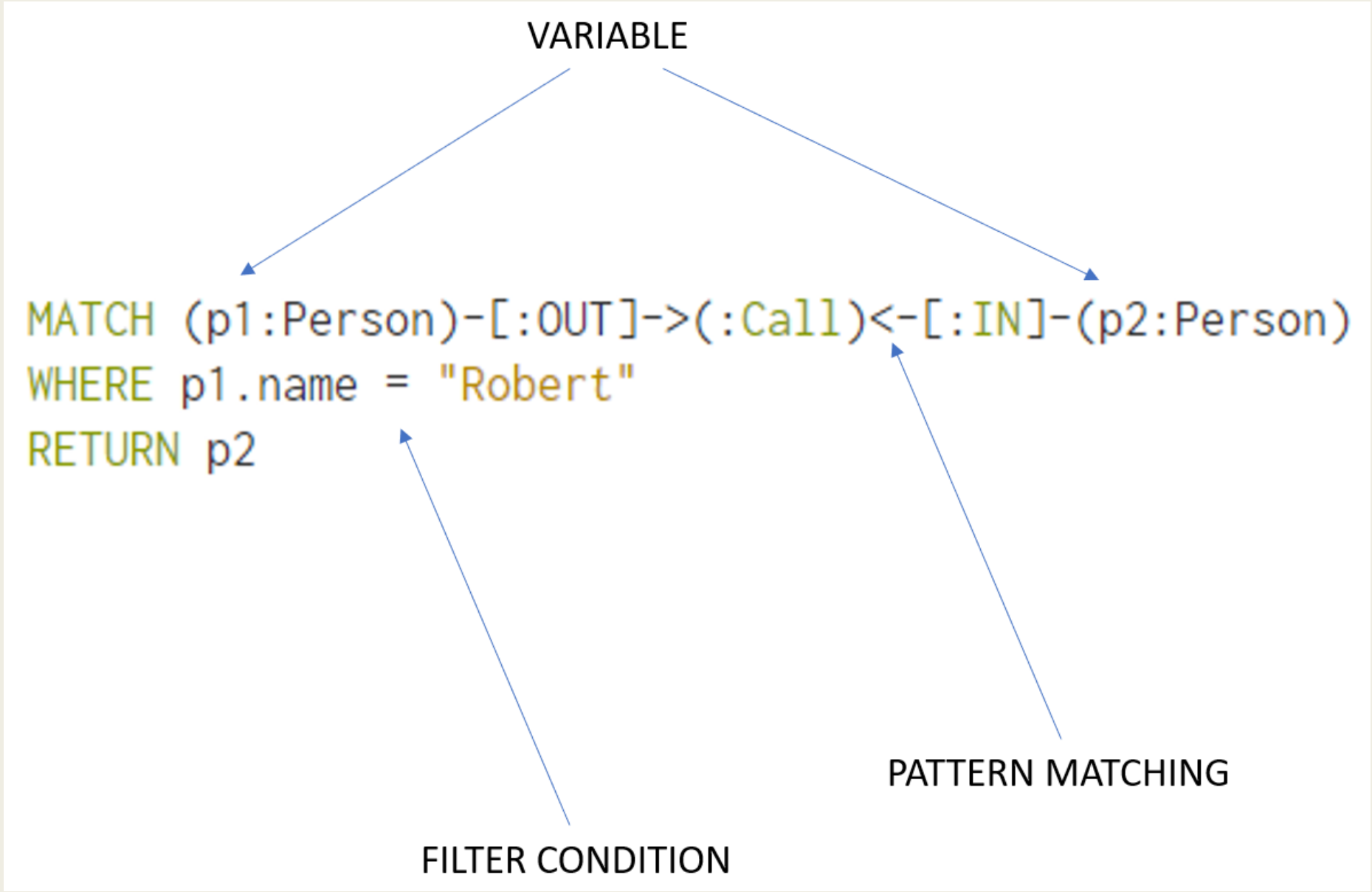
```
()              // all types of nodes
(a)             // all types of nodes via variable “a”
(:Label)        // all nodes of specific type via label
(c:City)        // all nodes of type “City” (labeled as City) via “c”
```

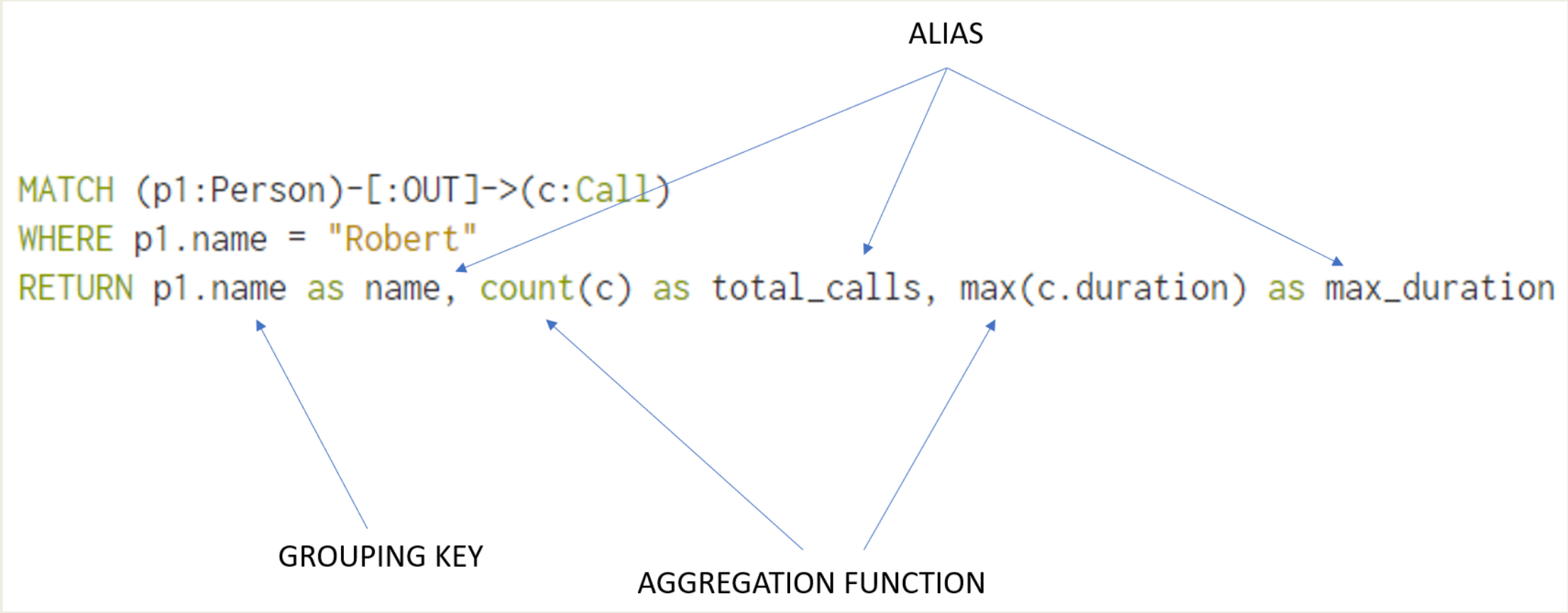
```
-->             // relationship direction
[:REL_TYPE]     // relationship of specified type
-[:FROM]->      // directed relationship of type FROM
```

Пример

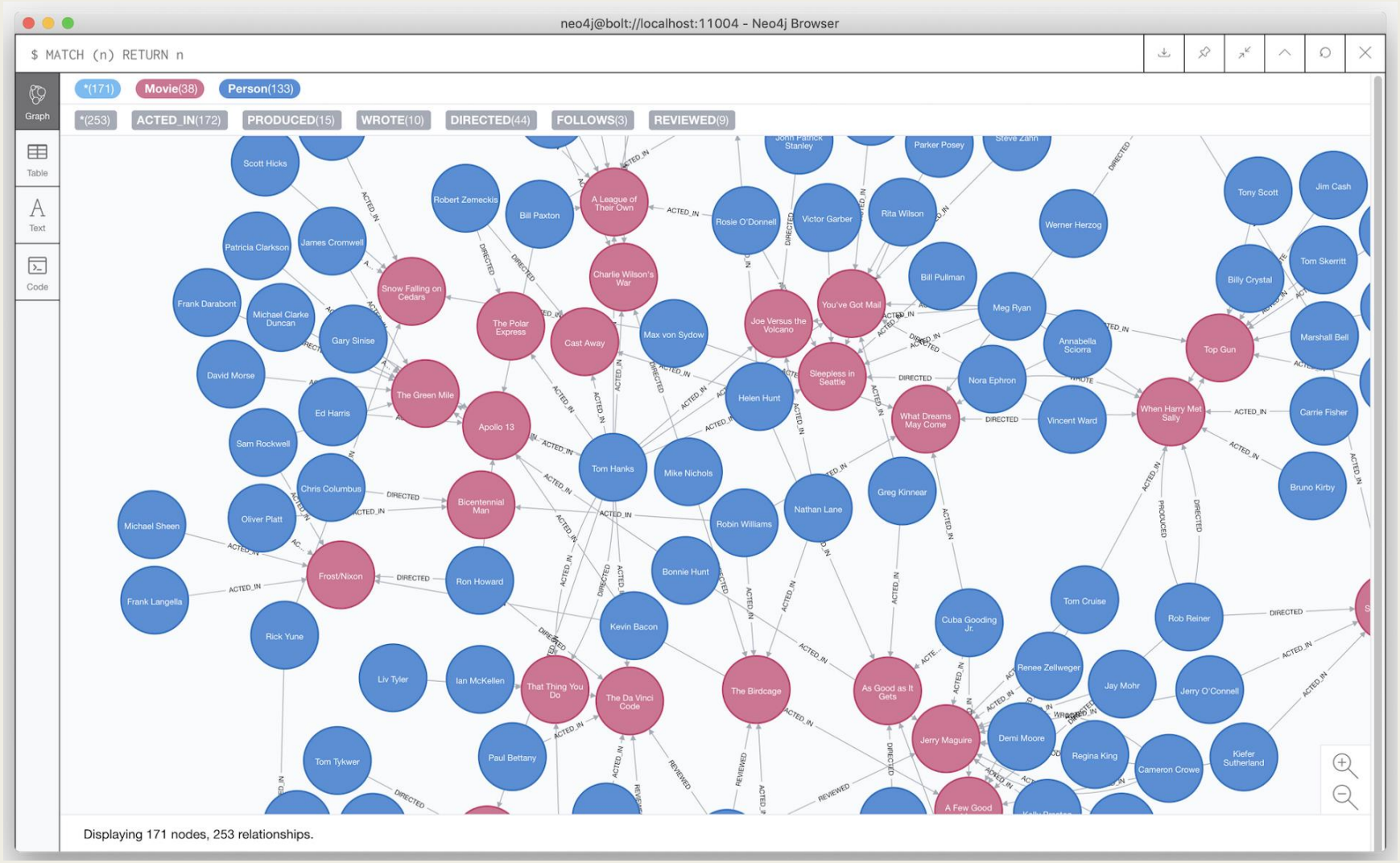
```
MATCH (p:Person)-[:FROM]->(c:City)
WHERE p.name = “Natalia”
RETURN p, c
```




VARIABLE
MATCH (p1:Person)-[:OUT]->(:Call)<-[:IN]-(p2:Person)
WHERE p1.name = "Robert"
RETURN p2
PATTERN MATCHING
FILTER CONDITION


ALIAS
MATCH (p1:Person)-[:OUT]->(c:Call)
WHERE p1.name = "Robert"
RETURN p1.name as name, count(c) as total_calls, max(c.duration) as max_duration
GROUPING KEY
AGGREGATION FUNCTION

Стандартная БД Movie



Neo4j sandbox https://sandbox.neo4j.com/

Преимущества

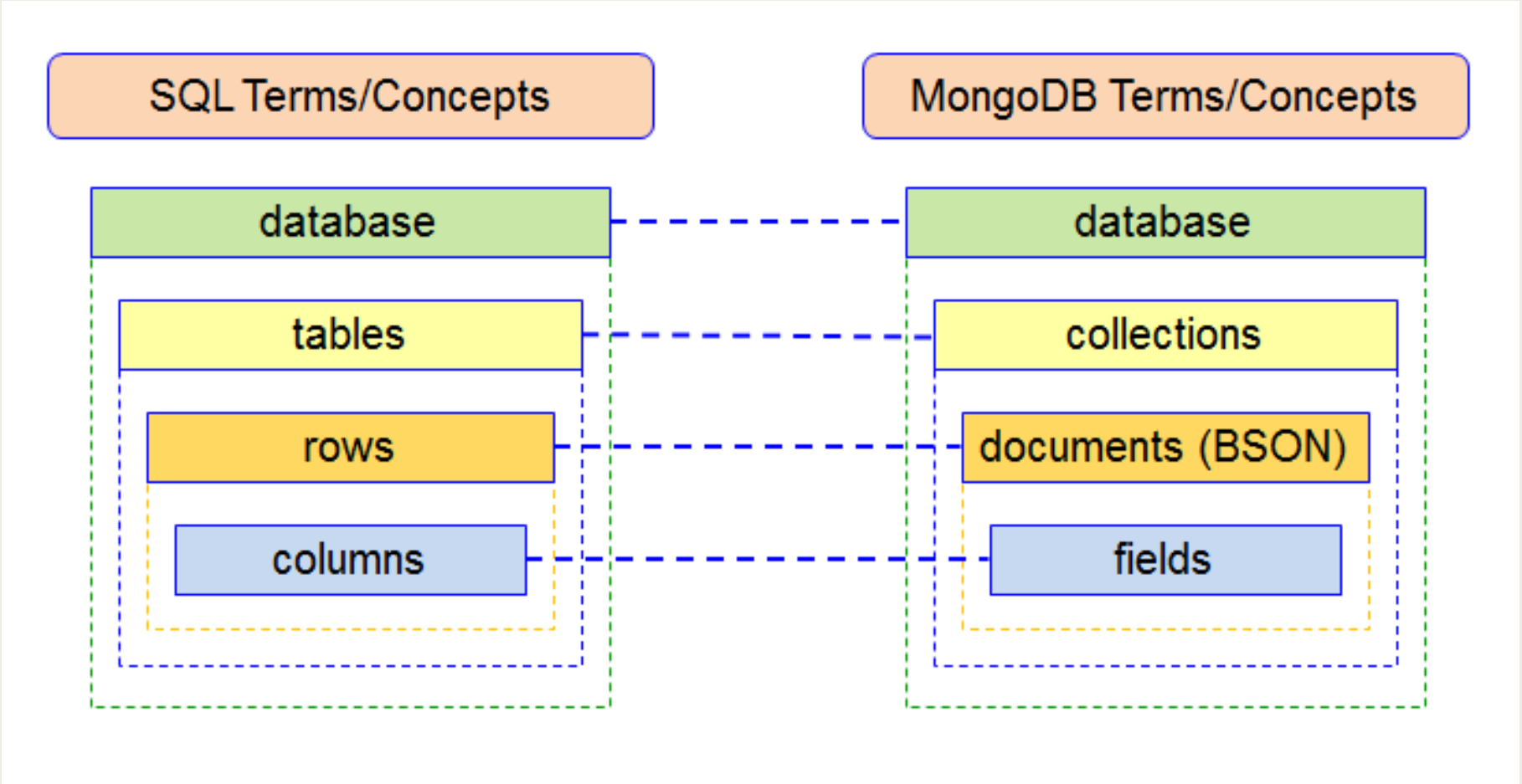
- гибкая модель данных
- простой поиск – легкое перемещение по графу данных
- язык запросов Cypher – прост в освоении и довольно понятен
- можно легко представлять связанные и полуструктурированные данные
- принципы транзакционных систем - ACID

Недостатки

- занимает больше места на диске, чем реляционная БД
- Не любую предметную область можно представить в виде графа

Реляционные vs MongoDB

Реляционная БД	MongoDB
Таблица	Коллекция
Строка	Документ
Столбец	Поле
Объединение таблиц	Встроенные документы (embedded)
Первичный ключ (primary key)	Первичный ключ (primary key). По умолчанию MongoDB генерирует Default key_id


SQL Terms/Concepts
MongoDB Terms/Concepts
database
tables
rows
columns
database
collections
documents (BSON)
fields

Пример документа

```
{
   _id: ObjectId(7bf78ad8902c)
   title: 'MongoDB',
   description: 'Simple MongoDB Database',
   by: 'proselyte',
   url: 'proselyte.net',
   tags: ['proselyte tutorials', 'NoSQL', 'MongoDB'],
   developers: [
      {
         developer:'developer1',
         specialty: 'Java Developer'
      },
      {
         developer: 'developer2'
         specialty: 'C++ Developer'
         }
   ]
}
```

- _id - 12 байтовое шестнадцатеричное число, которое гарантирует уникальность каждого документа
- можно передавать id при вставке данных в документ
- или MongoDB генерирует уникальный id автоматически для каждого документа
- первые 4 цифры – текущее время, следующие 3 – id компьютера, следующие 2 – id процесса на сервере MongoDB, а последние 3 - порядковое значение

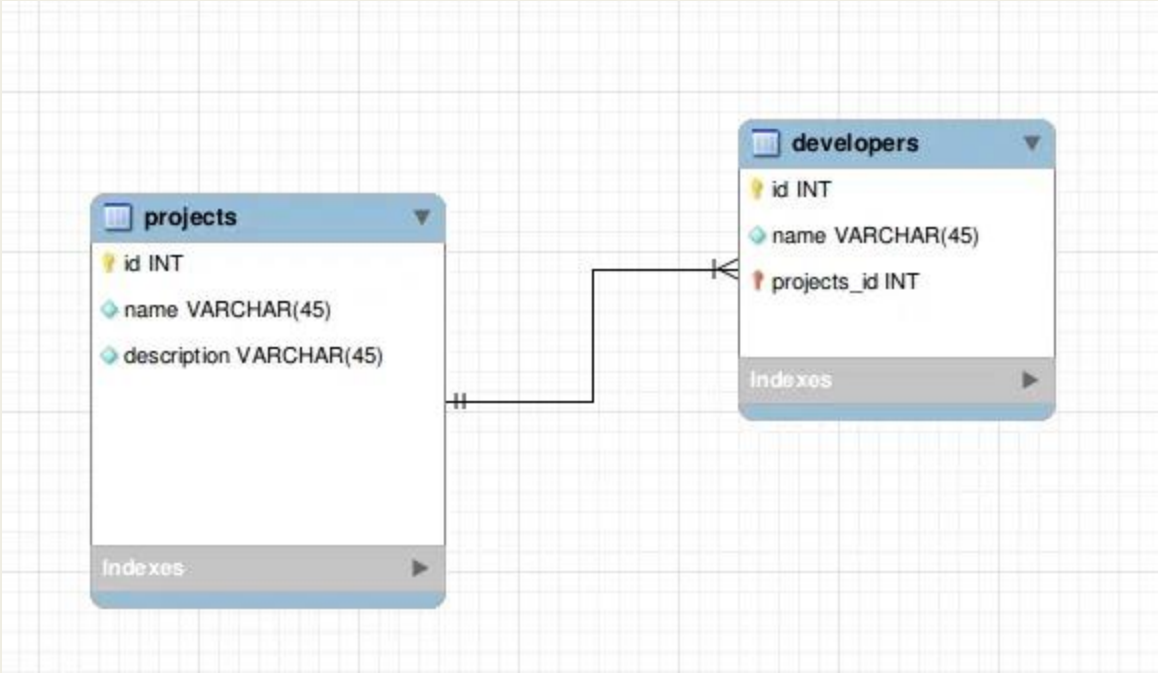
Рекомендации по моделированию схемы:

- комбинировать объекты в одном документе, если предполагается использовать их вместе (статья + комментарии)
- дублировать данные, так как время вычислений "дороже", чем место на диске
- делать join-ы во время записи, а не во время чтения
- использовать сложные агрегации в схеме
- оптимизировать схему под наиболее частые случаи

Пример

- например, нужно создать БД для хранения проектов, разработчиков, которые работают над этими проектами
- требования:
  - *каждый проект имеет уникальное имя и описание*
  - *каждый проект имеет много разработчиков*

Реляционная схема



Коллекция в MongoDB

```
{
   _id: PROJECT_ID
   name: NAME_OF_PROJECT,
   developers: [
      {
         name:'DEVELOPER_NAME',
      },
      {
         name: 'DEVELOPER_NAME'
      }
   ]
}
```

Команды MongoDB

Команда	Описание
show dbs	Список всех БД
use DB_Name	Создание новой БД или переключение на существующую
db	Посмотреть, какая БД сейчас запущена
db.dropDatabase()	Удаление текущей БД
db.createCollection(Name,Options)	Создание коллекции
db.collectionName.drop()	Удаление коллекции

SQL vs CRUD - Insert

- SQL

```sql
INSERT INTO book (
  `ISBN`, `title`, `author`
)
VALUES (
  '9780992461256',
  'Full Stack JavaScript',
  'Colin Ihrig & Adam Bretz'
);
```

- CRUD

```
db.book.insert({
  ISBN: "9780992461256",
  title: "Full Stack JavaScript",
  author: "Colin Ihrig & Adam Bretz"
});
```

SQL vs CRUD Update

```sql
UPDATE book
SET price = 19.99
WHERE ISBN = '9780992461256'
```

```
db.book.update(
  { ISBN: '9780992461256' },
  { $set: { price: 19.99 } }
);
```

SQL vs CRUD - Delete

```sql
DELETE FROM book
WHERE publisher_id = 'SP001';
```

```
db.book.remove({
  "publisher.name": "SitePoint"
});
```

SQL vs CRUD - Search

```sql
SELECT title FROM book
WHERE price > 10;
```

```
db.book.find(
  { price: { &gt;: 10 } },
  { _id: 0, title: 1 }
);
```

SQL vs CRUD - Count

```sql
SELECT COUNT(1) FROM book
WHERE publisher_id = 'SP001';
```

```
db.book.count({
  "publisher.name": "SitePoint"
});
```

SQL vs CRUD - Aggregation

```sql
SELECT format, COUNT(1) AS `total`
FROM book
GROUP BY format;
```

```
db.book.aggregate([
  { $group:
    {
      _id: "$format",
      total: { $sum: 1 }
    }
  }
]);
```

Преимущества

- отсутствие схемы (но можно определить на JSON структуру, где данные будут валидироваться)
- понятная структура каждого объекта.
- легко масштабируется
- MongoDB поддерживает динамические запросы документов (document-based query)
- отсутствие сложных JOIN запросов

Недостатки

- не настолько соответствует требованиям ACID, как реляционные базы данных
- транзакции с использованием MongoDB являются сложными
- нет положений о хранимых процедурах или функциях, поэтому не получится реализовать какую-либо бизнес-логику на уровне базы данных, что можно сделать в реляционных БД

Области применения

- каталог товаров в электронной коммерции.
- блоги и системы управления контентом, особенно те, где много контента, в том числе видео и изображений.
- хранение данных датчиков и устройств
- социальные сети, новостные форумы и другие похожие сценарии
- слабосвязанные данные без четкой схемы хранения
- стартапы и развертывание новых проектов, где структура данных пока неизвестна
- MongoDB часто задействуется как бэкенд больших онлайн-игр (важна масштабируемость)

Где лучше не использовать

- транзакционные системы, приложения, требующие транзакций на уровне базы данных, например банковские приложения
- проекты, где модель данных определена заранее
- хранение сильносвязанных данных

Выводы

- нереляционных БД очень много: документориентированные, ключ-значение, графовые, колоночные и не только
- у каждой СУБД есть свои плюсы и минусы, предпочтительные области применения
- выбор СУБД всегда осуществляется под задачу
- для запросов есть не только SQL